Efficient Maintenance of 2-Hop Labeling Index on Dynamic Small-World Graphs

Yuanyuan Zeng
School of Data Science, The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
zengyuanyuan@cuhk.edu.cn

Yixiang Fang
School of Data Science, The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
fangyixiang@cuhk.edu.cn

Kun Chen
Ant Group
China
ck413941@antgroup.com

Yangfan Li
School of Computer Science and Engineering, Central South University
Changsha, Hunan, China
liyangfan37@csu.edu.cn

Chenhao Ma*
School of Data Science, The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
machenhao@cuhk.edu.cn

ABSTRACT

2-hop labeling has been widely utilized to accelerate the efficiency of online shortest distance queries. Given the nature of frequent changes in real-world graphs, the efficient maintenance of 2-hop labeling index has been extensively studied recently. However, existing methods cannot efficiently process large-scale graphs due to their high time and memory costs, and most of them process large batches of updates sequentially, significantly decreasing efficiency. In this paper, we propose a novel algorithm for maintaining the 2-hop labeling index in a parallel manner, called **M2HL**, which can efficiently handle both edge insertions and deletions. Moreover, we theoretically prove that M2HL maintains both correctness and minimality for the updated 2-hop labeling index. Our experiments on ten large-scale graphs demonstrate that M2HL outperforms the state-of-the-art 2-hop labeling maintenance methods by up to four orders of magnitude in speed while maintaining correctness and minimality, as well as exhibiting strong scalability and low memory usage.

PVLDB Reference Format:
Yuanyuan Zeng, Yixiang Fang, Kun Chen, Yangfan Li, and Chenhao Ma. Efficient Maintenance of 2-Hop Labeling Index on Dynamic Small-World Graphs. PVLDB, 18(7): 2005-2017, 2025.
doi:10.14778/3734839.3734840

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/1fyxytx/VLDB_2025.

1 INTRODUCTION

The small-world networks are characterized by their short average path lengths and skewed degree distributions [20] and are prevalent in real-world scenarios such as social networks, citation networks, biological systems, and communication networks [3, 20, 21]. As a fundamental problem in graph analytics, the shortest distance query $q(s, t)$ reports a minimized length of a path between two nodes s and t in a given graph G, and it has served as a building block in many graph-based areas, such as GPS navigation [26], route planning [1, 13, 30], and community searching [9, 10, 15, 23, 24]. Due to the frequent distance computations and high cost, various index-based methods have been proposed to speedup the computations by building extra auxiliary structures [6, 7, 20, 27, 37, 38]. To our knowledge, 2-hop labeling [11] is widely regarded among all distance labeling methods for its excellent query performance [21, 44].

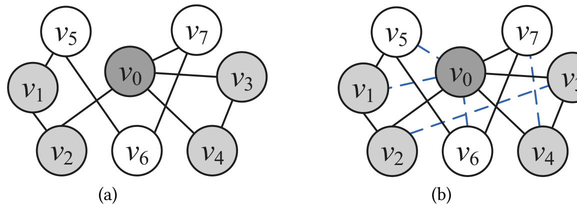

Figure 1: (a) Graph G_S and (b) Graph G_L

Table 1: The 2-hop labeling of G_S and G_L

	The 2-hop labeling L_0 of G_S	The 2-hop labeling L_1 of G_L
v_0	$(v_0, 0)$	$(v_0, 0)$
v_1	$(v_1, 0)$, $(v_0, 2)$	$(v_1, 0)$, $(v_0, 1)$
v_2	$(v_2, 0)$, $(v_0, 1)$, $(v_1, 1)$	$(v_2, 0)$, $(v_0, 1)$, $(v_1, 1)$
v_3	$(v_3, 0)$, $(v_0, 1)$	$(v_3, 0)$, $(v_0, 1)$, $(v_2, 1)$
v_4	$(v_4, 0)$, $(v_0, 1)$, $(v_3, 1)$	$(v_4, 0)$, $(v_0, 1)$, $(v_3, 1)$
v_5	$(v_5, 0)$, $(v_1, 1)$, $(v_0, 3)$	$(v_5, 0)$, $(v_0, 1)$, $(v_1, 1)$
v_6	$(v_6, 0)$, $(v_5, 1)$, $(v_0, 2)$, $(v_1, 2)$	$(v_6, 0)$, $(v_0, 1)$, $(v_5, 1)$
v_7	$(v_7, 0)$, $(v_0, 1)$, $(v_6, 1)$, $(v_5, 2)$	$(v_7, 0)$, $(v_0, 1)$, $(v_4, 1)$, $(v_6, 1)$

The 2-hop labeling approach consists of two typical phases: in the offline phase, it assigns each vertex v a label set $L(v)$ with some hub vertices and distances. During the online phase, the shortest distance between any two reachable vertices u and v can be answered directly by checking their shared hub vertices. For example, Figure 1 can be visualized as a social media platform, such as Twitter or Facebook, where the vertices represent individual users and the edges illustrate their friendship connections. The insertion and deletion of edges connected to each node can signify changes in their friend network. The distance between any two users serves as an indicator of their closeness [3, 5, 32]. The 2-hop labeling of G_S is recorded in Table 1. Here, each entry of a vertex's label set is a key/value pair, i.e., the label $(v_1, 1) \in L(v_2)$ means that the shortest distance between v_1 and v_2 in G_S is 1. This index and its variations have also been used to solve many distance/path

*Chenhao Ma is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 7 ISSN 2150-8097.
doi:10.14778/3734839.3734840

Table 2: Comparison of 2-hop index maintenance algorithms

Algorithm	Correctness	Minimality	Scalability	Parallelism	Edge-batch update
FULPLL	✓	✗	✗	✗	✗
BPCL	✓	✗	✗	✓	✗
M2HL	✓	✓	✓	✓	✓

query problems, such as reachability query [8, 29, 35, 39, 41] and shortest path counting query [28, 31, 45]. Due to its importance and effectiveness, we adopt 2-hop labeling as the base index for studying the index maintenance problem in this paper.

Motivation. In many real-world applications, the graphs are typically dynamic, undergoing discrete changes in their topological structures by either inserting or deleting vertices and edges [14]. For instance, in Twitter, users having 100 followers on average were found to obtain 10% more new followers but lose about 3% of existing followers in a given month [25]. To quickly report the shortest distances in such scenarios, a naive method is to use the state-of-the-art index construction algorithm PSL [20] to reconstruct the 2-hop labeling index. However, even with significant computing resources, building a complete 2-hop labeling index from scratch for each change is extremely time-consuming, especially on large-scale graphs. This is because the speedup achieved by PSL [20] is nonlinear, implying that simply increasing processing power does not lead to a proportional reduction in indexing time. For instance, indexing the Orkut graph with 29 million vertices and 106 million edges costs over 10^4 seconds using 40 cores. Meanwhile, we observe that the numbers of changed vertices and edges are often much less than those of the entire graph. For example, the English Wikipedia saw a 2.6% increase in articles in 2023[1]. Prior studies [17, 18, 47] have shown that the majority of vertex labels remain unchanged in this case. Consequently, full index reconstruction significantly increase computational workloads, leading to substantial time consumption. Based on the above analysis, it is strongly desirable to design efficient algorithms for maintaining the 2-hop labeling index on large dynamic graphs.

Challenges. Maintaining 2-hop labeling index in large dynamic graphs is a challenging task with two primary issues:

(1) **How to maintain correctness and minimality.** The 2-hop labeling index is characterized by its correctness and minimality, with the former guaranteeing the accuracy of query results and the latter striving to minimize the memory space utilized by the index. Graph changes alter the shortest distances between specific vertex pairs, compromising the correctness of index-based queries. Specifically, inserting new edges can reduce distances between vertices, while deleting edges can increase them. Besides, many existing labels can be overshadowed by newly added labels, thereby affecting their minimality. Detailed descriptions of these two properties are provided in Section 2.1.
(2) **How to design efficient and scalable techniques.** Although the numbers of changed vertices and edges are typically much smaller than those of the entire graph, they may still need much computational cost to update labels. In addition, introducing auxiliary structures with high space costs to improve the efficiency of index maintenance is undesirable, particularly for large-scale graphs.

[1]https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

Prior works. Existing methods primarily focus on ensuring the correctness of queries while neglecting minimality. Moreover, we observe that these methods suffer from significant issues of low efficiency and poor scalability. Table 2 compares the representative maintenance methods of 2-hop labeling index. Specifically, as the first work of maintaining 2-hop labeling index, FULPLL [12] suffers from three non-negligible issues: 1) retaining the outdated labels in the edge insertion scenario, thus violating the minimality property; 2) deleting massive irrelevant labels in the edge deletion scenario, thereby significantly increasing time cost; and 3) lacking the parallel optimization strategy, leading to substantial time costs. The state-of-the-art method BPCL [47] incorporates a parallel optimization strategy to reduce indexing time and an auxiliary structure to quickly identify the labels that need to be reconstructed during edge deletions. However, its scalability is poor due to the substantial memory costs associated with the auxiliary structure and labels, and moreover it fails to maintain the minimality of 2-hop labeling index in the edge insertion scenario. Further, both methods handle dynamic edges by sequentially processing each edge, limiting the power of multiple core processing.

Our Approach. To ensure both correctness and minimality, **M2HL** systematically verifies all relevant label entries, addressing both redundant labels introduced by edge insertions and missing labels caused by edge deletions. Specifically:

(1) **For edge insertions**, M2HL first gathers all new labels introduced by the inserted edges to maintain correctness. It then removes any redundant labels that are dominated by these new labels, ensuring minimality and reducing the index memory overhead.
(2) **For edge deletions**, M2HL precisely identifies and removes erroneous labels stemming from the deleted edges. It then inserts missing labels that were previously overshadowed by these erroneous labels, thereby restoring both correctness and minimality. Each erroneous label corresponds to a change in the shortest distance between a specific pair of nodes.

We further provide a theoretical analysis to confirm the correctness and minimality of these methods. Additionally, we introduce effective pruning strategies to minimize irrelevant computations during the index maintenance process:

(1) **For edge insertions**, these strategies ensure that new labels are not dominated by existing ones and exclude irrelevant labels that are unaffected by the update.
(2) **For edge deletions**, the strategies accurately identify all missing labels caused by the removal of erroneous labels, further reducing unnecessary computations.

A key innovation of **M2HL** lies in its ability to process all dynamic edges concurrently. By consolidating updates into rounds without conflicts, M2HL avoids redundant computations that would arise from processing individual edges sequentially. This design also optimizes the use of multiple computing cores, significantly improving overall efficiency.

Contributions. Our principal contributions are listed as follows:

- We identify the core principles of index construction on static graphs and thoroughly analyze the limitations of existing 2-hop labeling maintenance methods.
- We propose M2HL, a novel method for maintaining the 2-hop labeling index in a parallel manner, which theoretically guarantees the correctness and minimality.

- We design effective pruning strategies to significantly reduce irrelevant computations in index maintenance, demonstrating well-bounded memory costs and good scalability.
- Extensive experiments demonstrate the superior performance of M2HL. Particularly, M2HL achieves up to four orders of magnitude speedup compared to state-of-the-art methods.

Roadmap. We review the related works in Section 8 and present the problem of maintaining 2-hop labeling index in Section 2. Section 3 analyzes the existing methods. Sections 4 and 5 develop the index maintenance strategies for the scenarios of edge insertion and deletion, respectively. Section 7 presents the experimental results and Section 6 introduces the extensions of our method on directed and weighted graphs. Finally, we conclude in Section 9.

2 PRELIMINARIES

Let $G(V,E)$ be an undirected graph where V and $E \subseteq V \times V$ are sets of n vertices and m edges, respectively. $N(v,G)=\{u|e(u,v)\in E\}$ denotes the neighbor set of v in G and $deg(v) = |N(v,G)|$ is the degree of v. Given a pair of vertices (s,t), $p(s,t) = \langle v_0 = s, \ldots, v_{k-1}, v_k = t\rangle$ is a path between s and t, where $e(v_i, v_{i+1}) \in E$ for $i \in [0, k-1]$. The distance of $p(s,t)$, denoted by $|p(s,t)|$, is the number of edges. $dist(s,t) = \min_{p(s,t)\in P(s,t)} |p(s,t)|$ denotes the shortest distance between s and t, where $P(s,t)$ is a set of all paths between them.

Besides, we use E^+ and E^- to denote the sets of edges to be inserted and deleted, respectively. G^+ can be obtained by inserting all edges of E^+ into G, while G^- is obtained by deleting all edges in E^- from G. For example, Figure 1 shows two graphs G_S and G_L, where dynamic edges are marked in blue. G_L can be obtained by inserting all dynamic edges into G_S, while G_S can be obtained by deleting all dynamic edges from G_L. Similarly, $N^+_\Delta(v)$ and $N^-_\Delta(v)$ denote the neighbor sets of v in these two types of edge sets. Table 3 summarizes frequently used notations.

Table 3: Notations and meanings.

Notation(s)	Meanings
$G(V,E)$	an undirected graph
$N(v,G)$	the neighbor set of a vertex v in G
$deg(v)$	the degree of a vertex v
$p(s,t)$	a path from the vertex s to the vertex t
$\|p(s,t)\|$	the distance of $p(s,t)$
$L(v), C(v)$	sets of 2-hop label and hub nodes of v
$L_d(v)$	a set of labels whose distances are d
$C_d(v)$	a set of hub nodes in $L_d(v)$
$L_{<d}$	a set of labels whose distances are less than d
V_A	a set of affected vertices
E^+, E^-	sets of edges to be inserted and deleted
G^+, G^-	the updated graphs $G \cup E^+$ and $G \setminus E^-$
$N^+_\Delta(v), N^-_\Delta(v)$	neighbor sets of v in E^+ and E^-
$F(v)$	an auxiliary parameter of v to update the labels

2.1 Overview of 2-hop labeling index

Index structure. The 2-hop labeling index requires building a label set $L(v)$ with $\forall v \in V$, where each entry of $L(v)$ is a key/value pair $(u, dist(u,v))$ with $u \in V$. Here, $C(v) = \{u|(u, dist(u,v)) \in L(v)\}$ denotes the hub set of v. Then, $\bigcup_{v\in V} L(v)$ is a 2-hop labeling index if it satisfies the 2-hop cover constraint below.

Definition 1 (2-hop cover constraint [3, 20]). *A labeling function L satisfies the 2-hop cover constraint if any vertex pair (s,t) satisfies $dist(s,t)=dist(s,w)+dist(w,t)$ with certain $w \in C(s)\cap C(t)$.*

Core properties. The 2-hop labeling index L of any graph G needs to satisfy the following two properties.

- *Correctness* [3, 20]. For any query task $q(s,t)$ with $s,t \in V$, we have $dist(s,t) = \min_{v\in C(s)\cap C(t)} L(s)[v] + L(t)[v]$.
- *Minimality* [3, 20]. For any label $(u, dist(v,u)) \in L(v)$ with $v \in V$, there is a pair of vertices (s,t) such that $dist(s,t) \neq Q(s,t,L(s)\cup L(t))$ if we delete $(u, dist(v,u))$ from $L(v)$.

Query process. Given a query $q(s,t)$ with $s,t \in V$, the shortest distance is $Q(s,t,L(s)\cup L(t)) = \min_{v\in C(s)\cap C(t)} L(s)[v] + L(t)[v]$, where the time complexity is $O(|L(s)| + |L(t)|)$.

Label size. Considering that the label size of v is the number of entries in $L(v)$, denoted as $|L(v)|$, we can conclude that the size of 2-hop labeling index is $O(n \cdot \delta)$, where $\delta = \max_{v\in V} |L(v)|$ is the maximum label size of vertices in G [20].

Example 1. *The 2-hop labeling indices of G_S and G_L are recorded in Table 1. As shown in Figure 1(a), the label $(v_1, 1) \in L(v_5)$ means that the shortest distance between v_5 and v_1 is 1. For $q(v_5, v_3)$ in Figure 1(a), we have $dist(v_5, v_3) = \min_{u\in C(v_5)\cap C(v_3)} L(v_5)[u] + L(v_3)[u] = 4$.*

2.2 Maintenance of 2-hop labeling index

In this section, we formally introduce the problem of maintaining the 2-hop labeling index for dynamic graphs.

Problem 1 (2-hop labeling index maintenance [12, 43, 47]). *Given an undirected graph $G(V,R)$, a minimal 2-hop labeling index L of G, and a set of edges to be inserted E^+ (resp. a set of edges to be deleted E^-), return the updated 2-hop labeling index L^+ for the updated graph G^+ (resp. L^- for G^-).*

Node order. To achieve the minimality of the index, existing methods (e.g., PLL [3] and PSL [20]) often impose an order on the vertices. Specifically, the vertex order is assigned by using a ranking function $r(\cdot)$, which is based on some metrics such as vertex degree and vertex centrality [20]. For each vertex $v \in V$, it is required that $\forall(w, dist(v,w)) \in L(v)$ satisfies $r(w) > r(v)$ with $w \neq v$, which helps to reduce the redundant labels.

Note that the whole index may need to be reconstructed from scratch to achieve the minimality if the node order sequence changes, as only vertices with larger ranks can appear in the label set of vertices with lower ranks. Hence, our solution aims to maintain the correctness and minimality of the updated 2-hop labeling index under the same node order sequence, which helps to reduce the time overhead.

Example 2. *As shown in Figure 1, we have $r(v_0) > \cdots > r(v_{11})$ when ranking the vertices according to their degrees and IDs. Table 1 lists the 2-hop labeling index L_0 and L_1 of Figures 1(a) and 1(b) respectively, where all dynamic edges are marked in blue in Figure 1. We illustrate Problem 1 as follows.*

- ***Edge insertion.*** *Given the minimal 2-hop labeling index L_0 of Figure 1(a) and the edge set E^+ to be inserted, it returns the minimal 2-hop labeling index L_1 of Figure 1(b).*
- ***Edge deletion.*** *Given the minimal 2-hop labeling index L_1 of Figure 1(b) and the edge set E^- to be deleted, it returns the minimal 2-hop labeling index L_0 of Figure 1(a).*

3 ANALYSIS OF EXISTING CONSTRUCTION AND MAINTENANCE ALGORITHMS

In this section, we analyze the state-of-the-art methods about the construction and maintenance of 2-hop labeling index, respectively.

3.1 2-hop labeling index construction methods

We begin with the concepts of peak path and valley path.

Definition 2 (**Peak path and valley path** [34]). *Given a vertex pair (s,t), $p(s,t)$ is a peak path when satisfying $r(u) > \max\{r(s), r(t)\}$ with $\exists u \in p(s,t) \setminus \{s,t\}$; otherwise, $p(s,t)$ is a valley path, since $\forall u \in p(s,t)$ satisfies $r(u) \le \max\{r(s), r(t)\}$.*

Example 3. *As shown in Figure 1, $p_1(v_2, v_6) = \langle v_2, v_1, v_6\rangle$ is a peak path since $r(v_1) > \max\{r(v_2), r(v_6)\}$. In contrast, $p_2(v_0, v_5) = \langle v_0, v_2, v_5\rangle$ is a valley path since $r(v_2) < \max\{r(v_0), r(v_5)\}$.*

Next, we identify two core principles of the index construction based on the above path definitions.

Lemma 1. *Given any two vertices u and v with $r(u) > r(v)$, the label $(u, dist(u,v))$ is inserted into $L(v)$ iff any shortest path between u and v is a valley path.*

Proof. Let $p(u,v)$ be any shortest valley path between u and v, where $\forall v^* \in p(u,v) \setminus \{u\}$ satisfies $r(u) > r(v^*)$. Based on the constraint of node order strategy, the label $(u, dist(u,v))$ needs to be inserted into $L(v)$ to satisfy the 2-hop cover in Definition 1. Otherwise, we have $Q(u, v, L(u) \cup L(v)) \neq dist(u,v)$, thus destroying the correctness property. □

Lemma 2. *Given any two vertices u and v with $r(u) > r(v)$, the label $(u, dist(u,v))$ cannot be inserted into $L(v)$ if there exists at least one shortest peak path between u and v.*

Proof. Assume that $(u, dist(u,v)) \in L(v)$ and $p^*(u,v)$ is a shortest peak path where w is a middle vertex in p^* with the maximal rank value. In this case, we have $p^*(u,v)$ concatenated by two shortest valley paths $p_1^*(u,w)$ and $p_2^*(w,v)$. Based on Lemma 1, the labels $(w, dist(u,w))$ and $(w, dist(v,w))$ are inserted into $L(u)$ and $L(v)$, respectively. Then, we have $dist(u,v) = L(u)[w] + L(v)[w]$ with $w \in C(u) \cap C(v)$, proving that inserting $(u, dist(u,v))$ to $L(v)$ will break the minimality of 2-hop labeling index. Hence, the lemma holds. □

Next, we briefly introduce two most related methods about building the minimal 2-hop labeling index below.

Prune Landmark Labeling (PLL) [3]. This method sequentially performs n rounds of pruned BFS searches to collect labels. In the i-th round, the search process is sourced from v_i and expanded to other vertices u with $r(v_i) > r(u)$. Then, the label $(v_i, dist(u, v_i))$ is inserted into $L(u)$ when satisfying Lemmas 1 and 2. Finally, the 2-hop labeling index can be built after finishing totally n rounds of searches.

Parallel Shortest-distance Labeling (PSL) [20]. PSL proposes the distance dependency property to build the 2-hop labeling index in D rounds where $D \ll n$ denotes the diameter of the graph. Assuming that $L_d(v)=\{(w, dist(v,w)) \in L(v) | dist(v,w) = d\}$ and $L_{<d}(v) = \{(w, dist(v,w)) \in L(v) | dist(v,w) < d\}$, this property proves that $L_d(v)$ only depends on $\bigcup_{u \in N(v,G)} L_{<d}(u)$, i.e, for each label $(w,d) \in L_d(v)$, we have $(w, d-1) \in L_{d-1}(u)$ with $u \in N(v,G)$. Similarly, the label $(w, dist(v,w))$ can be inserted into $L(v)$ when satisfying Lemmas 1 and 2. Following this principle, PSL can process all labels with the same distance originating from all vertices in each round, which helps to ensure high parallel efficiency.

Example 4. *Take v_5 in Figure 1(a) as an example. As shown in Table 1, the label $(v_1, 1)$ is inserted into $L(v_5)$ since the shortest path $\langle v_1, v_5\rangle$ is a valley path and there is no shortest peak path. Similarly, due to the existence of $\langle v_5, v_1, v_2\rangle$, we have $dist(v_5, v_2) = dist(v_5, v_1) + dist(v_1, v_2)$. Thus, the label $(v_2, 2)$ cannot be inserted in $L(v_5)$.*

3.2 2-hop labeling index maintenance methods

In this section, we mainly introduce two state-of-the-art solutions for maintaining the 2-hop labeling index.

- *FULPLL* [12]. In the edge deletion scenario, FULPLL detects all labels possibly affected by the deleted edges and recomputes the missing hubs to maintain the 2-hop cover constraint. For edge insertions, it uses the strategy in [4] to maintain the 2-hop labeling index. However, FULPLL suffers from performance bottlenecks due to its requirement to sequentially process each updated label. This sequential nature significantly limits its scalability for large-scale or highly dynamic graphs. In addition, FULPLL overlooks redundant labels and deletes the correct labels in the scenarios of edge insertion and deletion, respectively.
- *BPCL* [47]. Compared to FULPLL, BPCL adopts the parallel optimization strategy to update the 2-hop labeling index and designs an auxiliary structure to quickly capture the necessary reconstructed labels of each vertex in the scenario of edge deletion. However, this approach introduces two significant challenges: (1) underutilization of resources: computational tasks assigned to individual edges fail to fully leverage available computational resources, leading to reduced efficiency; (2) redundant calculations: repeated computations across different dynamic edges result in longer processing times. Furthermore, the serious memory cost of BPCL includes:
  - Extra auxiliary information. To locate the missed affected labels, for any two nodes $v, w \in V$, BPCL constructs an auxiliary structure PPR to record the elements $((w,c), v)$ and $((v,c), w)$, where (1) $Q(w, v, L_{<d}) = d(w,c) + d(c,v)$ and (2) c is the hub node of v and w [44, 47].
  - Complicated data structure. Upon examining the source code, we observed that BPCL uses complex data structures, such as a combination of "map" and "array," to store label entries and auxiliary information. While this design reduces the time needed to locate affected labels, it significantly increases memory usage compared to simpler, single-array-based structures used by PSL and M2HL.
  - Experimental verification. Our experimental results confirm the memory overhead of BPCL. For example, on the Pokec dataset, BPCL's memory usage exceeds 1.5 TB, while our proposed M2HL requires only 46 GB.

Based on the above analysis, we conclude that there is a strong need for more effective methods that not only guarantee correctness and minimality, but also improve efficiency significantly.

4 EDGE INSERTION

In this section, we introduce how to efficiently update the 2-hop labeling index in the edge insertion scenario. Our high-level idea is to first add the new labels to maintain the correctness, and then delete the redundant labels that are dominated by the new labels, to guarantee the minimality. The details are shown in Algorithm 1.

Algorithm 1: M2HL in the edge insertion scenario

```
Input: $G$, $L = \bigcup_{v \in V} L(v)$, $E^+$
Output: the 2-hop labeling index $L^+$ of $G^+$
1 $L^\#, V_A \leftarrow NLC(G, L, E^+)$   // Collect all new labels to maintain
      the correctness property by Algorithm 2
2 $L^+ \leftarrow RLD(L^\#, V_A)$   // Remove all redundant labels to maintain
      the minimality by Algorithm 3
3 return $L^+$  // The minimal 2-hop labeling index of the graph $G^+$
```

4.1 New label collection

In the edge insertion scenario, the actual distances between some pairs of vertices may decrease, compromising query correctness. To address this issue, it is necessary to collect the new distance messages into the labels. As shown in Algorithm 2, given a minimal 2-hop labeling index L of G and E^+, L^* is first initialized to collect the new labels (Line 1), where $C^*(v) = \{u|(u, dist(u, v)) \in L^*(v)\}$. Then, we insert $(u, 1)$ into $L^*(v)$ with $\forall e(u, v) \in E^+$ if $r(u) > r(v)$ (Lines 2-4). In the d-th step, each node v needs to build the candidate node set $Cand(v)$ (Line 7) which includes

(1) $C^*_{d-1}(u)$ with $u \in N(v, G) \cup N^+_\Delta(v)$. In this case, $\forall(w, d-1) \in L^*_{d-1}(u)$ represents the shortest path that passes through one inserted edge at least.
(2) $C_{d-1}(u)$ with $u \in N^+_\Delta(v)$. In this case, the new distance information consists of the exiting labels and the inserted edges.

Then, each candidate label (w, d) is inserted into $L^*(v)$ if it cannot be dominated by the existing labels (Lines 8-12), and the whole process is terminated until each vertex no longer receives any new label. Finally, each affected vertex v is inserted into V_A if $L^*(v) \neq \emptyset$ (Line 14) and the new 2-hop labeling index is $L^\# = \bigcup_{v \in V} L(v) \cup L^*(v)$ (Line 15).

Example 5. *Given the 2-hop labeling index of G_S and E^+, the index update process is shown in Table 4. Specifically, the new label $(u, 1)$ is inserted into $L^*(v)$ with $\forall e(u, v) \in E^+$ with $r(u) > r(v)$. For example, $(v_0, 1)$ and $(v_2, 1)$ are inserted into $L^*(v_1)$ and $L^*(v_3)$, respectively. Then, each vertex collects the candidate hub vertices which satisfies the proposed two principles in Algorithm 2. Take the vertex v_6 as an example, where $N(v_6, G_S) = \{v_5, v_7\}$ and $N^+_\Delta(v_6) = \{v_0\}$. When $d = 2$, we have $Cand(v_6) = \{v_0, v_4\}$ originating from $C^*_1(u)$ with $u \in N(v_6, G_S) \cup N^+_\Delta(v_6)$. However, these two labels $(v_0, 2)$ and $(v_4, 2)$ cannot be inserted into $L^*(v_6)$ since they are dominated by the existing labels. The whole process is terminated when $d = 2$ since each vertex no longer collects new labels.*

Time complexity. Let $\delta^* = \max_{v \in V} |L^*(v)|$ be the maximal number of new labels over all vertices. In the worst case, each vertex v executes $Q(\cdot)$ to check the candidate new labels from its neighbors. Therefore, the total time complexity is $O(\sum_{v \in V}(|N(v, G)| + |N^+_\Delta(v)|) \cdot \delta \cdot \delta^*)$, which can be reduced to $O(m \cdot \delta \cdot \delta^*)$.

Algorithm 2: New Label Collection (NLC)

```
   Input: $G$, $\bigcup_{v \in V} L(v)$, $E^+$
   Output: $L^\#$, $V_A$
1  Initialize $L^*$, $V_A$, and $d \leftarrow 2$
2  foreach $e(u, v) \in E^+$ do
3      if $r(u) > r(v)$ then Insert $(u, 1)$ into $L^*(v)$ ;
4      else Insert $(v, 1)$ into $L^*(u)$ ;
5  while $L^*_{d-1}$ is not empty do
6      foreach $v \in V$ in parallel do
7          Insert $w$ into $Cand(v)$ when satisfying (1) $w \in C^*_{d-1}(u)$,
             $u \in N(v, G) \cup N^+_\Delta(v)$ or (2) $w \in C_{d-1}(u), u \in N^+_\Delta(v)$
8      foreach $v \in V$ in parallel do
9          foreach $w \in Cand(v)$ with $r(w) > r(v)$ do
10             $L_w \leftarrow L_{<(d+1)}(w) \cup L^*_{<d}(w)$ and $L_v \leftarrow L_{<(d+1)}(v) \cup L^*_{<d}(v)$
11             if $Q(w, v, L_w \cup L_v) > d$ then
12                 Insert $(w, d)$ into $L^*_d(v)$ if $(w, d) \notin L^*_d(v)$
13     $d \leftarrow d + 1$
14 foreach $v \in V$ with $L^*(v) \neq \emptyset$ do $V_A.add(v)$;
15 return $L^\# \leftarrow \bigcup_{v \in V} L(v) \cup L^*(v)$, $V_A$
```

Table 4: The illustration of M2HL in edge insertion scenario

	The 2-hop labeling index of G_S	NLC	$F(\cdot)$	RLD			
				Lemmas			
		$d = 1$		4	5	6	7
v_0	$(v_0, 0)$	–	-1	–	–	–	–
v_1	$(v_1, 0)$, $(v_0, 2)$	v_0	$r(v_0)$	–	–	~~$(v_0, 2)$~~	–
v_2	$(v_2, 0)$, $(v_0, 1)$, $(v_1, 1)$	–	-1	$(v_0, 1)$	$(v_1, 1)$	–	–
v_3	$(v_3, 0)$, $(v_0, 1)$	v_2	$r(v_2)$	–	–	–	–
v_4	$(v_4, 0)$, $(v_0, 1)$, $(v_3, 1)$	–	-1	$(v_0, 1)$	$(v_3, 1)$	–	–
v_5	$(v_5, 0)$, $(v_1, 1)$, $(v_0, 3)$	v_0	$r(v_0)$	–	–	~~$(v_0, 3)$~~	$(v_1, 1)$
v_6	$(v_6, 0)$, $(v_5, 1)$, $(v_0, 2)$, $(v_1, 2)$	v_0	$r(v_0)$	–	–	~~$(v_0, 2)$~~	$(v_5, 1)$, ~~$(v_1, 2)$~~
v_7	$(v_7, 0)$, $(v_0, 1)$, $(v_6, 1)$, $(v_5, 2)$	v_4	$r(v_4)$	–	–	–	$(v_6, 1)$, ~~$(v_5, 2)$~~

4.2 Redundant label deletion

As aforementioned, the correctness property has been maintained by inserting the corresponding labels. Thus, the remaining issue is how to efficiently rule out all redundant labels that are dominated by the inserted labels, so that the minimality is guaranteed. Obviously, it is time-consuming to sequentially check all existing labels, especially on the large-scale graphs, since its time cost can reach $O(n \cdot \delta^2)$.

To tackle this issue, we design some effective pruning strategies to accurately check the candidate redundant labels. For simplicity, let $F(v)$ denote the largest ranking value in $L^*(v)$, i.e, $F(v) = \max_{w \in C^*(v)} r(w)$ with $v \in V_A$, where $V_A = \{v|L^*(v) \neq \emptyset\}$ is the set of vertices that own new labels. Otherwise, $F(v)$ is set as -1.

Lemma 3. *$\forall(w, d) \in L(v)$ and $\forall(w^*, d^*) \in L^*(v)$, we have $d^* < d$ if $w = w^*$.*

Proof. Based on Line 10 of Algorithm 2, (w^*, d^*) is not dominated by any existing label, which proves that $d^* < d$ if $w = w^*$. □

Lemma 4. *$\forall(w, d) \in L(v)$ is not redundant if $v, w \notin V_A$, where L is a minimal 2-hop labeling index.*

Proof. Following Lemmas 1 and 2, $(w, d) \in L(v)$ means that all shortest paths between w and v are valley paths. Due to $L^*(v) = \emptyset$ and $L^*(w) = \emptyset$, there are no new shortest paths $p^*(v, w)$ that pass through the inserted edges. Therefore, we have $d = dist(v, w) \neq Q(v, w, L(v) \cup L(w))$ if removing (w, d) from $L(v)$, which proves that the label (w, d) is not redundant. □

Lemma 5. *$\forall(w, d) \in L(v)$ is redundant if $v \notin V_A$, $w \in V_A$, and $Q(w, v, L^*_{<d}(w) \cup L_{<d}(v)) \leq d$.*

Proof. Due to $Q(w, v, L^*_{<d}(w) \cup L_{<d}(v)) \leq d$, there is at least one shortest path $p^*(v, w)$ with $|p^*(v, w)| \leq d$. Apparently, (w, d) is redundant when $|p^*(v, w)| < d$. Assuming that $p^*(v, w)$ is a unique new shortest path with $|p^*(v, w)| = d$, there are two possible cases:

(1) $p^*(v, w)$ is a valley path. Based on Lemma 1, the label $(w, |p^*|)$ is inserted into $L^*(v)$ to satisfy the 2-hop cover constraint. However, due to $|p^*| = d$, this conclusion is contradictory to Lemma 3. Therefore, p^* cannot be a valley path if $|p^*| = d$.
(2) $p^*(v, w)$ is a peak path. In this case, there exists a middle vertex u located in p^*, where $(u, dist(u, w)) \in L^*(w)$ and $(u, dist(v, u)) \in L(v)$. Based on Lemma 2, the 2-hop cover constraint is satisfied when removing (w, d) from $L(v)$, i.e., $dist(v, w) = L(v)[u] + L^*(w)[u]$.

Based on the above analysis, the lemma holds. □

Lemma 6. *$\forall(w, d) \in L(v)$ with $v \in V_A$ and $w \notin V_A$ is redundant if (1) $r(w) \leq F(v)$ and (2) $Q(w, v, L_{<d}(w) \cup L^*_{<d}(v)) \leq d$.*

Proof. Due to $Q(w, v, L_{<d}(w) \cup L^*_{<d}(v)) \leq d$, $p^*(v, w)$ is a new path where $|p^*(v, w)| \leq d$. Based on $v \in V_A$ and $w \notin V_A$, a detailed analysis of two cases is provided below.

- $r(w) = F(v)$. Based on the definition of $F(\cdot)$, the new label $(w, |p^*(v, w)|)$ has been inserted into $L^*(v)$ since the ranking value of each node is unique. Therefore, we conclude that (w, d) is redundant based on Lemma 3.
- $r(w) < F(v)$. There are two possible cases below.
  (1) $(w, |p^*|) \in L^*(v)$. Similar to the case of $r(w) = F(v)$.
  (2) $(w, |p^*|) \notin L^*(v)$. Referring to the case in Lemma 5, $p^*(v, w)$ is a shortest peak path with $|p^*| \leq d$. Then, (w, d) is a redundant label based on Lemma 2.

□

Lemma 7. *$\forall (w, d) \in L(v)$ with $v, w \in V_A$ is redundant if $Q(v, w, L_v \cup L_w) \leq d$, where $L_v = L_{<d}(v) \cup L^*_{<d}(v)$ and $L_w = L_{<d}(w) \cup L^*_{<d}(w)$.*

Proof. Let $p^*(v, w)$ be a new shortest path with $|p^*| \leq d$. Due to $(w, d) \in L(v)$, we have $Q(v, w, L_{<d}(v) \cup L_{<d}(w)) > d$ based on Lemma 2. Considering that (w, d) is redundant if $|p^*| < d$, we analyze the following three cases about $|p^*| = d$ in detail.

- $Q(v, w, L^*_{<d}(w) \cup L_{<d}(v)) = d$. Referring to Lemma 5, there exists a middle vertex $u \in p^* \setminus \{v, w\}$ to satisfy $|p^*| = L(v)[u] + L^*(w)[u]$.
- $Q(v, w, L_{<d}(w) \cup L^*_{<d}(v)) = d$. Referring to Lemma 6, there exists a middle vertex $u \in p^* \setminus \{v, w\}$ to satisfy $|p^*| = L^*(v)[u] + L(w)[u]$.
- $Q(v, w, L^*_{<d}(w) \cup L^*_{<d}(v)) = d$. Similarly, there exists a middle vertex $u \in p^* \setminus \{v, w\}$ to satisfy $|p^*| = L^*(v)[u] + L^*(w)[u]$.

Thus, we conclude that $p^*(v, w)$ is a shortest peak path if $|p^*(v, w)| = d$, proving that (w, d) is redundant based on Lemma 2. □

Algorithm 3: Redundant Label Deletion (RLD)

```
   Input: $\bigcup_{v \in V} L(v) \cup L^*(v), V_A$
   Output: the 2-hop labeling index $L^+$ of $G^+$
1  Initialize a list $F$
2  foreach $v \in V_A$ do $F(v) \leftarrow \max_{w \in C^*(v)} r(w)$ ;
3  for $\forall (w, d) \in L(v)$ with $w \neq v$ and $v \in V$ in parallel do
4      if $v, w \notin V_A$ then continue ;                                   // Lemma 4
5      if $v \notin V_A$ and $w \in V_A$ then                                 // Lemma 5
6          $L_w \leftarrow L^*_{<d}(w)$ and $L_v \leftarrow L_{<d}(v)$
7      if $v \in V_A$ then
8          if $w \notin V_A$ and $r(w) \leq F(v)$ then                        // Lemma 6
9              $L_w \leftarrow L_{<d}(w)$ and $L_v \leftarrow L^*_{<d}(v)$
10         else if $w \in V_A$ then                                           // Lemma 7
11             $L_w \leftarrow L_{<d}(w) \cup L^*_{<d}(w)$ and $L_v \leftarrow L_{<d}(v) \cup L^*_{<d}(v)$
12     if $Q(w, v, L_w \cup L_v) \leq d$ then Delete $(w, d)$ from $L(v)$ ;
13 return $L^+ \leftarrow \bigcup_{v \in V} L(v) \cup L^*(v)$
```

The details of removing redundant labels are shown in Algorithm 3. First, we calculate the $F(v)$ value with each vertex $v \in V$. Then, we adopt the four effective strategies mentioned in Lemmas 4 to 7 to rule out all redundant labels (Lines 5-16), thus maintaining the minimality of the updated 2-hop labeling index.

Example 6. *The process of redundant label deletion is shown in the "RLD" part of Table 4, where all redundant labels are emphasized with delete lines. First, we compute the $F(\cdot)$ values of all vertices, where $F(v_0)$, $F(v_2)$, and $F(v_4)$ are equal to -1. Then, each vertex sequentially checks the redundant labels based on Lemmas 4 to 7 as follows.*

*(1) **Lemma 4.** Take the label $(v_0, 1) \in L(v_2)$ as an example, where $v_0, v_2 \notin V_A$. When deleting $(v_0, 1)$ from $L(v_2)$, we have $L(v_0) \cap L(v_2) = \emptyset$, proving that $(v_0, 1)$ is not redundant.*
*(2) **Lemma 5.** Take the label $(v_1, 1) \in L(v_2)$ as an example, where $v_2 \notin V_A$ and $v_1 \in V_A$. We conclude that $Q(v_1, v_2, L^*_{<1}(v_1) \cup L_{<1}(v_2)) = 2 > L(v_2)[v_1]$, proving that $(v_1, 1)$ cannot be deleted from $L(v_2)$.*
*(3) **Lemma 6.** Take the label $(v_0, 2) \in L(v_1)$ as an example, where $v_1 \in V_A$ and $v_0 \notin V_A$. We have $Q(v_0, v_1, L_{<2}(v_0) \cup L^*_{<2}(v_1)) = 1 < L(v_1)[v_0]$, proving that $(v_0, 2)$ is a redundant label in $L(v_1)$.*
*(4) **Lemma 7.** Take the label $(v_1, 2) \in L(v_6)$ as an example, where $v_1, v_6 \in V_A$. Considering that $L_{v_1} \leftarrow L_{<2}(v_1) \cup L^*_{<2}(v_1)$ and $L_{v_6} \leftarrow L_{<2}(v_6) \cup L^*_{<2}(v_6)$, we have $Q(v_1, v_6, L_{v_1} \cup L_{v_6}) = L(v_6)[v_1]$, proving that $(v_1, 2)$ is redundant. This is because there exists a shortest peak path $\langle v_1, v_0, v_6 \rangle$ between v_1 and v_6 in Figure 1(b).*

Theorem 1 (Correctness and minimality). *In the edge insertion scenario, the 2-hop labeling index L^+ of G^+ constructed by Algorithm 1 satisfies correctness and minimality.*

Proof. During the index update process, each vertex collects all new distance messages originating from E^+ and gets the corresponding new labels that are not dominated by the existing labels. This process is terminated until each vertex does not get new labels, thereby ensuring the correctness property of L^+.

For the minimality of L^+, we have L^* is minimal based on the distance-dependency property [20]. Considering the redundant labels have been deleted from L based on Lemmas 5 to 7, we prove that each remaining label $(w, d) \in L(v)$ is not redundant as follows.

- $v, w \notin V_A$. Please refer to Lemma 4.
- $r(w) > F(v)$ with $v \in V_A$ and $w \notin V_A$. Based on the definition of $F(\cdot)$, $p^*(v, w)$ is a new valley path that passes through the vertices in $L^*(v)$. Then, we analyze the following three cases.
  (1) $|p^*(v, w)| > d$. In this case, (w, d) cannot be deleted from $L(v)$. Otherwise, the query correctness will be destroyed.
  (2) $|p^*(v, w)| = d$. In this case, all shortest path between v and w are valley paths. Therefore, (w, d) needs to be retained in $L(v)$ based on Lemma 1, thus maintaining the correctness.
  (3) $|p^*(v, w)| < d$. In this case, $p^*(v, w)$ is the shortest valley path. Accordingly, the label $(w, |p^*(v, w)|)$ needs to be inserted into $L^*(v)$ to guarantee the correctness. However, this operation is contradictory to the prerequisite $r(w) > F(v)$. Therefore, we have $|p^*(v, w)| \geq d$ if $r(w) > F(v)$, proving that (w, d) is not redundant.
- $Q(w, v, L_v \cup L_w) > d$ where $L_v = L_{<d}(v) \cup L^*_{<d}(v)$ and $L_w = L_{<d}(w) \cup L^*_{<d}(w)$. In this case, the distance of any new path between v and w is larger than d. Therefore, we have $Q(v, w, L(v) \cup L(w)) \neq d$ when deleting (w, d) from $L(v)$.

Based on the above analysis, each label in L^+ cannot be deleted, thus ensuring the minimality. Hence, the theorem holds. □

Time complexity. In the worst case, Algorithm 3 avoids checking the label $(w, dist(v, w)) \in L(v)$ with $v, w \notin V_A$. Considering that the time cost of each query is $O(\delta)$, the total time cost is $O(n \cdot \delta^2 - n^\# \cdot \delta^\# \cdot \delta)$, where $n^\# = |V| - |V_A|$ and $\delta^\#$ is the maximal number of labels in $V \setminus V_A$.

5 EDGE DELETION

In this section, we introduce how to efficiently update the 2-hop labeling index in the edge deletion scenario. Our key idea is to first rule out all error labels caused by the deleted edges, and then supplement all missing labels to maintain correctness and minimality. We summarize these two main steps in Algorithm 4.

Algorithm 4: M2HL in the edge deletion scenario

```
  Input: $G, L = \bigcup_{v \in V} L(v), E^-$
  Output: the 2-hop labeling index $L^-$ of $G^-$
1 $L, F, V_A \leftarrow ELD(G, L, E^-)$    // Delete all error labels caused by
    $E^-$ based on Algorithm 5
2 $L^- \leftarrow MLI(G^-, L, F, V_A)$ // Insert all missing labels to maintain
    the correctness and minimality based on Algorithm 6
3 return $L^-$              // The minimal 2-hop labeling index of $G^-$
```

5.1 Error label deletion

As mentioned in Section 3.2, FULPLL rules out all labels that are located in the shortest paths associated with the deleted edges. For example, given a vertex $v \in V$ and a deleted edge $e(s,t)$, $\forall(w, dist(v,w)) \in L(v)$ is ruled out if there exists a shortest path $p(v,w)$ passing through $e(s,t)$. However, this strategy removes plenty of non-error labels, i.e., $(w, dist(v,w)) \in L(v)$ is not an error label if there are extra shortest valley paths $p^{\#}(v,w)$ with $|p^{\#}(v,w)| = dist(v,w)$. In addition, FULPLL also suffers from computational inefficiency caused by sequentially handling the deleted edges.

To alleviate the above issue, we design an efficient algorithm to rule out all error labels. Note that our algorithm not only accurately finds all error labels but also handles all deleted edges in parallel. The details are shown as follows.

Lemma 8 (**Error label deletion**). *For any entry $(w, dist(v,w)) \in L(v)$, it is a non-error label if there exists at least one shortest valley path $p^{\#}(v,w)$ in G^- with $|p^{\#}(v,w)| = dist(v,w)$.*

Proof. Considering that the actual distance between any two vertices cannot decrease in the edge deletion scenario, the distances of all peak paths between v and w in G^- are still larger than $dist(v,w)$ if $(w, dist(v,w)) \in L(v)$ based on Lemma 2. Therefore, based on Lemma 1, $(w, dist(v,w))$ is not an error label if there exists another shortest valley path $p^{\#}(v,w)$ in G^- with $|p^{\#}(v,w)| = dist(v,w)$. □

Based on the above analysis, for each label $(w, dist(v,w)) \in L(v)$, it is important to accurately and efficiently determine whether there are extra shortest valley paths $p^{\#}(v,w)$ in G^- with $|p^{\#}(v,w)| = dist(v,w)$. To address this issue, we propose a synchronous label removal strategy to rule out all error labels in parallel.

Lemma 9 (**Synchronous label removal**). *For each label $(w,d) \in L_d(v)$, it is a non-error label when satisfying $(w, d-1) \in L^{ner}_{d-1}(u)$ with $u \in N(v, G^-)$. Here, $L^{ner}_{d-1}(u)$ denotes a non-error label set of u, where the distance of each label is $d-1$.*

Proof. As introduced in Section 3.1, each vertex v collects the labels with the same distance from its neighbors in each round. Similar to this distance-dependency property, for any label $(w,d) \in L_d(v)$, if $\exists u \in N(v, G^-)$ satisfies $(w, d-1) \in L^{ner}_{d-1}(u)$, there exists another shortest valley path $p^{\#}(v,w)$ with $|p^{\#}(v,w)| = d$, thereby proving that (w,d) is not an error label. □

However, directly tracking the non-error labels following Lemma 9 would be time-consuming, as the number of non-error labels is usually much larger than that of error labels. To avoid this issue, we choose to propagate the error labels based on Lemma 9. Algorithm 5 depicts the details of the synchronous label removal process. Given the dynamic graph G^-, the minimal 2-hop labeling index L, and the edge set E^- to be deleted, we first initialize $L^{\#}$ to store all error labels (Line 1). Then, the error labels are gradually ruled out based on Lemma 9 in each round (Lines 2-8). Take the d-th round as an example. We sequentially execute the following two steps.

Algorithm 5: Error Label Deletion (ELD)

```
   Input: $G^-$, $L$, $E^-$
   Output: $L$, $F$, $V_A$
1  Initialize $L^{\#}$, $V_A$, $F$, and $d \leftarrow 1$
2  while $L^{\#}_{d-1} \neq \emptyset$ or $d = 1$ do
3      for each hub node $w \in C_d(v)$ with $v \in V$ in parallel do
4          Insert $w$ into $Cand(v)$ if (1) $w \in C^{\#}_{d-1}(u)$ with $u \in N(v, G^-)$ or
             (2) $w \in C^{\#}_{d-1}(u) \cup C_{d-1}(u)$ with $u \in N^-_{\Delta}(v)$
5      for $\forall w \in Cand(v)$ with $v \in V$ in parallel do
6          if $(w, d-1) \notin L(u)$ with $\forall u \in N(v, G^-)$ then       // Lemma 9
7              Migrate $(w,d)$ from $L_d(v)$ to $L^{\#}_d(v)$
8      $d \leftarrow d + 1$
9  foreach $v \in V$ with $L^{\#}(v) \neq \emptyset$ do
10     $V_A.add(v)$ and $F(v) \leftarrow \max_{w \in C^{\#}(v)} r(w)$
11 return $\bigcup_{v \in V} L(v)$, $F$, $V_A$
```

Table 5: The illustration of M2HL in edge deletion scenario

	The 2-hop labeling index of G_L	ELD	$F(\cdot)$	MLI	
		$d = 1$		$d = 2$	$d = 3$
v_0	$(v_0, 0)$	–	-1	–	–
v_1	$(v_1, 0)$, ~~$(v_0, 1)$~~	v_0	$r(v_0)$	v_0	–
v_2	$(v_2, 0)$, $(v_0, 1)$, $(v_1, 1)$	–	-1	–	–
v_3	$(v_3, 0)$, $(v_0, 1)$, ~~$(v_2, 1)$~~	v_2	$r(v_2)$	–	–
v_4	$(v_4, 0)$, $(v_0, 1)$, $(v_3, 1)$	–	-1	–	–
v_5	$(v_5, 0)$, ~~$(v_0, 1)$~~, $(v_1, 1)$	v_0	$r(v_0)$	–	v_0
v_6	$(v_6, 0)$, ~~$(v_0, 1)$~~, $(v_5, 1)$	v_0	$r(v_0)$	v_0, v_1	–
v_7	$(v_7, 0)$, $(v_0, 1)$, ~~$(v_4, 1)$~~, $(v_6, 1)$	v_4	$r(v_4)$	v_5	–

- Determining candidate labels (Lines 3-4). For each vertex $v \in V$, the candidate error label $(w,d) \in L_d(v)$ is originated from (1) $w \in C^{\#}_{d-1}(u)$ with $u \in N(v, G^-)$ or (2) $w \in C^{\#}_{d-1}(u) \cup C_{d-1}(u)$ with $u \in N^-_{\Delta}(v)$.
- Ruling out error labels (Lines 5-7). Each candidate hub node $w \in Cand(v)$ needs to be migrated from $L_d(v)$ to $L^{\#}_d(v)$ if there is no extra shortest valley path between v and w by Lemma 9.

Finally, all error results are ruled out from the label set, where V_A and F are computed for the subsequent processes (Lines 9-10).

Example 7. *Given the graph G_L and E^- represented by the blue dashed lines in Figure 1(b), the process of error label deletion is shown in Table 5, where all error labels are emphasized with delete lines. The details are shown as follows.*

(1) $d = 1$. We delete the corresponding label whose distance is 1 based on E^-. For example, due to the deletion of $e(v_0, v_5)$ and $e(v_2, v_3)$, the labels $(v_0, 1)$ and $(v_2, 1)$ are deleted from $L(v_5)$ and $L(v_3)$, respectively.

(2) $d = 2$. The error label removal procedure is terminated since all error labels have been ruled out.

Time complexity. In the worst case, each vertex $v \in V$ takes $O(\sum_{u \in N(v,G)} \sum_{d=0}^{D} |L_d(u)|) = O(|N(v,G)| \cdot \delta)$ to check all labels, where $\delta = \max_{v \in V} |L(v)|$. Therefore, the time cost is $O(m \cdot \delta)$.

5.2 Missing label insertion

After ruling out all error labels caused by the deleted edges, each vertex collects the missing labels from its neighbors based on the distance-dependency property, thus maintaining correctness and minimality. For each deleted label $(u, dist(v,u))$, we observe that the correctness of $q(v,w)$ is not affected when $dist(v,w) < dist(v,u) + L[w][u]$. However, the time cost of verifying this is $O(\delta)$, so it costs $O(m \cdot \delta^2)$ time to check all labels. To reduce the time cost, we design two pruning strategies to avoid redundant verification.

Lemma 10. *For any two vertices $v, w \notin V_A$ with $r(w) > r(v)$, $(w, dist(v,w))$ cannot be inserted into $L(v)$.*

Proof. Based on the definition of V_A, there is no change in the labels of v and w when $v, w \notin V_A$, thereby proving that $dist(v,w) = Q(v,w,L(v) \cup L(w))$. Therefore, the label $(w, dist(v,w))$ cannot be inserted into $L(v)$. Otherwise, the minimality will be destroyed. □

Lemma 11. *For each vertex $v \in V_A$, $(w, dist(v,w))$ cannot be inserted into $L(v)$ if $r(w) > F(v)$ and $w \notin V_A$.*

Proof. Let $L^{\#}(v)$ be the error label set of v. Considering that $\forall u \in L^{\#}(v)$ satisfies $r(w) > r(u)$ if $r(w) > F(v)$, we have $r(u) < r(w')$ with $\forall w' \in C(w)$ based on the node order strategy, proving that $L(w) \cap L^{\#}(v) = \emptyset$. This is because the rank value of each hub node is larger than itself. Therefore, the correctness of $q(v,w)$ is not affected by any deleted label in $L^{\#}(v)$. □

Algorithm 6: Missing Labels Insertion (MLI)

```
   Input: $G^-, L, F, V_A$
   Output: the 2-hop labeling index $L^-$ of $G^-$
1  Initialize $L^*$ and $d \leftarrow 2$
2  for $\forall w \in C_1(v)$ with $v \in V$ do Insert $(w, 1)$ into $L^*_1(v)$ if $w \in V_A$ ;
3  while $L^*_{d-1} \neq \emptyset$ do
4      foreach $u \in N(v, G^-)$ with $v \in V$ in parallel do
5          for $\forall w \in C^*_{d-1}(u)$ with $r(w) > r(v)$ do
6              if $v, w \notin V_A$ then continue;                // Lemma 10
7              Insert $w$ into $Cand(v)$
8          for $\forall w \in C_{d-1}(u)$ with $v \in V_A$ and $r(v) < r(w)$ do
9              if $r(w) \leq F(v)$ then Insert $w$ into $Cand(v)$;  // Lemma 11
10     for $\forall w \in Cand(v)$ with $v \in V$ in parallel do
11         if $w \in C_d(v)$ and $w \in V_A$ then Insert $(w, d)$ into $L^*_d(v)$;
12         if $Q(w, v, L_{<d}(w) \cup L_{<d}(v)) > d$ then
13             Insert $(w, d)$ into $L^*_d(v)$ and $L_d(v)$ ▶ the missing labels have
                 been inserted into $L$
14     $d \leftarrow d + 1$
15 return $L^- \leftarrow \bigcup_{v \in V} L(v)$
```

Algorithm 6 shows the details of missing label insertion, where L^* is used to store the inserted labels. Specifically, all 1-hop labels are first collected in L^*_1 to activate the whole procedure (Line 2). Then, for each vertex $v \in V$, the hub node w in the d-th round is originated from (1) $C^*_{d-1}(u)$ or (2) $C_{d-1}(u)$ with $v \in V_A$ (Lines 4-9), where $u \in N(v, G^-)$. Finally, the label (w,d) is inserted into $L(v)$ if it is not dominated by the existing labels (Lines 10-13). Note that the candidate label (w,d) is also inserted into $L^*_d(v)$ when $(w,d) \in L_d(v)$ to avoid the early termination of the procedure, which helps to maintain the correctness and minimality.

Example 8. *The process of missing label insertion is the "MLI" part of Table 5. When $d = 2$, we have $Cand(v_1) = \{v_0\}$ and $Cand(v_7) = \{v_5\}$. Then, the labels $(v_0, 2)$ and $(v_5, 2)$ are inserted into $L(v_1)$ and $L(v_7)$, respectively. Similarly, when $d = 3$, we have $Cand(v_5) = \{v_0\}$. Accordingly, the label $(v_0, 3)$ is inserted into $L(v_5)$.*

Theorem 2 (**Correctness and Minimality**). *In the edge deletion scenario, the 2-hop labeling index L^- of G^- constructed by Algorithm 4 satisfies correctness and minimality.*

Proof. After deleting all error labels $L^{\#}$ by Lemma 9, we have two pruning strategies to reduce the redundant computations during the process of adding missing labels. Specifically, for each deleted error label $(u, dist(v,u))$, each candidate node w is inserted into $Cand(v)$ if it satisfies (1) $r(w) > r(v)$ and (2) $dist(v,w) = dist(v,u) + L(w)[u]$, because the correctness of $q(v,w)$ is possibly affected by the deleted label. Following this property, the missing labels can be regarded as a subset of $\sum_{v \in V} Cand(v)$, which proves that the correctness can be maintained after updating the labels.

Based on the minimal 2-hop labeling index L of G, the remaining label set $L \leftarrow L \setminus L^{\#}$ is also minimal based on Lemma 8. During the subsequent process of missing label insertion, for each vertex $v \in V$, each candidate node $w \in Cand(v)$ cannot be inserted into $L(v)$ if $Q(v,w,L(v) \cup L(w)) > (w, dist(v,w))$ (Lines 12-13), which means that the label $(w, dist(v,w))$ is dominated by the existing labels in $L(v)$. Therefore, L^- is a minimal 2-hop labeling index of G^- when inserting all missing label entries into L. □

Time complexity. In the worst case, each vertex v does not check the label (w,d) with $v, w \notin V_A$. Therefore, the time cost is $O(m \cdot \delta \cdot \delta - n^{\#} \cdot \delta^{\#} \cdot \delta)$, where $n^{\#} = |V| - |V_A|$ and $\delta^{\#}$ is the maximal number of labels in $V \setminus V_A$.

6 EXTENSIONS OF M2HL ON DIRECTED AND WEIGHTED GRAPHS

In this part, we briefly discuss how to extend M2HL for handling directed graphs and weighted graphs as follows.

Directed graphs. Due to the constraint of edge direction, each vertex $v \in V$ collects the in-label and out-label sets, denoted as $L_I(v)$ and $L_O(v)$ respectively, and the minimal 2-hop labeling index of G is $\bigcup_{v \in V} L_I(v) \cup L_O(v)$ [20]. During the index construction process, the in-label (u,d) can be inserted into $L_I(v)$ if $Q(u, v, L_{<d,O}(u) \cup L_{<d,I}(v)) < d$. Accordingly, the out-label (w,d) can be inserted into $L_O(v)$ if $Q(v, w, L_{<d,O}(v) \cup L_{<d,I}(w)) < d$.

- **Edge insertion.** Given an edge set E^+ to be inserted, the new out-labels and in-labels are inserted into $L^*_O(v)$ and $L^*_I(v)$ by executing Algorithm 2 in G. Specifically, in the d-th step, the labels (w,d) and (u,d) are inserted into $L^*_O(v)$ and $L^*_I(v)$ respectively based on Lemmas 1 and 2 when they are not dominated by the existing results. This process is terminated until each vertex does not collect any new label. Furthermore, all redundant labels can be ruled out via Algorithm 3.
- **Edge deletion.** Given an edge set E^- to be deleted, the error in-labels and out-labels are deleted from $L_I(v)$ and $L_O(v)$ by executing Algorithm 5 in G and its reversion, respectively. Similarly, all error labels with the same distance are simultaneously handled in each round. Next, we execute Algorithm 6 to maintain the correctness and minimality.

Weighted graphs. Considering that the minimality is possibly destroyed when directly executing PSL on weighted graphs [20], we adopt the PVC method in [40] to generate the minimal 2-hop labeling index L. The core idea of PVC is that each vertex $v \in V$ only collects the labels (w,d) in the d-th step based on Lemmas 1 and 2. Then, we discuss the index maintenance process in the following two scenarios.

- **Weight decrease.** Similar to the edge insertion scenario, the actual distances of vertex pairs may be decreased. Specifically, all weight-decrease edges can first be regarded as newly inserted edges. Then, referring to the core idea of PVC, we collect all new labels and rule out all redundant labels based on Algorithms 2 and 3, respectively.

- **Weight increase.** Similar to the edge deletion scenario, the actual distances of vertex pairs may be increased. Here, the error labels can still be deleted based on Algorithm 5 since the shortest paths will not go through any weight-increase edge. Afterwards, we execute Algorithm 6 to update the index in the new graph following the core idea of PVC.

7 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of our proposed methods.

7.1 Setup

Datasets. In our experiments, we employ ten real-world datasets as reported in Table 6, which are commonly used in previous works [20, 21]. The largest graph within our dataset contains over 2.9 billion edges. These datasets originate from various types (e.g., social networks and web networks) of small-world networks characterized by a maximal degree value that significantly exceeds the average degree value. Note that all directed graphs have been converted to undirected graphs.

Table 6: Statistic of Real-world Graphs

Alias	Dataset	$\|V\|$	$\|E\|$	d_{avg}	d_{max}
TK	WikiTalk	2.4M	4.7M	4	100,029
SP	SocPokec	1.6M	30.6M	27	14,854
LJ	SocLiveJ	4.8M	42.8M	17	20,333
OK	Orkut	2.9M	106.3M	76	33,313
ID	Indochina	7.4 M	194.1 M	40	256,425
U2	UK2002	18.5M	298.1M	28	194,955
U5	UK2005	39.4M	936.1M	39	1,776,858
IT	IT2004	41.3M	1.15B	49	1,326,744
SK	SK2005	50.6M	1.94B	57	8,563,816
U6	UK2006	77.7M	2.96B	39	4,070,242

Algorithms. We mainly compare the following algorithms:

- **FULPLL [12]**. A fully dynamic 2-hop labeling index maintenance algorithm.
- **BPCL [47]**. A parallel 2-hop labeling index maintenance method, where the bandwidth is set as 0.
- **PSL [20]**. A parallel 2-hop labeling index construction algorithm for static graphs. For each test, we directly use this method to reconstruct the 2-hop labeling index with respect to the same node order strategy.
- **BatchHL$^+$ [17]**. The SOTA maintenance algorithms of the highway distance labeling.
- **M2HL**. Our proposed maintenance algorithms (Algorithm 1 and Algorithm 4).

Setting. We generate five types of tests to evaluate the update time in the scenarios of edge insertions and deletions, where the degrees of vertices to insert/delete edges in these tests are distributed in 10%, 30%, 50%, 70%, and 90%, respectively. Following the setting in [18, 47], the instances of each test are set as 1,000 by default and each instance does not belong to the original graph in the edge insertion scenario. To evaluate the scalability of our method, we vary the number of dynamic edges from 10 to 10^5 while keeping the number of computing cores fixed at 40. Furthermore, we also evaluate the speedup by adjusting the number of computing cores from 10 to 60. The experimental result is marked as "INF" when an algorithm cannot finish in 10^6 seconds or exceeds the memory limitation. For query evaluation, we randomly select 10^6 vertex pairs (s, t) for all datasets and report the average time cost.

Environment. All algorithms are deployed in a Linux server which has Intel(R) Xeon(R) Silver 4210R with 64 computing cores and 1.5 TB of main memory. All algorithms are implemented by C++ and the parallel optimization is supported by OpenMP. Note that BPCL, PSL, and M2HL are equipped with 40 cores.

7.2 Index maintenance evaluation

In this section, we mainly examine the time cost and updated index size of the index maintenance solutions.

• **Index update time.** In this experiment, we evaluate the update time cost of FULPLL, PSL, BPCL, and M2HL on all datasets. The final results in Figures 2 and 3 represent the average update time for all types of tests mentioned above. Tables 7 and 8 show the update time for the five types of tasks in the scenarios of edge insertion and deletion, respectively. Note that the results of PSL and BPCL are omitted in these two tables, because (1) the processing time of PSL across the five types of tasks is very close and (2) BPCL cannot finish within 10^6 seconds or using 1.5TB memory on most datasets.

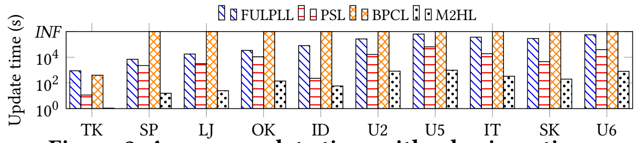

Figure 2: Average update time with edge insertion

Table 7: Update time (s) in the edge insertion scenario

Dataset	FULPLL (10^4 s)					M2HL (s)				
	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Talk	0.85	0.91	0.91	0.82	0.93	1.07	1.23	1.09	1.19	1.12
pokec	0.78	0.74	0.66	0.66	0.61	15.8	12.6	11	9.47	7.25
socLiveJ	1.95	1.85	1.81	1.68	1.61	24.9	19.3	16.2	13.3	17.7
orkut	3.51	3.52	3.47	3.25	3.32	140.2	99.4	81.9	62.9	51.4
Indochina	9.32	8.36	8.13	7.89	6.61	57.8	54.6	49.2	43.3	39.6
uk2002	29.1	25.1	27.2	28.8	23.6	829.1	774.8	655.1	641.5	410.3
uk2005	69.8	67.2	61.1	61.1	60.1	979.5	1335.4	567.1	1205.4	759.8
it2004	30.4	41.7	39.4	35.5	34.8	331.8	548.5	726.8	453.1	478.9
sk2005	30.7	23.9	33.6	28.4	27.7	198.4	187.1	236.5	212.2	172.3
uk2006	62.5	59.5	52.5	54.5	51.1	797.6	951.2	466.4	779.9	467.2

In the edge insertion scenario, as shown in Figure 2 and Table 7, M2HL achieves up to 3, 2, and 2 orders of magnitude speedup compared to FULPLL, PSL, and BPCL, respectively. The superior performance of M2HL can be mainly attributed to its parallelism optimization for handling incremental edges and its effective pruning strategies for reducing the time cost of redundant label removal. In contrast, it is time-consuming for FULPLL to update the 2-hop labeling index, especially on large-scale graphs, because it needs to sequentially handle each inserted edge and cannot be optimized by the parallelism strategy. In addition, it is very time-consuming for PSL to reconstruct the whole 2-hop labeling index, although it is equipped with some optimization techniques. For BPCL, it only employs parallelism optimization to accelerate the index maintenance process for a single dynamic edge. Since the computational workload caused by a single dynamic edge is limited, this strategy inevitably reduces the effectiveness of parallel acceleration.

In the edge deletion scenario, as shown in Fig. 3 and Table 8, M2HL achieves up to 3, 3, and 2 orders of magnitude faster execution times compared to FULPLL, BPCL, and PSL, respectively. Specifically, M2HL not only handles all edges in parallel but also can efficiently and accurately determines the error labels to avoid redundant computations. In contrast, it is required for FULPLL to remove all labels related to any deleted edge, thus causing serious time costs.

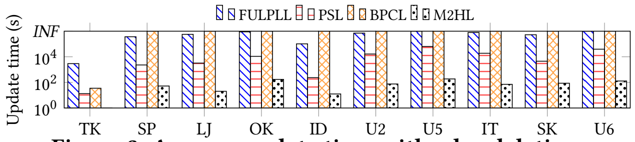

Figure 3: Average update time with edge deletion

Table 8: Update time (s) in the edge deletion scenario

Dataset	FULPLL (10^4 s)					M2HL (s)				
	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Talk	1.11	0.28	0.29	0.28	0.28	0.59	0.34	0.35	0.41	0.40
pokec	43.2	42.5	40.1	38.2	21.2	128.3	62.3	42.2	25.6	2.75
socLiveJ	68.2	62.8	47.6	52.9	53.3	35.2	18.7	22.7	16.5	7.7
orkut	INF	84.2	82.3	INF	81.4	506.1	140.2	97.1	88.1	29.4
Indochina	8.65	40.5	13.1	9.85	9.55	12.1	11.96	13.58	15.8	9.2
uk2002	50.4	70.4	55.8	89.2	88.1	99.2	99.49	64.12	81.89	31.7
uk2005	INF	INF	INF	INF	INF	240.04	330.4	83.5	210.1	72.1
it2004	75.9	INF	75.7	72.1	71.4	17.2	97.5	82.6	108.1	48.1
sk2005	54.3	54.1	52.3	53.2	51.1	87.5	105.3	103.9	88.9	45.2
uk2006	INF	INF	INF	INF	INF	118.7	229.1	26.37	190.8	79.8

• **Memory cost.** In this experiment, we evaluate the memory cost of the three algorithms in the scenarios of edge insertion and deletion, with the number of dynamic edges set to 1,000. Specifically, the experimental result is marked as "OM" when an algorithm exceeds the memory limitation.

Table 9: The comparison of memory cost (GB)

Dataset	Edge insertion				Edge deletion			
	FULPLL	PSL	BPCL	M2HL	FULPLL	PSL	BPCL	M2HL
TK	1.1	1.06	128.2	1.06	1.07	1.07	115.3	1.07
SP	44.7	44.5	OM	44.5	44.7	44.7	OM	44.7
LJ	84.1	83.9	OM	83.9	83.9	83.9	OM	83.9
OK	140.1	139.8	OM	139.8	139.9	139.9	OM	139.9
ID	22.7	22.3	OM	22.3	21.9	21.9	OM	21.9
U2	345.1	343.8	OM	343.8	344.1	344.1	OM	344.1
U5	985.8	985.4	OM	985.4	985.6	985.6	OM	985.6
IT	555.3	554.3	OM	554.3	554.5	554.5	OM	554.5
SK	187.6	186.8	OM	186.8	187.1	187.1	OM	187.1
U6	1030.5	1027.3	OM	1027.3	1030.4	1030.4	OM	1030.4

As shown in Table 9, we observe that the memory cost of FULPLL is slightly larger than those of PSL and M2HL in the edge insertion scenario. This is because FULPLL only guarantees the correctness property, whilst ignoring the redundant labels dominated by the new 2-hop labeling index entries. Note that the memory cost of M2HL is consistent with that of PSL in all scenarios. This is because M2HL not only rules out all redundant labels in the edge insertion scenario but also accurately determines all candidate labels in the edge deletion scenario. Notice that the memory cost of BPCL is almost 2 orders of magnitude higher than that of M2HL. The high memory cost of BPCL is mainly attributed to its complex data structure and the precomputed auxiliary structure, making it fail with out-of-memory errors on most datasets.

7.3 Scalability and query evaluation

In this part, we evaluate the scalability of the algorithms with respect to the update size and multi-core number, respectively. Note that the experimental results of BPCL are not listed since it cannot finish on most datasets.

• **Scalability w.r.t. the number of dynamic edges.** Here, we evaluate the scalability of M2HL, PSL, and FULPLL on update time by varying the number of dynamic edges from 10 to 100,000.

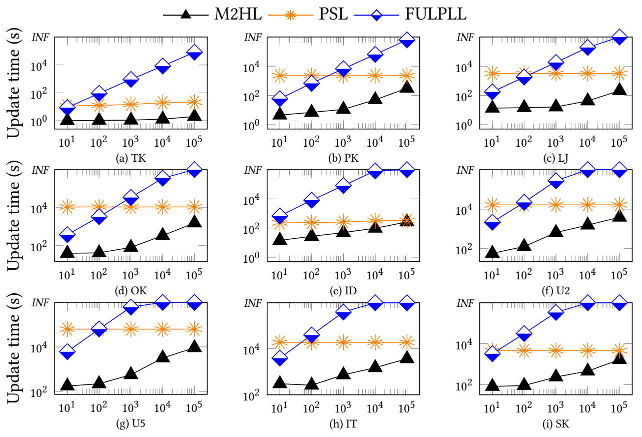

Figure 4: Update time (s) vs. the number of inserted edges

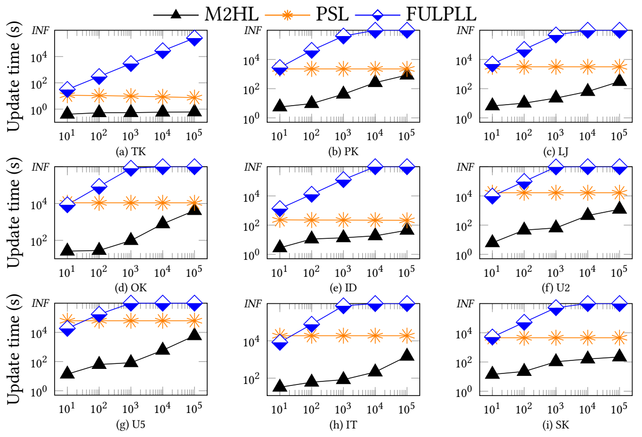

Figure 5: Update time (s) vs. the number of deleted edges

As shown in Figure 4, we observe that the time cost of M2HL and FULPLL increases as the number of inserting edges increases, whilst M2HL achieves up to 4 orders of magnitude speedup compared to FULPLL. Specifically, the update time of FULPLL grows approximately linearly since it needs to sequentially supplement the new labels caused by each inserting edge. Therefore, FULPLL cannot finish the task within a reasonable time limit when inserting too many edges. In contrast, M2HL can simultaneously handle all inserting edges, which not only avoids the production of redundant labels but also contributes to improving the effect of parallelism optimization. The processing time of PSL remains almost the same across all tests, as PSL requires a complete reconstruction of the 2-hop labeling index. In comparison, M2HL achieves up to a 2-order and 1-order magnitude improvement over PSL when the number of inserted edges is 10 and 100,000, respectively.

As shown in Figure 5, we observe that M2HL is up to 4 orders of magnitude faster than FULPLL in the edge deletion scenario. Similar to the edge insertion scenario, the superior performance of M2HL is attributed to the parallelism optimization of handling decremental edges and effective pruning strategies to avoid redundant computations.

• **Scalability w.r.t. the number of multi cores.** In this experiment, we assess the maintenance speedup of M2HL by varying the number of cores from 10 to 60, where the number of dynamic edges is set as 100,000. Note that in this setting, FULPLL is omitted since it cannot be parallelized.

As shown in Figure 6, compared to the setting with 10 cores, M2HL achieves up to 5.4× (on average 4.4×) acceleration in terms of update time when utilizing a total number of 60 cores. This is mainly because (1) M2HL designs effective strategies to handle all inserting edges in each round and (2) the computational workload of index maintenance is higher when inserting more edges. Thus, M2HL can efficiently maintain the 2-hop labeling index with a large number of dynamic edges.

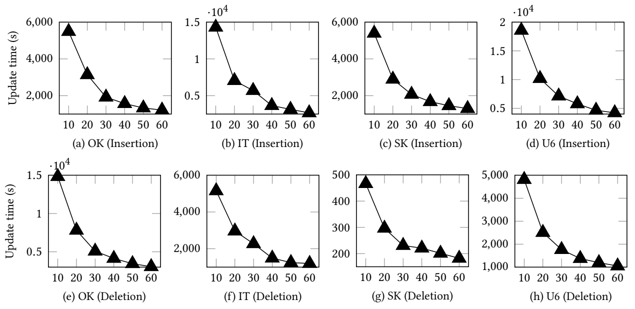

Figure 6: The update time vs. the number of cores

• **Query time** In this part, we evaluate the average query time of FULPLL, PSL, BPCL, and M2HL. As shown in Figure 7, the query time of FULPLL and BPCL is slightly higher than that of M2HL. This is because FULPLL does not rule out the redundant labels dominated by the other elements. Note that the query time of M2HL is the same as that of PSL, since M2HL can maintain correctness and minimality in all cases.

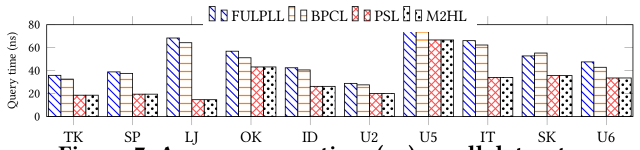

Figure 7: Average query time (ns) on all datasets

7.4 Comparison with BatchHL$^+$

In this part, we compare the index update time and query performance of M2HL and BatchHL$^+$ with our method. Specifically, the numbers of landmark vertices in BatchHL$^+$ are set to 50, 100, and 1000, respectively. For lack of space, we only present the results for four datasets, as the omitted datasets exhibit similar trends.

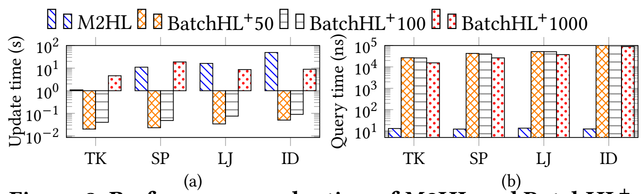

Figure 8: Performance evaluation of M2HL and BatchHL$^+$

As shown in Figure 8 (a), when the number of landmarks is set to 50, BatchHL$^+$ achieves up to 2 orders of magnitude speedup on the update time compared to M2HL. This is due to BatchHL$^+$'s more relaxed index structure, which results in lower time cost. However, the performance gap narrows significantly as the number of landmarks increases, as more vertices are affected, thereby increasing processing time. For instance, when the number of landmarks exceeds 500, their update time costs are comparable.

As shown in Figure 8 (b), M2HL achieves up to 4 orders of magnitude speedup on query time compared to BatchHL$^+$. This exceptional query performance is due to M2HL's compact index structure, where the index-hop number between any two connected vertices does not exceed 2. Additionally, the distance between any two disconnected vertices can be quickly determined. In contrast, BatchHL$^+$ employs a highway structure to quickly estimate the upper bound of the distance between any two vertices. Following this, a DFS-based search strategy is needed to compute the exact shortest distances. In the worst case, the query time complexity of this method is $O(E)$.

7.5 Performance evaluation on directed and weighted graphs

Table 10 records the average time cost of M2HL and PSL in scenarios involving edge insertions and deletions for two types of graphs, where the number of dynamic edges is 1000. Specifically, M2HL demonstrates a speedup of up to two orders of magnitude compared to PSL across all scenarios. Furthermore, we notice that the time cost associated with PSL on directed graphs is significantly higher than that on weighted graphs. This disparity arises because the inclusion of edge directions increases the diameter of the graphs, consequently diminishing the effectiveness of the node ranking strategy.

Table 10: Average time (s) on directed and weighted graphs

	Datasets	Insertion	Deletion	PSL
Directed graph	SP	15.9	82.7	4236.5
	LJ	36.7	39.9	9749.5
Weighted graph	SP	8.7	13.8	1448.6
	LJ	14.8	20.2	1896.6

Figure 9 illustrates the average update time for both directed and weighted graphs as the number of dynamic edges varies. Notably, the time cost of our method scales proportionally with the increase in dynamic edges. In comparison to PSL, when the number of dynamic edges reaches 10^5, M2HL exhibits remarkable performance improvements. Specifically, for directed graphs, M2HL achieves speedups of up to 11.5× and 23.9× in scenarios involving edge insertion and deletion, respectively. Similarly, for weighted graphs, M2HL demonstrates speedups of up to 23.2× and 26.2× in the same scenarios. Figures 10 and 11 showcase the update time of M2HL on two types of graphs as the number of cores varies. When compared to the setting with 10 cores, M2HL achieves an acceleration of up to 4.8× (with an average of 4.6×) on directed graphs when utilizing a total of 60 cores. Similarly, for weighted graphs, M2HL demonstrates an acceleration of up to 5.3× (with an average of 4.8×) when using 60 cores.

8 RELATED WORK

In this section, we first discuss the related works about the 2-hop labeling on the static and dynamic graphs, respectively. Then, we introduce the applications of 2-hop indexes to various path query problems. It is noted that road networks typically exhibit a longer diameter and lower degrees [26]. Due to the structural difference

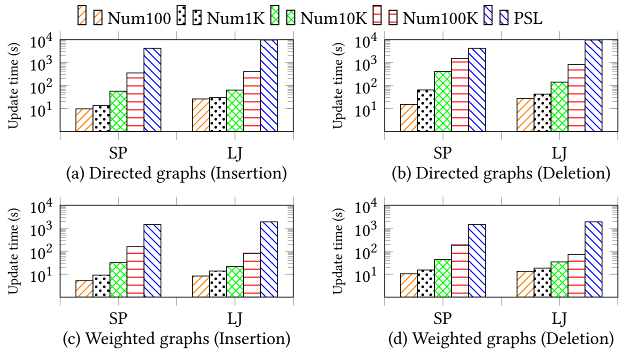

Figure 9: Average time (s) vs. the number of dynamic edges

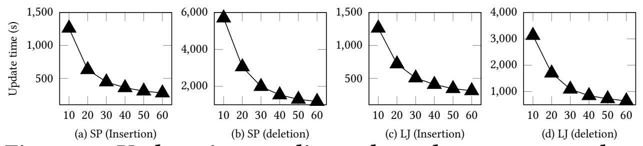

Figure 10: Update time on directed graphs vs. core number

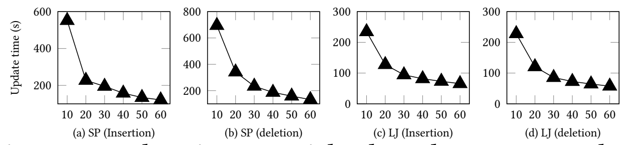

Figure 11: Update time on weighted graphs vs. core number

between small-world graphs and road networks, the methods constructing distance indexes differ significantly between these two types of graphs, leading to two distinct research directions.

8.1 2-hop labeling on static graphs

Cohen et al. [11] first proposed the 2-hop labeling to efficiently solve the distance queries. On *the small-world graphs*, Akiba et al. [3] proposed the pruned landmark labeling (PLL) which adopts the node order strategy to build a minimal 2-hop labeling. Later on, Li et al. [20] proposed parallel shortest-distance labeling (PSL) to build the 2-hop labeling in parallel, whilst guaranteeing the minimality. We will introduce their detailed steps in Section 3.1. Compared to PSL, the core-tree labeling [21] reduces the index size by exploring the core-periphery property with a negligible cost in query time.

On *the road networks*, Akiba et al. [2] proposed the pruned highway labeling to decompose the road network into disjoint paths. H2H [26] combines the 2-hop labeling and the tree decomposition strategy to provide excellent query performance. Due to the large index size of H2H, the authors of [16] proposed HC2L which adopts a balanced tree hierarchy to reduce the index size and the search space of 2-hop labeling. Since we focus on the maintenance of 2-hop labeling index on dynamic graphs, we omit the details which can be found in [17, 22, 36].

8.2 2-hop labeling on dynamic graphs

On *the small-world graphs*, Akiba et al. [4] proposed IncPLL to update the 2-hop labeling in the edge insertion scenario. However, it does not remove the outdated entries since the authors considered it too costly. D'angelo et al. [12] proposed DecPLL to update the 2-hop labeling in the edge deletion scenario. Zhang et al. [43] proposed WPSL to update the 2-hop labeling on the dynamic weighted graphs. However, this method needs to use the vertex pruning records to quickly capture the connections of vertices and their label entries, thus incurring serious memory costs. In addition, this method cannot correctly process the case when edge weight increases [47]. To address this limitation, the authors in [47] proposed BPCL to modify the maintenance strategy of WPSL in the edge weight increasing scenario. However, we observe that BPCL incurs significant memory overhead to store both the index and a pre-computed auxiliary structure, leading to poor scalability.

On *the road networks*, the authors in [46] proved that the index maintenance of contraction hierarchy [19] and hierarchical 2-hop index [26] can become unbounded even for single weight updates. More details of index maintenance on road network can be found in [46].

8.3 Other 2-hop labeling-based path queries

Reachability query. To construct the 2-hop labeling for the reachability query [11], each node v maintains two sets $L_I(v)$ and $L_O(v)$ to record the in-entries and out-entries, respectively. This setup ensures that the index-based hop number between any two reachable vertices is no greater than 2, significantly enhancing query efficiency. Furthermore, the 2-hop labeling index has been expanded to address reachability queries on edge-labeled graphs [29], temporal graphs [35], and bipartite graphs [8]. A comprehensive summary of these advancements can be found in [42].

Shortest path counting. Zhang et al. [45] first adopted 2-hop labeling to count the number of different shortest paths between two vertices. Specifically, each node v collects the distance and the number of shortest paths to each hub node u. Then, the number of shortest paths between any two vertices can be answered using only their labels. Furthermore, Peng et al. [28] designed the parallel optimization strategy to largely reduce the indexing time. Wang et al. [33] adopted the core-tree index to significantly reduce its space cost while slightly weakening the query efficiency.

9 CONCLUSION

In this paper, we study the problem of 2-hop labeling index maintenance on large dynamic graphs. We first identify the core principles of 2-hop labeling index construction and analyze the limitations of existing index maintenance methods. Based on these analysis, we then propose a novel approach **M2HL**, which not only theoretically guarantees the correctness and minimality of the 2-hop labeling index, but also efficiently handles the updates of both edge insertion and deletion in parallel. Our comprehensive experiments on ten real-world large-scale graphs demonstrate that M2HL achieves significant improvements in terms of index update efficiency and scalability. Additionally, our approach can be easily extended for processing directed graphs and weighted graphs.

ACKNOWLEDGEMENTS

This work was supported in part by NSFC under Grant 62302421, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2023A1515011280, 2025A1515010439, Shenzhen Research Institute of Big Data under grant SIF20240002, SIF20240004, Guangdong Talent Program under Grant 2021QN02X826, Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS20211021111415025, Ant Group through CCF-Ant Research Fund, the Guangdong Provincial Key Laboratory of Big Data Computing, and The Chinese University of Hong Kong, Shenzhen.

,

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. 2012. Hierarchical Hub Labelings for Shortest Paths. In *ESA, 2012 (Lecture Notes in Computer Science, Vol. 7501)*. Springer, 24–35.

[2] Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. 2014. Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling. In *ALENEX, 2014*. SIAM, 147–154.

[3] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*. ACM, 349–360.

[4] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2014. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *WWW, 2014*. ACM, 237–248.

[5] Sihem Amer-Yahia, Michael Benedikt, Laks V. S. Lakshmanan, and Julia Stoyanovich. 2008. Efficient network aware search in collaborative tagging sites. *Proc. VLDB Endow.* 1, 1 (2008), 710–721.

[6] Shikha Anirban, Junhu Wang, and Md. Saiful Islam. 2023. Experimental Evaluation of Indexing Techniques for Shortest Distance Queries on Road Networks. In *ICDE, 2023*. IEEE, 624–636.

[7] Shikha Anirban, Junhu Wang, Md. Saiful Islam, Humayun Kayesh, Jianxin Li, and Mao Lin Huang. 2022. Compression techniques for 2-hop labeling for shortest distance queries. *World Wide Web* 25, 1 (2022), 151–174.

[8] Xiaoshuang Chen, Kai Wang, Xuemin Lin, Wenjie Zhang, Lu Qin, and Ying Zhang. 2021. Efficiently Answering Reachability and Path Queries on Temporal Bipartite Graphs. *Proc. VLDB Endow.* 14, 10 (2021), 1845–1858.

[9] Yankai Chen, Yixiang Fang, Yifei Zhang, Chenhao Ma, Yang Hong, and Irwin King. 2024. Towards Effective Top-N Hamming Search via Bipartite Graph Contrastive Hashing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2024).

[10] Yankai Chen, Jie Zhang, Yixiang Fang, Xin Cao, and Irwin King. 2021. Efficient community search over large directed graphs: An augmented index-based approach. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence (IJCAI)*. 3544–3550.

[11] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[12] Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. 2019. Fully Dynamic 2-Hop Cover Labeling. *ACM J. Exp. Algorithmics* 24, 1 (2019), 1.6:1–1.6:36.

[13] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. 2014. Robust Distance Queries on Massive Networks. In *ESA (Lecture Notes in Computer Science, Vol. 8737)*. Springer, 321–333.

[14] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD, 2021*. ACM, 459–471.

[15] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective Community Search for Large Attributed Graphs. *Proc. VLDB Endow.* 9, 12 (2016), 1233–1244.

[16] Muhammad Farhan, Henning Koehler, Robert Ohms, and Qing Wang. 2023. Hierarchical Cut Labelling - Scaling Up Distance Queries on Road Networks. *Proc. ACM Manag. Data* 1, 4 (2023), 244:1–244:25.

[17] Muhammad Farhan, Henning Koehler, and Qing Wang. 2024. BatchHL$^+$: batch dynamic labelling for distance queries on large-scale networks. *VLDB J.* 33, 1 (2024), 101–129.

[18] Muhammad Farhan, Qing Wang, and Henning Koehler. 2022. BatchHL: Answering Distance Queries on Batch-Dynamic Networks at Scale. In *SIGMOD, 2022*. ACM, 2020–2033.

[19] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *WEA, 2008 (Lecture Notes in Computer Science, Vol. 5038)*. Springer, 319–333.

[20] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling Distance Labeling on Small-World Networks. In *SIGMOD*. ACM, 1060–1077.

[21] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2020. Scaling Up Distance Labeling on Graphs with Core-Periphery Properties. In *SIGMOD*. ACM, 1367–1381.

[22] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An Experimental Study on Hub Labeling based Shortest Path Algorithms. *Proc. VLDB Endow.* 11, 4 (2017), 445–457.

[23] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery. In *SIGMOD*. ACM, 845–859.

[24] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. Efficient Directed Densest Subgraph Discovery. *SIGMOD Rec.* 50, 1 (2021), 33–40.

[25] Seth A. Myers and Jure Leskovec. 2014. The bursty dynamics of the Twitter information network. In *WWW, 2014*. ACM, 913–924.

[26] Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. 2018. When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest Distance Queries on Road Networks. In *SIGMOD*. ACM, 709–724.

[27] You Peng, Zhuo Ma, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Xiaoshuang Chen. 2023. Efficiently Answering Quality Constrained Shortest Distance Queries in Large Graphs. In *ICDE, 2023*. IEEE, 856–868.

[28] You Peng, Jeffrey Xu Yu, and Sibo Wang. 2023. PSPC: Efficient Parallel Shortest Path Counting on Large-Scale Graphs. In *ICDE, 2023*. IEEE, 896–908.

[29] You Peng, Ying Zhang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2020. Answering Billion-Scale Label-Constrained Reachability Queries within Microsecond. *Proc. VLDB Endow.* 13, 6 (2020), 812–825.

[30] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. 2009. Fast shortest path distance estimation in large networks. In *CIKM*. ACM, 867–876.

[31] Yu-Xuan Qiu, Dong Wen, Lu Qin, Wentao Li, Ronghua Li, and Ying Zhang. 2022. Efficient Shortest Path Counting on Large Road Networks. *Proc. VLDB Endow.* 15, 10 (2022), 2098–2110.

[32] Monique V. Vieira, Bruno M. Fonseca, Rodrigo Damazio, Paulo Braz Golgher, Davi de Castro Reis, and Berthier A. Ribeiro-Neto. 2007. Efficient search ranking in social networks. In *CIKM, 2007*. ACM, 563–572.

[33] Yiqi Wang, Long Yuan, Zi Chen, Wenjie Zhang, Xuemin Lin, and Qing Liu. 2023. Towards Efficient Shortest Path Counting on Billion-Scale Graphs. In *ICDE, 2023*. IEEE, 2579–2592.

[34] Victor Junqiu Wei, Raymond Chi-Wing Wong, and Cheng Long. 2020. Architecture-Intact Oracle for Fastest Path and Time Queries on Dynamic Spatial Networks. In *SIGMOD, 2020*. ACM, 1841–1856.

[35] Dong Wen, Yilun Huang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2020. Efficiently Answering Span-Reachability Queries in Large Temporal Graphs. In *ICDE, 2020*. IEEE, 1153–1164.

[36] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation. *Proc. VLDB Endow.* 5, 5 (2012), 406–417.

[37] Yuanyuan Zeng, Yixiang Fang, Wengshen Luo, and Chenhao Ma. 2025. Accelerating Skyline Path Enumeration with a Core Attribute Index on Multi-attribute Graphs. *Proc. ACM Manag. Data* 3, 3 (2025), 24:1–24:26.

[38] Yuanyuan Zeng, Yixiang Fang, Chenhao Ma, Xu Zhou, and Kenli Li. 2024. Efficient Distributed Hop-Constrained Path Enumeration on Large-Scale Graphs. *Proc. ACM Manag. Data* 2, 3 (2024), 22:1–22:25.

[39] Yuanyuan Zeng, Kenli Li, Xu Zhou, Wensheng Luo, and Yunjun Gao. 2022. An Efficient Index-Based Approach to Distributed Set Reachability on Small-World Graphs. *IEEE Trans. Parallel Distributed Syst.* 33, 10 (2022), 2358–2371.

[40] Yuanyuan Zeng, Chenghao Ma, and Yixiang Fang. 2024. Distributed Shortest Distance Labeling on Large-Scale Graphs. *Proc. VLDB Endow.* 17, 10 (2024), 2641–2653.

[41] Yuanyuan Zeng, Wangdong Yang, Xu Zhou, Guoqin Xiao, Yunjun Gao, and Kenli Li. 2022. Distributed Set Label-Constrained Reachability Queries over Billion-Scale Graphs. In *ICDE*. IEEE.

[42] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. 2023. An Overview of Reachability Indexes on Graphs. In *SIGMOD, 2023*. ACM, 61–68.

[43] Mengxuan Zhang, Lei Li, Wen Hua, and Xiaofang Zhou. 2021. Efficient 2-Hop Labeling Maintenance in Dynamic Small-World Networks. In *ICDE*. IEEE, 133–144.

[44] Mengxuan Zhang, Lei Li, Goce Trajcevski, Andreas Züfle, and Xiaofang Zhou. 2023. Parallel Hub Labeling Maintenance With High Efficiency in Dynamic Small-World Networks. *IEEE Trans. Knowl. Data Eng.* 35, 11 (2023), 11751–11768.

[45] Yikai Zhang and Jeffrey Xu Yu. 2020. Hub Labeling for Shortest Path Counting. In *SIGMOD, 2020*. ACM, 1813–1828.

[46] Yikai Zhang and Jeffrey Xu Yu. 2022. Relative Subboundedness of Contraction Hierarchy and Hierarchical 2-Hop Index in Dynamic Road Networks. In *SIGMOD, 2022*. ACM, 1992–2005.

[47] Xinjie Zhou, Mengxuuan Zhang, Lei Li, and Xiaofang Zhou. 2024. Scalable Distance Labeling Maintenance and Construction for Dynamic Small-World Networks. In *ICDE, 2024*. IEEE, 133–144.