Distributed Shortest Distance Labeling on Large-Scale Graphs

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

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

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

ABSTRACT

Distance labeling approaches are widely adopted to speed up the shortest-distance query performance. Due to the explosive growth of data graphs, a single machine can hardly satisfy the requirements of both computational power and memory capacity, which causes an urgent need for efficient distributed methods. As the graph is distributed across different machines, it is inevitable to frequently exchange messages among different machines when deploying the existing centralized distance labeling methods on the distributed environment, thereby producing serious communication costs and weakening the scalability. To alleviate this problem, we design a distributed hop-based index **DH-Index**, which is designed based on a newly proposed boundary graph structure and restricts the index-based hop number of each connected vertex pair within 4 hops. In addition, we propose a hierarchical algorithm to accelerate the index construction and reduce the communication cost. Furthermore, a bidirectional searching strategy is proposed to efficiently resolve the query tasks based on DH-Index. The comprehensive experimental results on eight real-world graphs demonstrate that DH-Index achieves up to 65.5× and 3 orders of magnitude speedup than the existing methods in indexing time and query performance respectively, and exhibits superior capabilities on memory space, communication cost, and scalability.

PVLDB Reference Format:
Yuanyuan Zeng, Chenhao Ma, and Yixiang Fang. Distributed Shortest Distance Labeling on Large-Scale Graphs. PVLDB, 17(10): 2641-2653, 2024.
doi:10.14778/3675034.3675053

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/1fyxytx/VLDB_2024.git.

1 INTRODUCTION

The shortest distance query $q(s, t)$ requires to return the length of the shortest path between two vertices s and t in a given graph. Answering distance queries is one of the fundamental problems in graph analytics and serves as a building block in many graph-based areas, including community searching [9, 16, 20, 21], GPS navigation [22], fraud detection [30], and route planning services [2, 8, 28, 34]. Due to the inefficiency of many online searching methods, such as Dijkstra and the bidirectional BFS strategy [7], the index-based methods, especially the 2-hop index and its variations, have attracted much attention recently [18]. Specifically, the 2-hop index aims to connect any two reachable vertices, where the index-based hop number of them is no more than 2, thereby quickly answering the shortest distance queries. Apparently, the improvement of query performance can reduce the time cost of many graph-based applications, thus improving the user experience. In addition, the 2-hop index has been applied in many graph-based query problems, such as the reachability query [5, 14, 27, 32], the shortest path query [40, 41], and the shortest path counting [25, 26, 29, 43].

Motivation. Numerous excellent centralized techniques are proposed to optimize the building time and index size of the 2-hop index. For example, a pruned landmark labeling (PLL) method [3] is proposed to construct a global minimal 2-hop index based on a vertex ordering strategy. In addition, a parallel shortest distance labeling (PSL) approach [17] is proposed to further accelerate the efficiency of index construction on PLL. However, the huge memory cost of the complete 2-hop index on large-scale graphs is still expensive. To reduce the index size, CTL [18] adopts the tree decomposition (TD) method to generate a core-tree structure where only the core part is equipped with the 2-hop index. This strategy facilitates a critical and effective trade-off between the index size and query time. However, TD is mainly designed for road networks [22] and is not suitable for dense graphs. This is because the size of the core graph is possibly enormous when inserting the new edges originating from the deleted vertices. These algorithms can be treated as variants of the 2-hop index, indicating that 2-hop-based solutions are still the state-of-the-art choice. However, with the explosive growth of data graphs [13, 37–39], building the 2-hop index on large-scale graphs becomes challenging for a single machine due to the limited memory and computation resources [18].

Challenges. Since distributed computing clusters provide sufficient resources in a relatively easy and cheap way [4, 13], it is crucial to design efficient and scalable distributed 2-hop-based approaches that can harness the cluster computing resources to reduce the indexing time while ensuring robust query performance. However, it is not feasible to directly deploy the 2-hop-based techniques in the distributed setting. Specifically, the key to building the 2-hop index is to determine whether the new label entries satisfy the 2-hop cover [17]. Take the vertex pair (v_1, v_5) in Fig. 1 (a) as an example, where $(v_1, 3)$ is a label entry of v_5. The verification of 2-hop cover of $(v_1, 3)$ relies on $L(v_1)$ and $L(v_5)$, where $L(v)$ is the label set of v. Due to the graph being distributed across different machines, it is inevitable for the centralized index construction methods, such as PLL and PSL, to frequently visit the label entries of all vertices placed in different machines, which causes serious communication costs and weakens the scalability. Moreover, it is

*Chenhao Ma is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 10 ISSN 2150-8097.
doi:10.14778/3675034.3675053

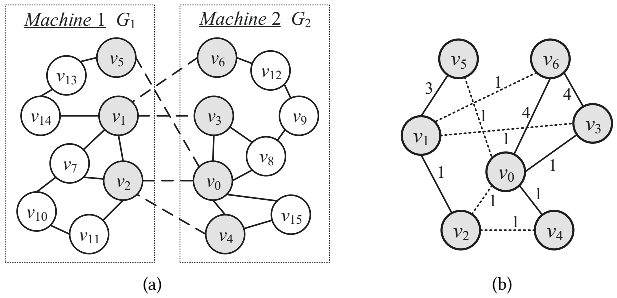


Figure 1: (a) Graph G and (b) Boundary graph G_B of G

also time-consuming to generate the core graph and build CTL in a distributed manner since these two processes cannot fully utilize the computing resources and involve frequent message exchanges (similar to PLL/PSL), respectively. The detailed analysis of their drawback in the distributed setting will be presented in Section 2.

To address the above challenges, our goal is to develop an efficient and scalable index-based algorithm to address the shortest distance query problem in the distributed setting, which needs to ensure good index construction scalability and enhance query performance for large-scale graphs. To achieve our objectives, we need to focus on three challenges: (1) how to quickly construct the proposed index; (2) how to provide good query performance; and (3) how to achieve good scalability.

Our approaches. Based on the aforementioned analysis, we propose an efficient distributed hierarchical hop-based index, called DH-Index, that simultaneously achieves the above goals. Specifically, DH-Index is equipped with the interior index DH_I and the boundary index DH_B. DH_I can be built on all subgraphs in parallel and DH_B is designed based on a boundary graph structure that collects the shortest distances from different machines, thus addressing the challenge (1). For example, Fig. 1(b) is the boundary graph of Fig. 1(a), which collects the shortest distances from the two machines. Compared to the core tree structure in [18], the boundary graph can be quickly constructed leveraging edge-cut graph partitioning techniques [24, 31] and exploiting local properties of shortest distances. Theoretically, we prove that, through DH-Index, the shortest distance between any connected vertex pair can be answered within a maximum of four hops. Building upon this, we devise an efficient bidirectional search strategy to quickly resolve query tasks, effectively tackling the challenge (2).

Furthermore, we design two strategies to enhance the scalability of DH-Index, thus addressing the challenge (3). First, the size of the boundary graph is minimized by ruling out the redundant edges that are dominated by other edges. This strategy can largely reduce the computations during the construction of DH_B. Second, we design an effective algorithm that can fully utilize the computing resources in the cluster to construct DH_B. Specifically, each vertex in the boundary graph is activated in specific machines to execute the process of index construction whilst guaranteeing the accuracy of query results.

Contributions. In this paper, we make the following principal contributions:

Table 1: Notations and meanings.

Notations	Meanings
G=(V, E)	an undirected graph
$N(v, G)$	the neighbor set of v in G
$deg(v)$	the degree of v
W_{uv}	the weight value of $e(u, v)$
$p(s, t)$	a path from s to t in G
$dist(s, t)$	the shortest distance between s and t in G
$m(s)$	the machine where the vertex s is placed
$\{G_i\}_{i=1}^{k}$	a total number of k subgraphs
E_{cut}	a set of cutting edges
$L(v)$	a labeling function of v to record label entries
V_B and V_I	sets of boundary vertices and interior vertices
V_C	a set of core vertices
$p^{bd}(s, t)$	a boundary path between s to t
$p^{in}(s, t)$	an interior path between s to t
$dhop(p)$	the index-based hop number of p

- We propose a boundary graph-based distributed index, **DH-Index**, which bounds the index-based hop number of each connected vertex pair within 4 hops.
- We design an efficient index construction method to build DH-Index in a distributed manner and propose a bidirectional searching strategy that leverages DH-Index to quickly resolve the query tasks.
- We design two optimization strategies to reduce the boundary graph size and improve the utilization of distributed computing resources, respectively.
- Extensive experiments on seven datasets with various workloads demonstrate that DH-Index achieves up to 65.5× and 3 orders of magnitude speedup than distributed PSL and BiBFS in indexing time and query performance, respectively.

Roadmap. The rest of the paper is organized as follows. Section 2 defines the shortest distance problem and discusses state-of-the-art methods. Section 3 elaborates on the DH-Index structure and the corresponding query algorithm, while Section 4 outlines our method for index construction. Section 5 assesses our approach through extensive experiments. Related work is reviewed in Section 6, and Section 7 concludes the paper.

2 PRELIMINARY

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

2.1 Problem Definition

Let G=(V, E) be an undirected graph where V and E 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 $deg(v)$=$|N(v, G)|$ denotes the degree of v. Given a vertex pair (s, t), we represent a path between s and t as $p(s, t)$=$\langle v_0 = s, v_1, \ldots, v_{k-1}, v_k = t\rangle$, where $e(v_i, v_{i+1}) \in E$ for $i \in [0, k-1]$. The shortest path between s and t is denoted as $sp(s, t)$, and $dist_G(s, t) = |sp(s, t)|$ is the shortest distance between two vertices in G. Without ambiguity, $dist_G(s, t)$ is represented by $dist(s, t)$.

Graph partitioning. In this paper, the data graph $G(V, E)$ is divided into a set of vertex-induced subgraphs $\{G_i(V_i, E_i)\}_{i=1}^{k}$ and a set of cutting edges E_{cut}, where $V = \bigcup_{i=1}^{k} V_i$ and $E = \bigcup_{i=1}^{k} E_i \cup E_{cut}$.

For each vertex v, $m(v)$ denotes the machine where v is placed and the adjacency list of v is also stored in the same machine. In this paper, we use $\{G_i\}_{i=1}^{k} \cup E_{cut}$ to denote the distributed graph. Referring to the works [38, 39], we make the following definition to classify all vertices based on the distribution of neighbors.

Definition 1 (**Vertex Category**). *Given a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, V_B and V_I are the sets of boundary vertices and interior vertices respectively, which are defined as follows.*

- $V_B = \{v \mid \exists e(u,v) \in E, m(u) \neq m(v)\}$.
- $V_I = \{v \mid \forall e(u,v) \in E, m(u) = m(v)\}$.
- $V = V_B \cup V_I$ *and* $V_B \cap V_I = \emptyset$.

Similarly, the paths can also be categorized into two categories as follows.

Definition 2 (**Path Category**). *Given a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, the paths between any two vertices s and t, $p(s,t) = \{v_0 = s, v_1, ..., v_{k-1}, v_k = t\}$ can be categorized as follows.*

- *If* $m(v_i) \neq m(v_{i+1})$ *for* $\exists i \in [0, k-1]$, $p^{bd}(s,t)$ *is a boundary path.*
- *If* $m(v_i) = m(v_{i+1})$ *for* $\forall i \in [0, k-1]$, $p^{in}(s,t)$ *is an interior path.*

Problem definition. Given a distributed undirected graph $G(V,E) = \{G_i\}_{i=1}^{k} \cup E_{cut}$ and a vertex pair (s,t), the shortest distance query $q(s,t)$ returns the shortest distance $dist(s,t)$ between s and t.

Example 1. *Fig. 1(a) depicts a partitioning graph G stored in 2 machines. We have*

- *Vertex category: The gray vertices (v_0 to v_6) are the boundary vertices since they have at least one neighbor stored in other machines, while the white vertices (v_7 to v_{15}) are the interior vertices.*
- *Cutting edges: There are a total of 5 cutting edges represented by the dotted lines in Fig. 1(b).*
- *Path category: Given a vertex pair (v_1, v_2), $p^{in} = \langle v_1, v_2 \rangle$ is an interior path since $m(v_1) = m(v_2)$. By contrast, $p^{bd} = \langle v_1, v_3, v_0, v_2 \rangle$ is a boundary path since $m(v_1) \neq m(v_3)$.*
- *Shortest path and distance: For the vertex pair (v_1, v_2), we have $sp(v_1, v_2) = \langle v_1, v_2 \rangle$ and $dist(v_1, v_2) = 1$.*

2.2 2-hop index for Distance Queries

State-of-the-art solutions for shortest distance queries primarily rely on the 2-hop index. To set the context, we begin by providing a brief overview of the 2-hop index [6] before delving into these solutions. The strength of the 2-hop index lies in its query efficiency. It is designed so that the distance of any two connected vertices can be answered within 2 hops using the index.

Index structure. For each vertex $v \in V$, the 2-hop index requires building a label set $L(v)$, which consists of a set of key/value pairs $(u, dist(v,u))$. Then, $\bigcup_{v \in V} L(v)$ is a complete 2-hop index of G when satisfying the 2-hop cover below.

Definition 3 (**2-hop cover [3, 17]**). *For any two vertices s and t, there exists $w \in L(s) \cap L(t)$ such that $dist(s,t) = dist(s,w) + dist(w,t)$.*

Index size. The label set size of each vertex v is the number of entries in $L(v)$. Then, the index size of 2-hop index is $O(n \cdot \delta)$, where δ is the maximum label set size in G [17], i.e., $\delta = \max_{v \in V} |L(v)|$.

Query process. Given a query $q(s,t)$ with $s, t \in V$, the shortest distance [17] between s and t can be calculated via the 2-hop index as

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

Based on Equation 1, the query time complexity of each query $q(s,t)$ is $O(|L(s)| + |L(t)|)$.

Vertex order. The vertex ordering strategy is widely adopted in many 2-hop index-based works to reduce the index size and accelerate the index construction. A typical ranking function $r(\cdot)$ prioritizes vertices with higher degrees and resorts to the original ID to break ties [17]. For example, we have $r(v_0) > r(v_1)$ since $deg(v_0) > deg(v_1)$. The vertex order can also be determined based on other vertex centralities, e.g., betweenness centrality and closeness centrality. Please see [19] as a comprehensive comparison.

Pruned landmark labeling (PLL) [3]. According to a degree-based vertex ordering strategy, PLL, a classic solution based on the 2-hop index, can build a complete 2-hop index that satisfies the following two properties.

- *Completeness.* Given any two vertices s and t, we have $dist(s,t) = \min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v]$.
- *Minimum.* For each label entry $(u, dist(v,u)) \in L(v)$ with $v \in V$, we have $dist(v,u) < dist(v,w) + dist(w,u)$ when $r(w) > r(u)$.

To construct the index, PLL needs to perform the pruned breadth-first-search (BFS) sourced from each vertex sequentially, whose time complexity is $O(\delta^2 \cdot m)$ [17]. It is very time-consuming, especially for large-scale graphs. More importantly, the distributed extension of PLL exhibits two limitations:

- *Serious communication cost.* Due to the graph being distributed across different machines, directly deploying the pruned BFS of PLL in the distributed environment inevitably leads to massive message exchanges, thereby producing serious communication costs.
- *Poor scalability.* Due to the order dependency, PLL needs to sequentially activate each vertex to update label entries. This kind of serial operation definitely wastes the computing resources of the cluster. Moreover, the uneven distribution of received messages can lead to significant differences in computational overhead among machines.

2.3 State-of-the-art Approaches

Parallel shortest distance labeling (PSL) [17]. The PSL approach significantly improves upon PLL on indexing time by breaking the order dependency in the index construction. Specifically, PSL designs a distance dependency-based pruning strategy to update the label entries in parallel whilst guaranteeing the accuracy of query results. In addition, PSL applies two optimization strategies to reduce the memory space of labels. However, the performance bottlenecks mainly exist in the distributed extension of index construction, which includes the following two respects.

- *Serious communication cost.* Just like PLL, all vertices must receive substantial label entries transmitted from their neighbors in every iteration. Moreover, PSL frequently duplicates the latest labels of candidate vertices to other machines during the index construction process. Both these actions lead to significant communication costs and memory overhead since the label set of each vertex remains incomplete until the program concludes.

- *Huge memory cost.* Based on the analysis above, it is inevitable to incur significant space overhead to build the 2-hop index, especially on large-scale graphs. In the distributed environment, due to the 2-hop index being distributed across different machines, the uneven distribution of label entries may cause the memory overflow of a single machine. In addition, the duplication of label entries significantly amplifies memory costs.

Core tree index (CTL) [18]. CTL utilizes tree decomposition techniques to create a core-tree structure, which comprises a core graph and several sub-trees. For the core component, CTL constructs a complete 2-hop index using the PSL method. For the sub-tree part, CTL gathers only the minimal distances from each vertex to all its corresponding ancestor vertices in the tree. When compared to PSL, CTL significantly reduces memory consumption at the cost of a slight decrease in query efficiency. The performance bottlenecks mainly include the following three aspects:

- *Serious indexing cost.* During tree decomposition, CTL must form a series of cliques comprising the neighbors of each candidate vertex [18]. Owing to data distribution across different machines, this step demands frequent inter-machine communication. Furthermore, the inability to execute tree decomposition in parallel leads to prolonged indexing duration, incurring significant costs.
- *Huge memory cost.* Even though tree decomposition reduces the number of vertices in the core graph, the density of this graph surges due to the addition of cliques, leading to substantial memory requirements. For instance, in the core graph of SocLiveJ, the edge count exceeds 8 billion, even as the vertex count dwindles to roughly 300,000. This densely connected core graph also escalates memory demands during index construction, as vertices need to acquire more label entries from neighboring vertices.
- *Workload imbalance.* The core-tree structure is composed of a single core graph and many sub-tree structures. Specifically, the core graph is equipped with a complete 2-hop index. By contrast, a tree-based index is deployed in each sub-tree. Due to the difference in indexing time complexity on these parts, keeping workload balance across all machines during index construction is challenging.

Based on the aforementioned analysis, there is a need for a more effective distributed method that optimizes query efficiency, scalability, and memory utilization.

3 DISTRIBUTED HOP-BASED INDEX

In this section, we first describe the index structure of our distributed hierarchical hop-based index, called DH-Index. Then, we design an efficient query processing algorithm.

3.1 DH-Index Structure

Observation. Consider a vertex pair (s, t).

(1) If $m(s) \neq m(t)$, the shortest path between them is definitely a boundary path traversing through two boundary vertices v_1 and v_2. This path can be denoted as $sp^{bd}(s,t) = sp(s,v_1) \bowtie sp(v_1,v_2) \bowtie sp(v_2,t)$, and the shortest distance between s and t can be formulated as $dist(s,t) = dist(s,v_1) + dist(v_1,v_2) + dist(v_2,t)$.

(2) Conversely, if $m(s) = m(t)$, a shortest interior path, $sp^{in}(s,t)$, might also exist between s and t. Hence, the shortest distance is given by the shorter from the boundary and interior paths: $dist(s,t) = \min\{|sp^{bd}(s,t)|, |sp^{in}(s,t)|\}$.

Example 2. *Reconsider the graph in Fig. 1, and take vertex pairs (v_0, v_1) and (v_5, v_{11}) as examples. For the query $q(v_0, v_1)$, there are two shortest paths which are $sp_0 = \langle v_0, v_3, v_1 \rangle$ and $sp_1 = \langle v_0, v_2, v_1 \rangle$, respectively, both of which are boundary paths. Therefore, we have $dist(v_0, v_1) = 2$. For the query $q(v_5, v_{11})$, we can find that the shortest interior and boundary paths are $sp^{in} = \langle v_5, v_{13}, v_{14}, v_1, v_2, v_{11} \rangle$ and $sp^{bd} = \langle v_5, v_0, v_2, v_{11} \rangle$, respectively. Then, we have $dist(v_5, v_{11}) = \min\{|sp^{bd}|, |sp^{in}|\} = 3$.*

In light of the observation, we design an index that captures the shortest distances of both boundary and interior paths. This is achieved by carefully designed labeling strategies for interior and boundary vertices. Our strategy aims to reduce the index size by minimizing the extensive global connections among vertices, without compromising the accuracy of query results.

A category-aware vertex order. To prioritize vertices that play a critical role in query tasks, we propose a category-aware vertex ordering strategy. This approach stems from the observation that boundary vertices, regardless of whether $m(u)$ equals $m(v)$ or not, have inherent importance in all query tasks $q(u, v)$ and should be ranked higher. Assuming $r(v)$ denotes the ranking function of vertex v, the order $r(u) > r(v)$ is satisfied if:

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

Example 3. *Using the proposed vertex ordering strategy on the vertices in Fig. 1(a), we observe no ranking conflicts between any pairs. For clarity, vertices are ordered as $r(v_0) > r(v_1) > \cdots > r(v_{15})$.*

DH-Index structure. DH-Index is composed of the interior index $DH_I = \bigcup_{v \in V_I} L(v)$ and the boundary index $DH_B = \bigcup_{v \in V_B} L(v)$. We utilize different labeling strategies for boundary and interior vertices to minimize the index size and enhance construction efficiency. Using the ranking function $r(\cdot)$, we define DH_I and DH_B as follows:

Definition 4 (**Interior Index**). *Given a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, DH_I encompasses the 2-hop index of each interior vertex in its respective subgraph. For any vertex $v \in V_I \cap V_i$ and any label $(u, d_{vu}) \in L(v)$, the following conditions hold:*

(1) $m(u) = m(v)$ and $r(u) \geq r(v)$,
(2) $d_{vu} = dist_{G_i}(u, v)$,
(3) For all $(w, d_{vw}) \in L(v)$, if $r(w) > r(u)$ then $d_{vu} < d_{vw} + dist_{G_i}(u, w)$.

Based on DH_I, for each interior vertex pair (s, t) with $m(s) = m(t)$, we have $dist_{G_i}(s,t) = \min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v]$.

Definition 5 (**Boundary Index**). *Given a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, DH_B consists of the 2-hop index for each boundary vertex in V_B. For any vertex $v \in V_B$ and any label $(u, d_{vu}) \in L(v)$, the following conditions are satisfied:*

(1) $r(u) \geq r(v)$,
(2) $d_{vu} = dist(u, v)$,

(3) *For every* $(w, d_{vw}) \in L(v)$, *if* $r(w) > r(u)$ *then* $d_{vu} < d_{vw} + dist(u, w)$.

Based on DH_B, for each boundary vertex pair (s, t), we have $dist(s, t) = \min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v]$.

In summary, the interior index DH_I aims to quickly capture the shortest distances of all interior paths whilst the purpose of DH_B is to get the shortest distance of each boundary vertex pair efficiently. We will demonstrate that the shortest distance of any two vertices can be computed jointly using DH_I and DH_B in Section 3.2 Then, the DH-Index is formally defined as follows.

Definition 6 (**DH-Index**). *The DH-Index of a distributed graph* $\{G_i\}_{i=1}^{k} \cup E_{cut}$ *is composed of the interior index* DH_I *and the boundary index* DH_B.

Example 4. *Table 2 shows the DH-Index of Fig. 1(a). Specifically, the label set of each boundary vertex in* $V_B = \{v_0, ..., v_6\}$ *records the 2-hop index about the related boundary vertices. Meanwhile, the label set of each interior vertex in* $V_I = \{v_7, ..., v_{15}\}$ *holds the minimal interior distances to the vertices placed in the same machine.*

Table 2: The label entries of DH-Index of Fig. 1(a)

ID	Label Entries
v_0	$\{v_0, 0\}$
v_1	$\{v_1, 0\}, \{v_0, 2\}$
v_2	$\{v_2, 0\}, \{v_0, 1\}, \{v_1, 1\}$
v_3	$\{v_3, 0\}, \{v_0, 1\}, \{v_1, 1\}$
v_4	$\{v_4, 0\}, \{v_0, 1\}, \{v_2, 1\}, \{v_1, 2\}$
v_5	$\{v_5, 0\}, \{v_0, 1\}$
v_6	$\{v_6, 0\}, \{v_1, 1\}, \{v_0, 3\}$
v_7	$\{v_7, 0\}, \{v_1, 1\}, \{v_2, 1\}$
v_8	$\{v_8, 0\}, \{v_0, 1\}, \{v_3, 1\}, \{v_6, 3\}$
v_9	$\{v_9, 0\}, \{v_8, 1\}, \{v_0, 2\}, \{v_3, 2\}, \{v_6, 2\}$
v_{10}	$\{v_{10}, 0\}, \{v_7, 1\}, \{v_1, 2\}, \{v_2, 2\}$
v_{11}	$\{v_{11}, 0\}, \{v_2, 1\}, \{v_{10}, 1\}, \{v_1, 2\}$
v_{12}	$\{v_{12}, 0\}, \{v_6, 1\}, \{v_9, 1\}, \{v_8, 2\}, \{v_0, 3\}, \{v_3, 3\}$
v_{13}	$\{v_{13}, 0\}, \{v_5, 1\}, \{v_1, 2\}$
v_{14}	$\{v_{14}, 0\}, \{v_1, 1\}, \{v_{13}, 1\}, \{v_5, 2\}$
v_{15}	$\{v_{15}, 0\}, \{v_0, 1\}, \{v_4, 1\}$

Index size. For a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, the DH-Index size is $O(\delta_I \cdot |V_I| + \delta_B \cdot |V_B|)$, where δ_I and δ_B represent the maximum number of label entries in interior and boundary vertices respectively. When compared with 2-hop index, the inequality $O(\delta_I \cdot |V_I| + \delta_B \cdot |V_B|) \leq O(\delta \cdot |V|)$ holds since δ, the maximum label set size in 2-hop index, surpasses both δ_i and δ_B.

3.2 Query Processing

Given a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, queries $q(s, t)$ using DH-Index for vertex pairs s and t can be categorized into four cases.

Case 1: Two boundary vertices, i.e, $s, t \in V_B$.

- **Report** $\min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v]$.

Since DH_B provides the 2-hop index for boundary vertices over the entire graph, the result for $q(s, t)$ can be directly determined using Equation 1. The time complexity is $O(\delta_B)$.

Example 5. *Take the boundary vertex pair* (v_1, v_5) *as an example. According to the recorded label entries in Table 2, we have* $q(v_1, v_5) = L(v_1)[v_0] + L(v_5)[v_0] = 3$.

Case 2: Only one boundary vertex, i.e, $s \in V_I \cap V_i$ **and** $t \in V_B$.

- Let $V_B^s = L(s) \cap V_B$.
- **Report** $\min_{v \in V_B^s} L(s)[v] + dist(v, t)$.

Let $v \in V_B^s$ be a boundary vertex on any path $p(s, t)$ between s and t. According to the location of s and t, Case 2 can be classified as the following two situations.

- $m(s) \neq m(t)$. Based on the property of boundary path in Definition 2, there exists at least one boundary vertex $u \in V_B \cap V_i$ that is located in $p(s, t)$, i.e., $p(s, t) = p(s, u) \bowtie p(u, t)$. Then, we have $dist(s, t) = \min_{u \in V_B \cap V_i} dist(s, u) + dist(u, t)$. Based on DH_I, we have $dist(s, u) = dist(s, v) + dist(v, u)$ with $r(v) \geq r(u)$. Therefore, we can conclude that

$$\begin{aligned} dist(s,t) &= \min_{u \in V_B \cap V_i} dist(s,u) + dist(u,t) \\ &= \min_{v \in V_B^s, u \in V_B \cap V_i} dist(s,v) + dist(v,u) + dist(u,t) \\ &= \min_{v \in V_B^s} L(s)[v] + dist(v,t), \end{aligned} \tag{2}$$

  where $dist(v, t)$ can be calculated based on Case 1.
- $m(s) = m(t)$. Apart from the boundary path, the shortest interior path also needs examination. Based on the property of DH_I and $L(s) \cap L(t) \subseteq V_B^s$, we have

$$\begin{aligned} dist(s,t) = dist_{G_i}(s,t) &= \min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v] \\ &= \min_{v \in V_B^s} L(s)[v] + dist(v,t). \end{aligned} \tag{3}$$

Based on the analysis of these two situations, we can conclude that $dist(s, t) = \min_{v \in V_B^s} L(s)[v] + dist(v, t)$, and the time complexity can be bounded by $O(|V_B| \cdot \delta_B)$.

Example 6. *Given two vertices* $s = v_{10}$ *and* $t = v_1$, *we have* $V_B^s = \{v_1, v_2\}$ *and* $dist(v_1, v_{10}) = \min_{v \in V_B^s} L(s)[v] + dist(v, t) = \min\{L(s)[v_1] + dist(v_1, v_1), L(s)[v_2] + dist(v_2, v_1)\} = 2$.

Case 3: Two interior vertices within different machines, i.e., $s, t \in V_I$ **and** $m(s) \neq m(t)$.

- Let $V_B^s = L(s) \cap V_B$ and $V_B^t = L(t) \cap V_B$.
- **Report** $min_{v \in V_B^s, u \in V_B^t} L(s)[v] + dist(v, u) + L(t)[u]$.

Let $v \in V_B^s$ be a boundary vertex located in any $p(s, t)$ between s and t. Based on the analysis of Case 2, we have $dist(s, w) = \min_{v \in V_B^s} L(s)[v] + dist(v, w)$ where $w \in V_B$ and $m(w) = m(t)$. Similarly, $dist(w, t) = \min_{u \in V_B^t} L(t)[u] + dist(w, u)$. Then, we can conclude that

$$\begin{aligned} dist(s,t) &= \min_{w \in p(s,t) \cap V_B} dist(s,w) + dist(w,t) \\ &= \min_{v \in V_B^s} L(s)[v] + dist(v,w) + \min_{u \in V_B^t} L(t)[u] + dist(w,u) \\ &= \min_{v \in V_B^s, u \in V_B^t} L(s)[v] + dist(v,u) + L(t)[u] \end{aligned} \tag{4}$$

The time complexity is $O\big(|V_B^s| \cdot |V_B^t| \cdot (|L(v)| + |L(u)|)\big)$ which can be bounded by $O(|V_B| \cdot \delta_B)$.

Example 7. *Take the query* $q(s, t)$ *with* $s = v_7$ *and* $t = v_8$ *as an example. We have* $V_B^s = \{v_1, v_2\}$, $V_B^t = \{v_0, v_3, v_6\}$, *and*

$$dist(s,t) = \min_{v \in V_B^s, u \in V_B^t} L(s)[v] + dist(v,u) + L(t)[u] = 3. \tag{5}$$

Case 4: Two interior vertices within the same machine, i.e., $s, t \in V_I$ and $m(s) = m(t)$.

- Let d_1 be the minimal boundary distance calculated by Equation 4.
- Let $d_2 = \min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v]$ be the minimal interior distance.
- **Report** $\min\{d_1, d_2\}$.

Similar to the second situation in Case 3, we calculated the shortest boundary distance d_1 based on Equation 4. Then, the distance of the shortest interior path can be calculated as $d_2 = \min_{v \in L(s) \cap L(t)} L(s)[v] + L(t)[v]$. Finally, $dist(s,t) = \min\{d_1, d_2\}$ and the time complexity is $O(|V_B^s| \cdot |V_B^t| \cdot (|L(v)| + |L(u)|) + |L(s)| + |L(t)|)$ which is simplified as $O(|V_B| \cdot \delta_B)$.

Example 8. *Take $q(v_7, v_{10})$ as an example. Based on Equation 4, the shortest boundary distance is $d_1 = 3$. Similarly, the shortest interior distance is $d_2 = 1$. Therefore, we have $dist(v_7, v_{10}) = \min\{d_1, d_2\} = 1$.*

Query algorithm. In the distributed environment, the label entries are placed in different machines. Therefore, it is difficult to adopt the centralized searching strategy to resolve the above query tasks. To address this issue, we design a DH-Index-based bidirectional distributed query algorithm (DHQA) to resolve the query tasks within limited rounds.

Algorithm 1 presents the pseudo-code of DHQA which outputs $dist(s,t)$ based on DH-Index within three supersteps. Here, $d_v(s)$ and $d_v(t)$ are used to record the distances from v to s and t, respectively. The details are shown as follows.

- When $superstep = 0$, the program is activated from the vertices s and t. Take the vertex s as an example. For each label entry $(u, d_{us}) \in L(s)$ and $m(u) = m(s)$, the message $\langle s, d_{us} \rangle$ is sent to the vertex u (Lines 5-7). Note that this step does not involve the message exchange among different machines, thereby not producing the communication cost.
- When $superstep = 1$, the target vertex v updates $d_v(s)$ and $d_v(t)$ based on the received messages (Lines 9-10). Then, the message $\langle w, d_{wv} + d_v(s) \rangle$ is sent to w when satisfying (i) $d_v(s) < \infty$ (Line 11), (ii) $(w, d_{wv}) \in L(v)$ (Line 12), and (iii) $r(w) > r^*$ (Line 12). Note that the condition (iii) helps to reduce communication cost.
- When $superstep = 2$, the minimal distances from v to s and t are further updated based on the received messages (Lines 15-16). Then, DHQA outputs $dist(s,t) = \min_{v \in V_T} d_s(v) + d_t(v)$ where $V_T = \{v | d_s(v) + d_t(v) < \infty\}$.

Correctness analysis. To prove the correctness of DHQA, we first prove that the DH-Index-based hop number of each shortest path $sp(s,t)$, denoted as $dhop(sp(s,t))$, is no more than 4. Without specifying, $dhop(sp(s,t))$ is simplified as $dhop(sp)$.

Lemma 1. *Given an interior vertex pair (s,t) with $m(s) = m(t) = i$, we have $dhop(sp) \leq 2$ if $dist(s,t) = dist_{G_i}(s,t)$.*

Proof. When $dist(s,t) = dist_{G_i}(s,t)$, $sp(s,t)$ is proved as an interior path. Based on Definition 4, each interior vertex pair is constructed as a 2-hop connection at most, i.e., $dhop(sp) \leq 2$. □

Lemma 2. *Given a boundary shortest path $sp(s,t)$, it holds that $dhop(sp) \leq 4$.*

Algorithm 1: DH-Index-based bidirectional Query Algorithm (DHQA)

Input: DH-Index, $q(s,t)$.
Output: $dist(s,t)$.

1 Initial $d(s,t) \leftarrow \infty$, $r^* \leftarrow \max\{r(s), r(t)\}$
2 **foreach** *vertex* $v \in V$ **do**
3   **if** *superstep*=0 **then**
4   Initial $d_v(s) \leftarrow \infty$ and $d_v(t) \leftarrow \infty$
5   **if** $v \in \{s, t\}$ **then**
6   **foreach** $(u, d_{uv}) \in L(v)$ **do**
7   Send $\langle v, d_{uv} \rangle$ to u with $m(u)$=$m(v)$
8   **else if** *superstep*=1 **then**
9   **foreach** *reveived message* $\langle tgt, d_{new} \rangle$ *with tgt* $\in \{s, t\}$ **do**
10             $d_v(tgt) \leftarrow \min\{d_v(tgt), d_{new}\}$
11   **if** $v \in V_B$ *and* $d_v(tgt) \neq \infty$ **then**
12   **foreach** $(w, d_{wv}) \in L(v)$ *with* $r(w) \geq r^*$ **do**
13   Send $\langle tgt, d_v(tgt) + d_{wv} \rangle$ to w
14   **else**
15   **foreach** *reveived message* $\langle tgt, d_{new} \rangle$ **do**
16             $d_v(tgt) \leftarrow \min\{d_v(tgt), d_{new}\}$
17         $V_T.insert(v)$ with $d_v(s) + d_v(t) < \infty$
18 **if** $V_T \neq \emptyset$ **then** $dist(s,t) \leftarrow \min_{v \in V_T} d_v(s) + d_v(t)$;
19 **return** $dist(s,t)$

Proof. Based on the analysis of Case 3, for each shortest boundary path $sp^{bd}(s,t)$, we have $sp^{bd}(s,t) = sp^{in}(s,v) \bowtie sp^{bd}(v,u) \bowtie sp^{in}(u,t)$, where $m(v) = m(s)$, $m(u) = m(t)$, and $v, u \in V_B$. Based on the properties of DH_I and DH_B, the hop numbers of three subpaths are no more than 2.

We first consider the scenario that $dhop(sp^{in}(s,v)) = 2$ and $dhop(sp^{in}(u,t)) = 2$. For $sp^{in}(s,v)$, we have $dist^{in}(s,v)$=$L(s)[w_1]$+$L(v)[w_1]$, where w_1 needs to have a higher rank than v, and then is a boundary vertex. Similarly, there exists a boundary vertex w_2 such that $dist^{in}(u,t) = L(u)[w_2] + L(t)[w_2]$. Hence, $sp(s,t)$ can be reformulated as:

$$\begin{aligned} sp(s,t) &= sp^{in}(s,v) \bowtie sp^{bd}(v,u) \bowtie sp^{in}(u,t) \\ &= sp^{in}(s,w_1) \bowtie sp^{in}(w_1,v) \bowtie sp^{bd}(v,u) \bowtie sp^{in}(u,w_2) \bowtie sp^{in}(w_2,t) \\ &= sp_1^{in}(s,w_1) \bowtie sp_2^{bd}(w_1,w_2) \bowtie sp_3^{in}(w_2,t). \end{aligned} \quad (6)$$

Based on the property of DH_B, we have $dhop(sp_2) \leq 2$, as $w_1, w_2 \in V_B$. Therefore, we can conclude that $dhop(s,t) = dhop(sp_1) + dhop(sp_2) + dhop(sp_3) \leq 4$.

For other cases, i.e., $dhop(sp^{in}(s,v)) < 2$ or/and $dhop(sp^{in}(u,t)) < 2$, $dhop(s,t)$ can also be bounded by 4. □

Theorem 1 (Correctness). *The query result of each task in DHQA is correct.*

Proof. According to Lemma 1 and Lemma 2, for any two connected vertices s and t, there is at least one shortest path $sp(s,t)$ that satisfies $dhop(sp) \leq 4$. Since DHQA adopts a bidirectional searching strategy, there is definitely a middle vertex v that simultaneously receives the messages of $\langle s, d_1 \rangle$ and $\langle t, d_2 \rangle$ in the third supersteps, where $dist(s,t) = d_1 + d_2$. Otherwise, if s and t are not connected, the result of $q(s,t)$ is ∞. Therefore, the query result of $q(s,t)$ in DHQA is correct. □

4 INDEX CONSTRUCTION

One of the most naive methods to build DH-Index is adopting the PSL method with new pruning strategies. Specifically, all vertices are activated to exchange messages and update label entries in parallel until finishing DH-Index. The communication cost of building DH_I can be avoided since the interior vertices of each subgraph are placed in the same machine. However, this strategy faces the following two disadvantages when building the boundary index DH_B.

- *Redundant path traversal.* When directly building DH_B in the whole graph G, it is inevitable to traverse the interior vertices which are located on the shortest paths among any two boundary vertices. Take the boundary vertex pair (v_1, v_5) on Fig. 1(a) as an example. it is inevitable to visit the interior shortest path $\langle v_1, v_{14}, v_{13}, v_5 \rangle$. During the index construction, it is necessary for these interior vertices to add the label entries originating from the activated boundary vertices, thus increasing the indexing time and space cost.
- *Huge communication cost.* Similar to the 2-hop index, it needs to frequently exchange the label entries of boundary vertices among different machines for building DH_B, thus producing huge communication costs.

To overcome the above challenges, we introduce a distributed hierarchical index construction algorithm (called DHCA) to build DH-Index. The overview of DHCA is shown in Fig. 2, where the three key steps include the construction of the interior index DH_I, the boundary graph G_B, and the boundary index DH_B. The details are listed as follows.

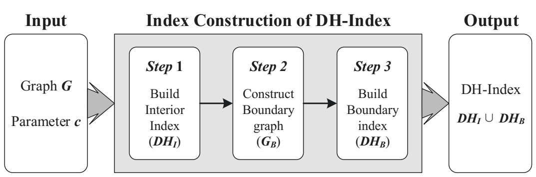


Figure 2: An overview of the DHCA method

Construction of DH_I. The interior index DH_I can be built on all subgraphs in parallel based on PSL since each vertex only needs to collect the label entries to vertices within the same subgraph. Table 3 records the process of building DH_I on two subgraphs. First, for each vertex v, we add the label entry $(v, 0)$ into $L(v)$, and then make a 2-hop cover verification for each new label entry (u, d_{uv}) originating from the neighbors of v. For example, the label entry $(v_1, 1)$ is added in the label sets $L(v_2)$, $L(v_7)$, and $L(v_{14})$ when 1-hop labels are processed. Similarly, when processing 2-hop labels, the label entry $(v_1, 2)$ is added in the label sets $L(v_{10})$, $L(v_{11})$, and $L(v_{13})$. This process can be terminated until there are no new label entries. Finally, we have $DH_I = \bigcup_{v \in V_I} L(v)$.

Construction of boundary graph G_B. To build the boundary index DH_B efficiently, it's crucial to bypass redundant path traversals. In light of this, we introduce the boundary graph G_B. This graph integrates the boundary vertex set V_B, cutting edges E_{cut}, and interior-path edges E_{ip} that represent the shortest interior paths between boundary vertices within the same machine. Crucially, G_B is structured to minimize redundant path traversals during the DH_B construction.

Table 3: The label entries of all vertices when building DH_I

(a) the 2-hop index on G_1

ID	0	1	2	3
v_1	v_1	–	–	–
v_2	v_2	v_1	–	–
v_5	v_5	–	–	v_1
v_7	v_7	v_1, v_2	–	–
v_{10}	v_{10}	v_7	v_1, v_2	–
v_{11}	v_{11}	v_2, v_{10}	v_1	–
v_{13}	v_{13}	v_5	v_1	–
v_{14}	v_{14}	v_1, v_{13}	v_5	–

(b) the 2-hop index on G_2

ID	0	1	2	3	4
v_0	v_0	–	–	–	–
v_3	v_3	v_0	–	–	–
v_4	v_4	v_0	–	–	–
v_6	v_6	–	–	–	v_0, v_3
v_8	v_8	v_0, v_3	–	v_6	–
v_9	v_9	v_8	v_0, v_3, v_6	–	–
v_{12}	v_{12}	v_6, v_9	v_8	v_0, v_3	–
v_{15}	v_{15}	v_0, v_4	–	–	–

First, we formally define the interior-path edges E_{ip} as follows.

Definition 7 (**Interior-path Edges**). *Let's consider a set of subgraphs $\{G_i\}_{i=1}^{k}$. The interior-path edge set E_{ip} is defined as the collection of derived edges, formally given by:*

$$E_{ip} = \bigcup_{i=1}^{k} \bigcup_{u,v \in V_i} e^i(u, v).$$

Here, each edge $e^i(u, v)$ must meet the following conditions:

1) $u, v \in V_B$ and $m(u) = m(v) = i$,
2) The weight of $e^i(u, v)$ satisfies $W_{u,v} = dist_{G_i}(u, v) < \infty$,
3) $\forall w \in V_B \cap V_i$, we have $dist_{G_i}(u, v) < dist_{G_i}(u, w) + dist_{G_i}(w, v)$.

Conditions 1) and 2) ensure E_{ip} contains the shortest interior distances for boundary vertex pairs within the same machine. To minimize E_{ip}'s space, Condition 3) introduces a ranking constraint.

Following the three conditions, E_{ip} can be efficiently constructed. During DH_I construction, each boundary vertex v also forms its 2-hop label entries. Thus, for any $u, v \in V_i \cap V_B$, the distance

$$dist_{G_i}(u, v) = \min_{w \in L(u) \cap L(v)} L(u)[w] + L(v)[w],$$

is computed to identify edges satisfying Conditions 1) and 2). For Condition 3), each edge $e_r(u, v)$ is evaluated to check if it traverses another boundary vertex w where $d_{uw} + d_{wv} = d_{uv}$, further refining E_{ip}'s size. We then formally define the boundary graph G_B:

Definition 8 (**Boundary Graph**). *For a distributed graph $\{G_i\}_{i=1}^{k} \cup E_{cut}$, and the corresponding DH_I, the boundary graph $G_B(V_B, E_B)$ is defined such that $V_B \subseteq V$ and $E_B = E_{cut} \cup E_{ip}$.*

Lemma 3. *Given any two boundary vertices s and t, we have $dist(s, t) = dist_{G_B}(s, t)$.*

Proof. Given a shortest path $sp(s, t) = \{v_i\}_{i \in [0,l]}$ where $v_0 = s, v_l = t$, and $e(v_i, v_{i+1}) \in E$. Let $\{u_j\}_{j \in [0,k]}$ denote all boundary vertices from $sp(s, t)$ that satisfies

- $u_0 = s, u_k = t$;
- $\sum_{j \in [0,k-1]} dist(u_j, u_{j+1}) = dist(s, t)$.

Then, for each $j \in [0, k-1]$, we have that $dist(u_j, u_{j+1})$ is equal to

(1) 1, if $m(u_j) \neq m(u_{j+1})$;
(2) $dist_{G_j}(u_j, u_{j+1})$, if $m(u_j) = m(u_{j+1}) = j$.

Case (2) holds because $dist(u_j, u_{j+1})$ equals the distance of the shortest interior path between u_j and u_{j+1} when $m(u_j) = m(u_{j+1})$. Based on the definitions of E_{ip} and G_B, for any two adjacent boundary vertices u_j and u_{j+1}, we have

- If $m(u_j) \neq m(u_{j+1})$, $e(u_j, u_{j+1}) \in E_{cut}$;
- Otherwise, $e(u_j, u_{j+1}) \in E_{ip}$;

Therefore, $\forall j \in [0, k-1]$, $dist(u_j, u_{j+1}) = W_{u_j u_{j+1}}$ holds and

$$\begin{aligned} dist(s,t) &= \sum_{i\in[0,l-1]} dist(v_i, v_{i+1}) = \sum_{j\in[0,k-1]} dist(u_j, u_{j+1}) \\ &= \sum_{j\in[0,k-1]} W_{u_j u_{j+1}} = dist_{G_B}(s,t), \end{aligned} \tag{7}$$

□

Example 9. *Take the shortest path* $sp(v_5, v_6) = \{v_5, v_{13}, v_{14}, v_1, v_6\}$ *as an example. This path passes the boundary vertices* v_5, v_1, *and* v_6, *where* $dist(v_5, v_1) = W_{v_5 v_1} = 3$ *and* $dist(v_1, v_6) = W_{v_1 v_6} = 1$. *Therefore, we have* $dist(v_5, v_6) = dist_{G_B}(v_5, v_6) = 4$.

Construction of DH_B. After getting the boundary graph G_B, we further build the boundary index on this graph. It's important to note that the edges in G_B can have varying weights. For instance, as illustrated in Fig. 1 (b), the weights of edges $e(v_1, v_5)$ and $e(v_3, v_6)$ are 3 and 4, respectively. Building indices on weighted graphs introduces challenges: PSL [17] can halt prematurely on weighted graphs, and its extension, WPSL [42], might collect redundant labels not on any shortest path. More importantly, it is inevitable to waste computing resources when directly applying PSL to build the boundary index on G_B.

Algorithm 2: Parallel Vertex-based Construction (PVC)

Input: $G(V, E)$, V_{tgt}
Output: $\cup_{v\in V} L(v)$
1 $dis \leftarrow 1$ and $L(v_t).insert(\{v_t, 0\})$ with $\forall v_t \in V_{tgt}$
2 **while** *True* **do**
3 $\quad Flag \leftarrow false$
4 $\quad$ **foreach** $u \in N(v)$ ***in parallel*** **do**
5 $\quad\quad L^*(u) = \{(w, d_{uw}) | d_{uw} > dis - W_{vu}\}$
6 $\quad\quad$ **if** $L^*(u) \neq \emptyset$ **then** $flag \leftarrow true$;
7 $\quad\quad$ **for** $\forall (w, dis - W_{vu}) \in L(u)$ *with* $r(w) > r(v)$ **do**
8 $\quad\quad\quad$ **if** $Query(w, v, L) > dis$ **then**
9 $\quad\quad\quad\quad L(v).insert((w, dis))$
10 $\quad dis \leftarrow dis + 1$
11 $\quad$ **if** $flag = false$ **then** break;
12 **return** $\bigcup_{v\in V} L(v)$

To address the challenges above, we present the parallel vertex-based construction algorithm (PVC) for building the boundary index DH_B on G_B in all machines in parallel, which adopts a vertex-based computing paradigm to iteratively update the label entries of all vertices.

As shown in Algorithm 2, in the *dis*-th round, each vertex v collects the label entries (w, d_{uw}) originating its neighbor u with $d_{uw} = dis - W_{vu}$ (Lines 7-9), which guarantees that (w, d_{uw}) cannot be dominated by the subsequent labels. Compared to PSL, PVC decides whether to terminate after this iteration by checking the number of candidate labels to be added in future rounds (Lines 5-6). The index construction process can be terminated when $L^*(\cdot) = \emptyset$ for all vertices, which helps to guarantee the completeness of DH_B on the weighted graphs.

In addition, our approach encompasses a task division strategy, aiming to harness the full computational power of all machines while minimizing communication overhead. The core idea is that only the vertices in V_{tgt} are activated in the first round (Line 1). Correspondingly, each vertex $v \in V_B$ only collects the label entries (w, d_{vw}) where $w \in V_{tgt}$ based on the 2-hop cover property (Line 8). Based on this design, the index construction of DH_B is executed in all machines in parallel. Note that PVC can also be applied to build the interior index DH_I by setting $V_{tgt} = V_i$ in the i-th machine.

Next, we establish that any two connected boundary vertices u and v can be reached within 2 hops via DH_B provided by Algorithm 2. This is achieved by demonstrating that DH_B is a superset of the 2-hop index generated by PSL.

Lemma 4. *Let* $L_1 = L(v)$ *and* $L_2 = \bigcup_{i=1}^{k} L^i(v)$ *be the label entries of* v *built from the boundary graph based on PSL and PVC, respectively. We have* $L_1 \subseteq L_2$.

Proof. Due to the minimum property of PSL [17] inherited from PLL [3], each label $(w, d_{wv}) \in L_1$ is not dominated by other label entries, and we have $dist(v, w) = d_{wv}$. Based on the verification of the 2-hop cover in PVC, this label is absolutely added into L_2 in the machine that w is activated. Therefore, it can prove that $L_1 \subseteq L_2$ with $v \in V_B$. Building upon Lemma 4, we can confidently assert that $dist_{G_B}(u, v) = Query(u, v, DH_B)$. □

Table 4: The process of building DH_B in G_B

ID	$dis = 0$		$dis = 1$		$dis = 2$		$dis = 3$	
	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2
v_0	v_0	−	−	−	−	−	−	−
v_1	−	v_1	−	−	v_0	−	−	−
v_2	v_2	−	v_0	v_1	−	−	−	−
v_3	−	v_3	v_0	v_1	−	−	−	−
v_4	v_4	−	v_0, v_2	−	−	v_1	−	−
v_5	−	v_5	v_0	−	−	−	−	v_1
v_6	v_6	−	−	v_1	−	−	v_0	−

Algorithm 3: Distributed hierarchical-based algorithm (DHCA)

Input: $G(V, E)$, c
Output: DH-Index of G
Get $\{G_i\}_{i=1}^{k}$, E_{cut}, $r(v)$ based on $G(V, E)$
foreach *subgaph* $G_i(V_i, E_i)$ *with* $i \in [1, k]$ ***in parallel*** **do**
$\quad \bigcup_{v\in V_i} L(v) \leftarrow PVC(G_i, V_i)$
$DH_I = \bigcup_{v\in V_I} L(v)$ and $E_{ip} \leftarrow PathBuild(\bigcup_{v\in V_B} L(v))$
Clear $L(v)$ for all $v \in V_B$ and duplicate $G_B(V_B, E_B)$ in all machines where $E_B = E_{cut} \cup E_{ip}$
foreach *graph* G_B *in* M_i *with* $i \in [1, k]$ **do**
$\quad$ **foreach** $ID(v)\%k = i$ *or* $ID(v) < c \cdot |V_B|$ **do** $V_{tgt}.push(v)$;
$\quad \bigcup_{v\in V_B} L^i(v) \leftarrow PVC(G_B, V_{tgt})$
$DH_B = \bigcup_{i=1}^{k} \bigcup_{v\in V_B} L^i(v)$
return *DH-Index* $\leftarrow DH_I \cup DH_B$

Procedure $PathBuild(\bigcup_{v\in V_B} L(v))$
foreach $v \in V_B$ *and* $(u, d_{uv}) \in L(v)$ *with* $u \neq v$ **do**
$\quad$ **if** $\nexists w \in V_B$ *with* $d_{vw} + d_{uw} = d_{uv}$ **then**
$\quad\quad$ Get $e_r(u, v)$ with $W_{uv} = d_{uv}$
$\quad\quad E_{ip}.insert(e_r(u, v))$
return E_{ip}

Table 4 records the process of building DH_B in G_B. To reduce the size of L_2, i.e., DH_B, we duplicate some high-rank vertices in V_{tgt} on all machines to prune the redundant labels, which will be detailed in the following overall algorithm.

Overall algorithm. Algorithm 3 shows the whole process of constructing DH-Index. Given a graph $G(V, E)$ with an initial partitioning result $\{m(v)\}_{v \in V}$, we first get the subgraph set $\{G_i\}_{i=1}^{k}$, the cutting edge set E_{cut}, and the ranking value $r(v)$ of each vertex $v \in V$ (Line 1). Next, the 2-hop indexes of all subgraphs are built in parallel, where $DH_I = \bigcup_{v \in V_I} L(v)$ (Lines 2-4). Meanwhile, the interior-path edge set E_{ip} is constructed based on $PathBuild(\cdot)$ that helps to build the boundary graph $G_B(V_B, E_B)$ (Line 4).

Furthermore, the activated vertex set V_{tgt} of each machine is confirmed based on V_B and the parameter c which is the ratio of duplicated vertices (Line 7). Since PVC inevitably generates redundant label entries which can be dominated by the labels in other machines, duplicating a part of boundary vertices with higher vertex orders can reduce the redundant label entries collected. Meanwhile, the computations originating from the activated vertices in each machine may be more heavy, thus increasing the indexing time. The empirical effect of c will be reported in Exp-6. By tuning c, PVC actually achieves a trade-off between indexing time and space cost, whilst not damaging the accuracy of query results. Finally, the boundary index $DH_B = \bigcup_{v \in V_B} L(v)$ is constructed based on PVC in parallel (Line 8), and DH-Index can be obtained by combining DH_I and DH_B (Line 9).

Time complexity. The time complexity of PLL is $O(\delta^2 \cdot m)$ where $\delta = \max_{v \in V} |L(v)|$. Assuming that m^* is the maximal number of edges among all subgraphs, the time complexity of DHCA is $O(\delta_I^2 \cdot m^* + \delta_B^2 \cdot |E_B|)$ where δ_I and δ_B are the maximal numbers of label entries among interior and boundary vertices, respectively.

5 EXPERIMENTS

In this part, we conduct extensive experiments to evaluate the performance of our methods. Section 5.1 introduces the setup of our experiments, followed by the experimental results in Section 5.2.

5.1 Experimental Setup

Datasets. In the experiments, we employ 7 real-life datasets. As shown in Table 5, SP and LJ are downloaded from Stanford Large Network data set Collection[1], and the other datasets are downloaded from Network Repository[2].

Table 5: Statistic of Real-world Graphs

Alias	Dataset	$\|V\|$	$\|E\|$	d_{avg}	Ratio	Type
SP	SocPokec	1.6M	30.6M	27	0.71	Social Network
LJ	SocLiveJ	4.8M	42.8M	17	0.425	Social Network
ID	Indochina	7.4 M	194.1 M	40	0.02	Web Graph
U2	UK2002	18.5M	298.1M	28	0.04	Web Graph
U5	UK2005	39.4M	936.1M	39	0.049	Web Graph
IT	IT2004	41.3M	1.15B	49	0.04	Web Graph
SK	SK2005	50.6M	1.94B	57	0.174	Web Graph
U6	UK2006	77.7M	2.96B	39	0.025	Web Graph
U7	UK2007	105.9M	3.74B	34	0.024	Web Graph
UN	UK0607	133.6M	5.51B	41	0.075	Web Graph

Algorithms. We compare the following algorithms:

- **BiBFS**. The distributed bidirectional searching method.
- **PSL [17]**. The parallelized distance labeling technique.
- **CTL [18]**. The SOTA centralized labeling technique.
- **DPSL**. The distributed extension of PSL.
- **DH-Index**. Our method which is introduced in Section 3.

Details of DPSL. To build the complete 2-hop index in the distributed setting, we adopt a vertex-centric computing approach, comprising the following three steps:

- *Vertex distribution.* The vertices are distributed to the corresponding machines based on a given partitioning result.
- *Message exchange.* During each round, each vertex v receives the label entries $(w, d_{uw}) \subseteq L(u)$ originating from each neighbor u of v where $r(w) > r(v)$. Here, d_{uw} denotes the current shortest distance between u and w. As iterations proceed, this distance is refined and will eventually represent the accurate distance $dist(u, v)$. Note that, this process will produce the communication cost if the vertices v and u are placed in different machines.
- *Label update.* The update of label entries relies on the 2-hop cover property. Specifically, for each vertex v and each label (w, d_{uw}) collected from neighbors, the label entry $(w, d_{uw} + 1)$ can be added in $L(v)$ when satisfying $Query(v, w, L) > d_{uw} + 1$. To minimize communication costs, label entries for each vertex are cached across all machines during the whole index construction phase.

Distributed query process. Consider that the performance of distributed queries is inferior to that of centralized querying. This is because distributed querying inevitably exchanges messages among different computing nodes whilst the centralized query only involves the computations in a same memory space. To keep fairness, the query process of all methods is executed in the distributed setting.

For a given query $q(s, t)$ in **BiBFS**, the procedure is first activated from the vertices s and t, and then forwardly transmitted the key/value pairs to their neighbors, respectively. This procedure is terminated when there is at least one vertex that simultaneously receives the messages originating from s and t.

For DPSL, the labeling entries are distributed to the machines where the corresponding vertices are placed. Then, we execute the bidirectional search to get the query result within two supersteps. Therefore, the index size and query time of PSL and DPSL are the same.

For CTL, we adopt the same bidirectional search strategy in Algorithm 1 to get the query results. Compared with DH-Index, CTL needs to exchange the messages twice since the vertices in the same tree are possibly distributed into different machines.

Environment. All distributed algorithms are implemented in a distributed graph computing system Blogel [35] which performs computational tasks in a superstep fashion. Blogel is deployed in a local cluster with 10 computing nodes[3] with AMD 2.6 GHz and 128 GB memory. The communication among machines is achieved by MPI. All algorithms are implemented in C++ and compiled with O3-level optimization. By contrast, PSL and CTL are deployed in the machine with 40 cores and 500 GB, and supported by the OpenMP framework.

[1] http://snap.stanford.edu/data/
[2] http://networkrepository.com/index.php
[3] In this paper, we use the node to represent the machine in a cluster.

Setting. Unless stated otherwise, we adopt the KaHIP [31] method to generate the initial partitioning result which achieves a trade-off between good locality and vertex balance. In addition, the parameter c is set as 0.02 by default (For "ukunion", c is set as 0.0002). The experimental result is marked as "N/A" when an algorithm runs out of memory. For query evaluation, we randomly select 10^6 vertex pairs (s, t) from each dataset under consideration.

Statement. The DH-Index method can be easily extended to handle weighted graphs and directed graphs. In addition, this strategy can also be extended to resolve the problem of shortest path query.

5.2 Experimental results

Exp-1: Indexing time. In this part, we evaluate the indexing time of all methods. The experimental results are shown in Fig. 3, where DH-Index basically outperforms the other methods in terms of indexing time. On many datasets, DH-Index achieves up to 65.5×, 4.8×, and 5.1× speedup compared to DPSL, PSL, and CTL in terms of indexing time, respectively. The superior performance of DH-Index can be attributed to its effective design, which avoids frequent communication costs and fully utilizes the computing resources of the cluster. Moreover, the construction of the boundary index can be executed in parallel, thereby accelerating the index construction. By contrast, DPSL's lengthy indexing time stems from frequent communication and 2-hop cover verification, necessitating extensive label message exchanges across machines and label entry duplication to all nodes, which is constrained by each machine's memory capacity, significantly impeding DPSL's practicality. In addition, PSL is only executed in a single computing node, thus increasing the indexing time. More importantly, DPSL and PSL also encounter out-of-memory issues on many datasets, which are caused by the huge size of the centralized 2-hop index on all vertices. By contrast, DH-Index can be finished on all datasets. This is because the 2-hop index is decomposed into the 4-hop index based on interior and boundary vertices, and then constructed on different machines, thus reducing indexing cost.

For CTL, although the number of core vertices can be largely reduced, it is also time-consuming to build the 2-hop index for CTL on the core graph. This is because (1) the core graph is dense, where the number of edges is only slightly less than the original graph, and (2) the limited computing resources in a centralized environment.

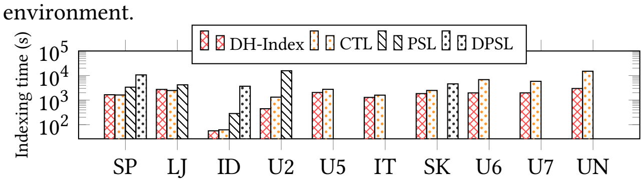


Figure 3: Indexing time (s) on all datasets

Exp-2: Index size. In this part, we evaluate the index size of DH-Index, PSL, and CTL on all datasets since the label size of DPSL and PSL is the same. As shown in Fig. 4, DH-Index significantly outperformed PSL, achieving an average 3.5× reduction, whilst CTL outperformed DH-Index, achieving an average 1.9× reduction. DH-Index successfully built indexes on all datasets, while PSL struggled with memory issues on many datasets. This efficiency is due to DH-Index's innovative design, which is mainly composed of two reasons. First, due to the design of DH-Index, the 2-hop index is decomposed into the 4-hop index based on interior and boundary vertices, thus largely reducing the number of label entries in interior vertices. Second, during the boundary index construction in DH-Index, redundant label entries are efficiently pruned. This is achieved by strategically duplicating certain boundary vertices with higher ranks, across all machines. Therefore, DH-Index can utilize all memory space of the cluster to evenly store the index, which helps to avoid encountering out-of-memory issues. Considering that CTL constructed the tree index for all tree nodes, its index size is naturally smaller than that of DH-Index since the size of the tree index is much less than that of the hop-based index.

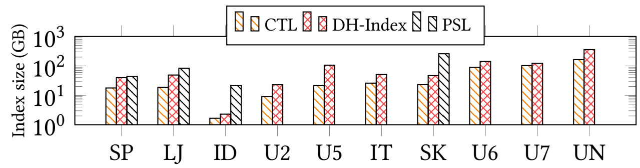


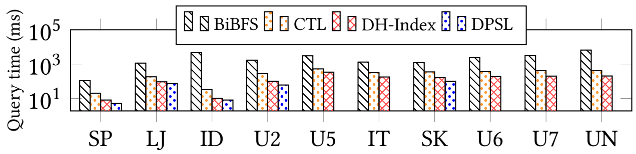
Figure 4: Index size (GB) on all datasets

Exp-3: Query time. Fig. 5 shows the average query time of BiBFS, DPSL, DH-Index, and CTL on two types of query tasks. Specifically, the query time of DH-Index is comparable to that of DPSL (lower than that of CTL) on many datasets. Although DH-Index always takes 3 supersteps to get the query result, which is larger than the 2 supersteps in DPSL, the first step in DH-Index does not involve the message exchange among different machines, thereby reducing the query costs. By contrast, CTL can also finish all query tasks within 3 supersteps, this strategy must exchange messages among different machines twice since the nodes need to be evenly divided into the cluster for workload balance. Therefore, it is inevitable for CTL to take more serious costs, especially for the query tasks with two tree nodes.

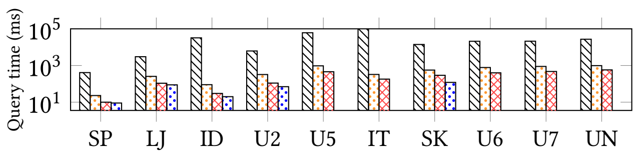
Examining Fig. 5 (a) and Fig. 5 (b), we can find that the improvement of DH-Index over BiBFS is more significant with average speedup of 305.9× when handling the query tasks $q(s, t)$ with $dist(s, t) > 6$ compared to the speedup of 20.5× when $dist(s, t) \leq 6$. The reason is when $dist(s, t) \leq 6$, BiBFS only needs 3 supersteps to get the query result and DH-Index also needs 3 supersteps. However, when $dist(s, t) > 6$, BiBFS always exhibits terrible performance because more supersteps are needed and massive redundant paths are explored, while DH-Index can still report the result within 3 supersteps.

Exp-4: Communication cost. In this part, we evaluate the communication costs of BiBFS, DPSL, CTL, and DH-Index. As shown in Fig. 6, when solving two kinds of queries, i.e., $dist(s, t) \leq 6$ and $dist(s, t) > 6$, DH-Index achieves up to 2 and 3 orders of magnitude less communication cost compared to BiBFS, respectively. In addition, the communication cost of DH-Index is lower than that of CTL on all datasets.

Similar to **Exp-3**, when handling the query tasks with long distances, it is inevitable for BiBFS to transmit more messages to unrelated vertices that are not located in any shortest path, thus heavily increasing the communication costs. By contrast, considering that each query task can be answered within 3 supersteps in DH-Index, and only involves a single round of exchanging messages, the communication cost of DH-Index will not fluctuate significantly when $dist(s, t)$ increases. In addition, the communication cost of



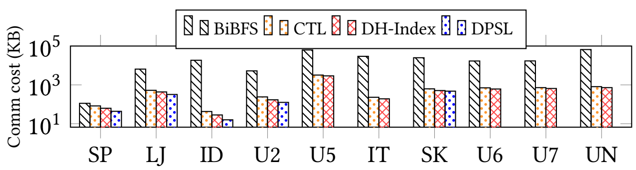
(b) the queries $q(s,t)$ with $dist(s,t) \leq 6$



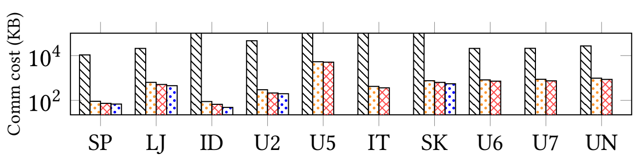
(b) the queries $q(s,t)$ with $dist(s,t) > 6$

Figure 5: Query time (ms) on all datasets

DH-Index is slightly higher than that of DPSL on some datasets, since the DPSL-based queries can be answered within 2 supersteps.



(a) the queries $q(s,t)$ with $dist(s,t) \leq 6$

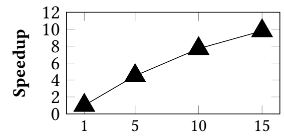


(a) the queries $q(s,t)$ with $dist(s,t) > 6$

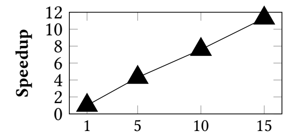
Figure 6: Communication cost (KB) on all datasets

Exp-5: Indexing speedup with multi-cores. We evaluate DH-Index's scalability by varying the number of cores and calculating the indexing speedup. The speedup on x cores is determined using $speedup = T_1/T_x$, where T_1 is the indexing time with a single core on each machine, and T_x is the time with x cores on each machine. This assessment helps us understand how well DH-Index takes advantage of multicore parallelism.

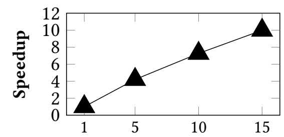
The indexing time speedup of DH-Index is shown in Fig. 7 where the core number in each machine increases from 1, 5, 10 to 15 on four graphs, which means that the total number of cores across different machines increases from 10, 50, 100, to 150. We observe that a near linear speedup has been achieved for DH-Index along with the increasing number of cores. The speedup of each approach is relatively stable over different graphs. Specifically, on 15 cores of each machine, DH-Index shows an average speedup of 10 and a maximum speedup of 11.2. This scalability is mainly due to three reasons. First, the construction of interior index DH_I can be accelerated by multi-core computing in all machines. Second, the eliminated redundant edges in the boundary graph help to reduce the time costs of boundary graph reconstruction and boundary index construction. Third, the construction of DH_B can also be accelerated by multi-core computing in all machines. The above three steps can effectively improve the utilization of cores.



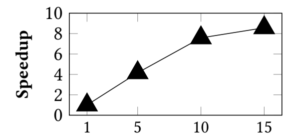
(a) ID



(b) U2



(c) IT



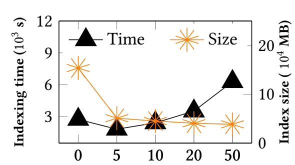
(d) SK

Figure 7: Speedup when varying the cores of each machine

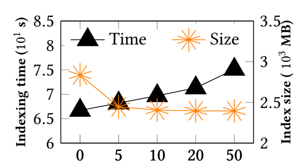
Exp-6: Effect of parameter c. We examine how varying the parameter c (the percent of duplicated boundary vertices) from 0 to 0.05 influences the indexing time and index size of DH-Index. Fig. 8 presents the experimental results in SP, ID, U2, and IT. Similar trends can also be observed on other datasets, which are omitted here due to page limit.

The indexing time of DH-Index tends to rise while its index size generally decreases as c increases, as evidenced by the trends in Fig. 8. Specifically, the indexing time is influenced by two conflicting factors. On one hand, the inclusion of more duplicated higher-order vertices increases the computational load in each machine, resulting in a longer indexing time. On the other hand, these additional vertices help prune redundant label entries, ultimately streamlining the process. This is evident on the SP dataset, where the indexing time decreases initially when c increases from 0 to 0.005, due to the dominant effect of reduced redundant label computations.

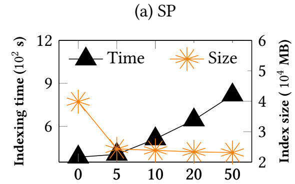
In terms of index size, since the duplicated vertices are always equipped with higher ranking values, the label entries originating from these vertices can largely reduce the storage of redundant labels which can be dominated by the duplicated vertices, thus reducing the index size. As c increases, the index size tends to stabilize since a small number of higher-order vertices are sufficient to prune most of the redundant labels, diminishing the impact of additional duplicated vertices.



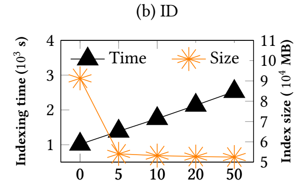
(a) SP



(b) ID



(c) U2



(d) IT

Figure 8: Effectiveness of $c \cdot 10^3$ on performance

Exp-7: Scalability evaluation. In this experiment, we assess DH-Index's scalability in terms of indexing time and index size, adjusting the machine count from 5 to 25. The results on ID and U2 datasets are provided, with consistent patterns observed across other datasets, although these are omitted for brevity. Figs. 9 and 10 display the change in indexing time and index size for DH_I and DH_B, respectively.

As the number of machines rises, we observe a general expansion in the size of boundary vertices, leading to an increased proportion of both indexing time and index size for DH_B in the overall costs. This increment is attributed to each boundary vertex aggregating more label entries during the boundary graph index construction.

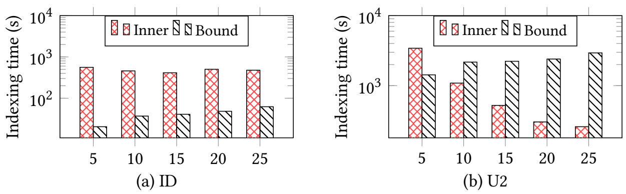


Figure 9: Indexing time when varying the machines

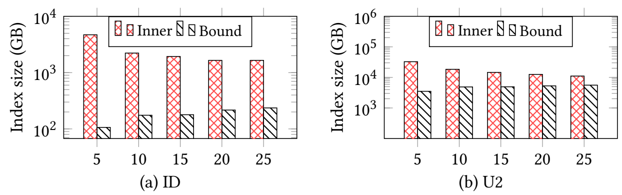


Figure 10: Index size when varying the machines

Exp-8: Effectiveness of the partitioning strategy. This experiment assesses the impact of partitioning strategies on DH-Index, focusing on Hash and KaHIP [31], in terms of indexing time and index size.

Hash-based partitioning, with its poorer locality, results in more boundary vertices and larger boundary index size than KaHIP, as shown in Fig. 11. KaHIP outperforms the Hash-based method, speeding up index construction by up to 17.3× and reducing index size by up to 9.3×. Hash's suboptimal partitions also lead to a significant increase in boundary vertices, causing DH-Index to run out of memory on several datasets.

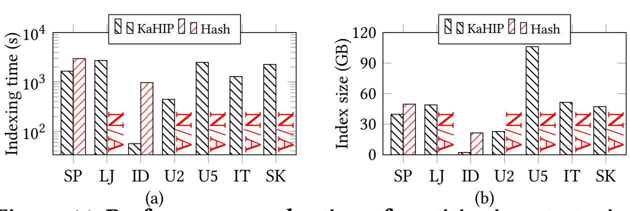


Figure 11: Performance evaluation of partitioning strategies

6 RELATED WORK

Apart from the state-of-the-art methods discussed in Section 2.3, this section explores other related works on shortest distance queries to provide a comprehensive understanding of the field.

6.1 Online Searching Strategy

The distance queries are resolved based on online searching methods, such as Dijkstra, bidirectional searching [7], ALT [12], and neural-optimized A^* [36]. Although there is no extra memory cost, these methods may traverse the whole graph when two query vertices are far from each other, thereby taking huge time costs, especially in large-scale data graphs. In addition, when directly extending these methods to the distributed environment, it is inevitable to produce extra communication costs caused by the message exchange across different machines, thus degenerating the query efficiency. More importantly, these two methods cannot efficiently handle the query tasks where two vertices are not connected in the data graph.

6.2 Index-based Searching Strategy

Index-based methods [3, 10, 15, 19, 22, 23, 33, 42, 44] have been tailored to accommodate graphs with various properties, significantly reducing the search space for each query task through the use of pre-computed distance information.

For the road networks, a contraction hierarchy (CH) method is proposed in [11] to establish the shortcut structures based on a predefined vertex order strategy. Based on the shortcut structure, the query tasks can be efficiently resolved based on a modified bidirectional Dijkstra algorithm. Inspired by CH, Zhu et al. [45] designed an arterial hierarchy (AH) approach that accelerates the construction of shortcuts by exploiting some 2-dimensional spatial properties in the road network. However, the hierarchy-based solutions require a large search space for long-distance queries. The hop-based labeling approaches are proposed in [1, 2] based on the 2-hop index. The work in [22] mitigates short-distance query inefficiencies in road networks through tree decomposition, introducing a hierarchical 2-hop index (H2H-Index) that labels vertices and preserves their hierarchy. The authors in [44] used reinforcement learning to reduce the label size of the index. For small-world networks such as social networks and web graphs, the state-of-the-art methods, including PLL [3], PSL [17], and CTL [18], have been analyzed in Sections 2.2 and 2.3 in detail.

7 CONCLUSION

In this paper, we tackle the shortest-distance query problem in the distributed environment. We introduce **DH-Index**, an innovative index structure ensuring that any pair of connected vertices is at most four hops away via the index. We then present a practical method for efficiently constructing DH-Index in the distributed setting and develop a bidirectional search strategy that leverages DH-Index for query execution. Our extensive experiments validate our approach, highlighting significant improvements in indexing speed, compactness, communication efficiency, and computational scalability. In future work, we will focus on designing effective strategies to solve the shortest distance queries on dynamic graphs.

ACKNOWLEDGMENTS

This work was supported in part by NSFC under Grants 62102341 and 62302421, Basic and Applied Basic Research Fund in Guangdong Province under Grants 2023A1515011280, Guangdong Talent Program under Grant 2021QN02X826, and Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS 20211021111415025. This work was also supported by Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010001).

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *SEA, 2011 (Lecture Notes in Computer Science, Vol. 6630)*. Springer, 230–241.

[2] 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.

[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] Hongzhi Chen, Changji Li, Chenguang Zheng, Chenghuan Huang, Juncheng Fang, James Cheng, and Jie Zhang. 2022. G-Tran: A High Performance Distributed Graph Database with a Decentralized Architecture. *Proc. VLDB Endow.* 15, 11 (2022), 2545–2558.

[5] 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.

[6] 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.

[7] Dennis de Champeaux and Lenie Sint. 1977. An Optimality Theorem for a Bi-Directional Heuristic Search Algorithm. *Comput. J.* 20, 2 (1977), 148–150.

[8] 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.

[9] 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.

[10] 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. *Proc. VLDB Endow.* 6, 6 (2013), 457–468.

[11] 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.

[12] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the shortest path: *A* search meets graph theory. In *SODA, 2005*. SIAM, 156–165.

[13] Kongzhang Hao, Long Yuan, and Wenjie Zhang. 2021. Distributed Hop-Constrained s-t Simple Path Enumeration at Billion Scale. *Proc. VLDB Endow.* 15, 2 (2021), 169–182.

[14] Daoliang He, Pingpeng Yuan, and Hai Jin. 2024. Answering reachability queries with ordered label constraints over labeled graphs. *Frontiers Comput. Sci.* 18, 1 (2024), 181601.

[15] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop Doubling Label Indexing for Point-to-Point Distance Querying on Scale-Free Networks. *Proc. VLDB Endow.* 7, 12 (2014), 1203–1214.

[16] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *Proc. VLDB Endow.* 8, 5 (2015), 509–520.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Efficient Shortest Path Index Maintenance on Dynamic Road Networks with Theoretical Guarantees. *Proc. VLDB Endow.* 13, 5 (2020), 602–615.

[24] Peng Peng, M. Tamer Özsu, Lei Zou, Cen Yan, and Chengjun Liu. 2022. MPC: Minimum Property-Cut RDF Graph Partitioning. In *ICDE, 2022*. IEEE, 192–204.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. 2014. Event detection in activity networks. In *SIGKDD*. ACM, 1176–1185.

[31] Peter Sanders and Christian Schulz. 2013. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7933)*. Springer, 164–175.

[32] 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.

[33] Yang Xiang. 2014. Answering exact distance queries on real-world graphs with bounded performance guarantees. *VLDB J.* 23, 5 (2014), 677–695.

[34] Yehong Xu, Lei Li, Mengxuan Zhang, Zizhuo Xu, and Xiaofang Zhou. 2023. Global Routing Optimization In Road Networks. In *ICDE, 2023*. IEEE, 2524–2537.

[35] Da Yan, James Cheng, Yi Lu, and Wilfred Ng. 2014. Blogel: A Block-Centric Framework for Distributed Computation on Real-World Graphs. *Proc. VLDB Endow.* 7, 14 (2014), 1981–1992.

[36] Lei Yang and Lei Zou. 2021. Noah: Neural-optimized A* Search Algorithm for Graph Edit Distance Computation. In *ICDE, 2021*. IEEE, 576–587.

[37] 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.

[38] 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.

[39] 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.

[40] Junhua Zhang, Wentao Li, Long Yuan, Lu Qin, Ying Zhang, and Lijun Chang. 2022. Shortest-Path Queries on Complex Networks: Experiments, Analyses, and Improvement. *Proc. VLDB Endow.* 15, 11 (2022), 2640–2652.

[41] Junhua Zhang, Long Yuan, Wentao Li, Lu Qin, and Ying Zhang. 2021. Efficient Label-Constrained Shortest Path Queries on Road Networks: A Tree Decomposition Approach. *Proc. VLDB Endow.* 15, 3 (2021), 686–698.

[42] 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.

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

[44] 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 *ICDE, 2023*. IEEE, 1678–1690.

[45] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD, 2013*. ACM, 857–868.