Scaling Up Distance Labeling on Graphs with Core-Periphery Properties

Wentao Li
CAI, FEIT, University of Technology Sydney, Australia
wentao.li@student.uts.edu.au

Miao Qiao
University of Auckland, New Zealand
miao.qiao@auckland.ac.nz

Lu Qin
CAI, FEIT, University of Technology Sydney, Australia
lu.qin@uts.edu.au

Ying Zhang
CAI, FEIT, University of Technology Sydney, Australia
ying.zhang@uts.edu.au

Lijun Chang
The University of Sydney, Australia
lijun.chang@sydney.edu.au

Xuemin Lin
The University of New South Wales, Australia
lxue@cse.unsw.edu.au

ABSTRACT

In indexing a graph for distance queries, distance labeling is a common practice; in particular, 2-hop labeling which guarantees the exactness of the query results is widely adopted. When it comes to a massive real graph with a relatively large treewidth such as social networks and web graphs, however, 2-hop labeling can hardly be constructed due to the oversized index. This paper discloses the theoretical relationships between the graph treewidth and 2-hop labeling's index size and query time. To scale up distance labeling, this paper proposes Core-Tree (CT) Index to facilitate a critical and effective trade-off between the index size and query time. The reduced index size enables CT-Index to handle massive graphs that no existing approaches can process while the cost in the query time is negligible: the query time is below 0.4 milliseconds on all tested graphs including one graph with 5.5 billion edges.

CCS CONCEPTS

• **Information systems** → **Database query processing**.

KEYWORDS

2-hop Labeling; Tree Decomposition; Shortest Distance; Indexing; Algorithm; Treewidth

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389748

ACM Reference Format:
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 *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3389748

1 INTRODUCTION

Given a graph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, a distance query $Q(s, t)$ with query nodes s and t reports the length of the shortest path from s to t. To handle bursty distance queries online, linear-time searching is not efficient enough; this incurs extensive studies on graph indexing with distance labeling. The *distance label* L_v on a node v is a set $L_v \subseteq V$ of nodes; the distance $dist(v, u)$ from v to each node u in L_v is precomputed. With the distance labels, $Q(s, t)$ reports $\min_{v \in L_s \cap L_t} dist(s, v) + dist(v, t)$, the minimum distance between s and t via the label nodes, as a result. This result is an upper bound of $dist(s, t)$ based on triangle inequality. To guarantee an *exact* result, a *2-hop cover* constraint [10] is usually imposed: For any two nodes u and v that are connected by a path, $L_u \cap L_v$ must include at least one node on the shortest path from u to v. Distance labeling that satisfy 2-hop cover constraint are called **2-hop labeling** [2, 19].

Enjoying a *short query delay*, the state-of-the-art 2-hop labeling PLL [2] and its parallelization PSL [17], however, fail on massive real graphs due to the oversized index: the *index size* on UK07 [7] *exceeds 500G*. This raises a natural question: *Is there any 2-hop labeling able to handle these graphs?*

Our theoretical analysis, unfortunately, reveals the inherent limitation of a 2-hop labeling on graphs with a relatively high treewidth. Our theoretical results involves several concepts. We call, for a 2-hop labeling, $|L_v|$ of a node v the *label size* of v. The *maximum label size* over all nodes, denoted as l, captures the query time $O(l)$ and index size $O(nl)$ of

the labeling. This derives the **2-hop complexity** $h(G)$ of a graph G — the *smallest* possible *maximum label size* over all 2-hop labeling on G. On the other hand, a tree decomposition T of graph G is a tree on a set of bags $\{B_1, B_2, \cdots, B_t\}$ that satisfy a set of graph-related constraints (Definition 2). Each bag is a subset of V while the treewidth $tw(T)$ of T is given by the size of the largest bag of T. The **treewidth** $tw(G)$ of G is defined as the **smallest** possible *treewidth* of a tree decomposition of G. The 2-hop complexity $h(G)$ and treewidth $tw(G)$ are related in the following way:

- For a graph G with n nodes and treewidth $tw(G)$, $h(G) = \Omega(tw(G))$ in the worst-case.
- Given a tree decomposition T with $tw(T) = tw(G)$, a worst-case optimal (up to a log factor) 2-hop labeling (maximum label size is $\tilde{O}(tw(G))$) can be constructed.

The relation between $h(G)$ and $tw(G)$ implies that 2-hop labeling may work well on graphs with small treewidth; *when it comes to a graph with a relatively large treewidth, however, the index size $O(ln)$ becomes the bottleneck.*

Existing 2-hop labeling approaches embody the above relationships. The networks with a relatively small treewidth (e.g., road network[1]), can be properly handled (hierarchical 2-hop labeling [19]); on graphs with relatively large treewidth, e.g., social networks and web graphs[2], however, the state-of-the-art 2-hop labeling PSL [17] fails for the oversized index.

The limitation of 2-hop labeling suggests a careful trade-off of the index size and query time. Such a trade-off is enabled with a tree decomposition T (denote by h the height of tree T and $w = tw(T)$, the treewidth of T): distance labeling can be built on the tree while a query can be answered in $O(h)$ hops. The high treewidth issue has been identified [3, 22] and a **core-tree decomposition** is used to partition the graph to a large bag (of nodes), called the core, and a number of small bags with bag size bounded by a parameter d and organized in a forest of height h_F[3]. Table 1 compares these approaches. On large graphs with a relative high treewidth, the approaches that do not consider the high treewidth, [9, 19, 23] and [22] (under $d = w$), suffer the gigantic index size; the ones with special treatment to the core, [3] and [22] ($d < w$), suffer from the quadratic index time in indexing the tree (the small bags) and a long query delay.

This paper proposes a size-tunable distance index CT-Index. With CT-Index, a slight sacrifice of the query time (still controlled under milliseconds which is acceptable by most online systems) leads to a cutting-edge scalability of distance labeling. In comparison with existing approaches [3, 22] that are based on core-tree decomposition, CT-Index carefully couples the index construction of the core and the forest to reduce the index time without affecting the query time (Section 4.4). Our contributions are summarized as follows.

Table 1: Labeling with Tree Decomposition

	# of Hops in Query	Index Size on Tree	Index Time on Tree
[9]	h	$O(nw)$	$O(nm)$
[23]	2	$\tilde{O}(nw)$	$\tilde{O}(nw^2 + mw)$
[19]	2	$O(nw)$	$\tilde{O}(nhw))$
[22]($d = w$)	h	$O(nw^2)$	$O(nm)$
[22]($d < w$),[3]	h_F	$O(nd^2)$	$O(nm)$
CT-Index	4	$O((h_F + d)n)$	$O(d(d + h_F)n)$

- We provide a solid theoretical analysis in showing the limitation of 2-hop labeling.
- We propose a distance index, CT-Index, that can scale up the state-of-the-art 2-hop labeling approach PSL [17] by reducing the index size at a negligible cost in query time. In comparison with existing core-tree-decomposition-based labeling approaches, both the index time and query time are dramatically reduced.
- CT-Index is the only approach that can index massive graphs such as WB, UK0705, and UK07 (see Section 7) for exact distance queries with a short query delay — 0.4 milliseconds over all the graphs including a web graph with 5.5 billion edges.

The paper is organized as follows. Section 2 defines the problem of distance indexing. Section 3 introduces existing solutions. Section 4 shows the limitation of 2-hop labeling and then introduces the structure and query processing of CT-Index. Section 5 constructs the CT-Index. Section 6 introduces related work. Section 7 evaluates CT-Index while Section 8 concludes the paper.

2 PROBLEM STATEMENT

Let G be a graph with $n = |V(G)|$ nodes and $m = |E(G)|$ edges, each edge $e \in E(G)$ has a weight $\delta(e)$, $\delta(e) = 1$ on unweighted graphs. For a node $v \in V(G)$, denote by $N_G(v) = \{u|(v,u) \in E\}$ the set of neighbors of v in G, by $deg_G(v) = |N_G(v)|$ the degree of v. For a set $S \subseteq V$ of nodes, denote by $G[S]$ the induced subgraph of S whose node set is S and edge set is $\{(u,v) \in E(G)|u,v \in S\}$; denote by $clique(S)$ the clique of all nodes in S — the node set is S while the edge set is $\{(u,v)|u,v \in S, u \neq v\}$. We assume that G is undirected while it is easy to extend our techniques to directed graphs.

For two nodes $s, t \in V(G)$, an s-t path p on G is a sequence of vertices $\langle v_0, v_1, v_2, \ldots, v_k\rangle$ where $s = v_0$, $t = v_k$ and its edge set $E(p) = \{(v_{i-1}, v_i)|i \in [k]\} \subseteq E(G)$. The length of p is the joint length over edges on the path, i.e., $\delta(p) = \Sigma_{e \in E(p)}\delta(e)$. The $dist_G(s,t)$ from s to t on graph G is the length of the shortest s-t path, $dist_G(s,t) = \min_{p:\ s\text{-}t \text{ path on graph } G} \delta(p)$.

[1] The treewidth of 9 million-scale road networks is no more than 600.
[2] Web graphs can have cliques with > 3000 nodes [8]; each clique must be entirely included in one bag of any tree decomposition of the graph.
[3] According to our experiment, the average $h_F < 600$ when $d \leq 100$.

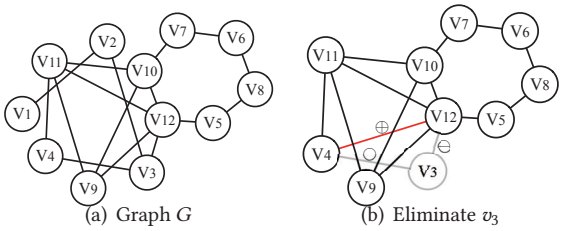

Figure 1: Running Example

Without loss of generality, we assume that G is connected, i.e., an s-t path exists for any two nodes s and t in G. For simplicity, notation G is discarded when the context is clear.

Shortest Distance Indexing Problem. *Efficiently* construct a *compact* index for graph G such that for any two nodes $s, t \in V$, the index can report the shortest distance $dist(s,t)$ between s and t within a *short* delay.

Example 1. *Figure 1(a) shows graph G with 12 nodes and 16 edges. The 4 neighbors $N(v_{10}) = \{v_7, v_9, v_{11}, v_{12}\}$ of v_{10} defines $deg(v_{10}) = 4$. Path $p = \langle v_{10}, v_7, v_6 \rangle$ with $\delta(p) = 2$ provides the distance $dist(v_{10}, v_6) = 2$ between v_{10} and v_6.*

3 EXISTING SOLUTIONS

This section formalizes two concepts related to distance labeling, 2-hop labeling, and tree decomposition, followed by two 2-hop labeling approaches [2, 19]. They pave the way to our theoretical analysis and our new index, CT-Index.

3.1 2-Hop Labeling

Given a graph G, 2-hop labeling indexes G for an efficient query processing for distance queries. **Index Structure.** For each node $v \in V(G)$, the index

- Includes a label set $L_v \subseteq V(G)$, and
- Records the distances from v to each node in L_v: $\{dist_G(v,u) | u \in L_v\}$.

The label sets must satisfy the 2-hop cover constraint.

Definition 1 (2-hop Cover). *For any $u, v \in V(G)$, there exists $w \in L_v \cap L_u$ such that $dist(u,w)+dist(w,v) = dist(u,v)$, that is, w is on a shortest path from u to v.*

Query Processing. For a query $Q(s,t)$ with $s, t \in V(G)$,

$$dist(s,t) = \min_{v \in L_s \cap L_t} dist(s,v) + dist(v,t).$$

Index Size & Index Time. Denote by $l = \max_{v \in V(G)} |L_v|$ the *maximum label size* of the 2-hop labeling. The query time is $O(l)$ while the index size is $O(nl)$.

2-Hop Complexity. A graph can have an infinite number of 2-hop labelings. The smallest maximum label size

$$h(G) = \min_{\mathcal{L}\text{: a 2-hop labeling of } G} \text{the maximum label size of } \mathcal{L}$$

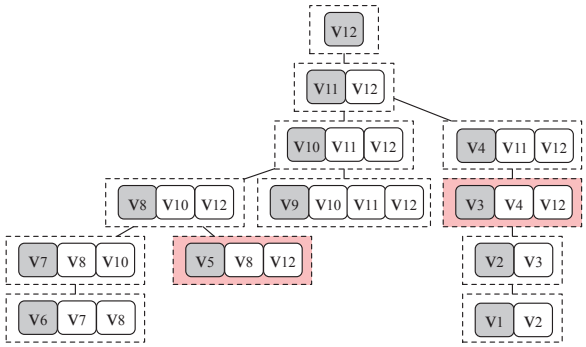

Figure 2: MDE-based Tree Decomposition of G

is called the *2-hop complexity* of G which captures the best complexity that a 2-hop labeling can have on G. This parameter is found to be closely related to another parameter of a graph, the treewidth which is defined in Section 3.2.

3.2 Tree Decomposition and Treewidth

The treewidth of a graph is given by a tree decomposition.

Definition 2 (Tree Decomposition). *A **tree decomposition** T of a graph G is a tree on tree nodes (called* bags*) $\{B_1, B_2, \cdots, B_t\}$ where each bag $B_i \subseteq V$, $i \in [t]$, such that*

(1) The bags jointly cover V, that is, $\bigcup_{i \in [t]} B_i = V$,
*(2) For each $(u,v) \in E$ of G, there is a bag that **covers** the edge, that is, there exists $k \in [t]$ such that $\{u,v\} \subseteq B_k$.*
(3) For any $i, j \in [t]$, $B_i \cap B_j$ is a subset of all the bags on the shortest path from B_i to B_j on T, that is, if two bags have $v \in B_i$ and $v \in B_j$, then $v \in B_k$ for all the bags B_k on the shortest path between B_i and B_j on T.

*The treewidth of T is $tw(T) = \max_{i \in [t]} |B_i| - 1$, the **treewidth** of G is $tw(G) = \min_{T\text{: tree decomposition of } G} tw(T)$.*

Example 2. *Figure 2 shows a tree decomposition T of the graph G in Figure 1(a). The bags on tree T jointly cover nodes in V. All edges are covered, e.g., edge (v_5, v_8) is in the bag $\{v_5, v_8, v_{12}\}$ and edge (v_3, v_4) is in bag $\{v_3, v_4, v_{12}\}$. T satisfies constraint (3), e.g., the two red bags in Figure 2, $\{v_5, v_8, v_{12}\}$ and $\{v_3, v_4, v_{12}\}$, have a common element v_{12}. v_{12} is in all the bags on shortest path between the two red bags. The treewidth of T is $tw(T) = 4$ and $tw(G) \leq 4$.*

A key property of a tree decomposition T of G which is called the cut property (or separator property), underpins the correctness of various labeling methods including ours.

Definition 3 (s-t Separator). *Given two distinct nodes $s, t \in V(G)$ and a set $C \subseteq V(G) \setminus \{s,t\}$, C is an s-t cut if there is no s-t path in the induced subgraph $G[V \setminus C]$ of G on $V \setminus C$. An s-t cut is also called an s-t separator.*

Lemma 1 (Cut/Separator-Property [20]). *Consider two bags B_i and B_j that are adjacent on tree decomposition T. By*

detaching B_i from B_j, tree T is partitioned into two parts. Let C_i be the union of the bags connected to B_i and C_j be the union of bags connected to B_j. $B_i \cap B_j$ is a separator for all s and t pairs with $s \in C_i \setminus (B_i \cap B_j)$ and $t \in C_j \setminus (B_i \cap B_j)$.

It has been proved that computing the treewidth of a graph is NP-Complete [4]. Tree decomposition can be found using heuristics while one of the most effective heuristics is based on minimum degree elimination.

3.2.1 MDE-based tree decomposition. Minimum Degree Elimination (MDE) [5] based tree decomposition eliminates, recursively, the node v in G with the minimum degree and then add the clique of the neighbors of v back to G. Each node v and its neighbors on the transient graph right before the deletion of v form a bag of the tree decomposition.

Minimum Degree Elimination (MDE) [5]. Generate n bags of nodes $\{B_1, B_2, \cdots, B_n\}$ and a *sequence* of nodes $\{v_1, v_2, \cdots, v_n\}$ in n rounds with the starting graph $G_0 = G$. In the i-th round, i takes value from 1 to n:

(1) v_i : the node with the minimum degree (or any one of such nodes if there is a tie situation) in G_{i-1},
(2) N_i : the neighbor set of v_i in G_{i-1},
(3) $B_i : \{v_i\} \cup N_i$,
(4) G_i : a graph that removes v_i from G_{i-1} and then adds $clique(N_i)$, that is, $V(G_i) = V(G_{i-1}) \setminus \{v_i\}$ and $E(G_i) = E(G_{i-1}) \cup E[clique(N_i)] \setminus \{v_i\} \times N_i$.

In the following discussions, the vertex indexes shall align with the sequence generated by MDE by default.

Example 3. *For graph G in Figure 1(a), the MDE completes in 12 steps. Step 1: v_1 which has the minimum degree of $G_0 = G$ is eliminated; the neighbors of v_1 in G_0 is $N_1 = \{v_2\}$, and thus $B_1 = \{v_1, v_2\}$. Remove v_1 and then add the trivial clique of v_1 to form G_1. When v_3 becomes the node with the minimum degree node in G_2, the neighbors of v_3 in G_2 are $N_3 = \{v_4, v_{12}\}$, thus bag $B_3 = \{v_3\} \cup N_3 = \{v_3, v_4, v_{12}\}$. Remove v_3 and add a clique of N_3 to form G_3. Figure 1(b) shows the transformation from G_2 to G_3, that is, remove edges on v_3 and node v_3, then add edge (v_4, v_{12}). It terminates when v_{12} is removed.*

MDE-based tree decomposition [24]. Consider the deliverables of the above MDE process. Construct tree decomposition T^{md} of G on the bags $\{B_1, B_2, \cdots, B_n\}$:

(1) The root of T^{md} is B_n
(2) For each $i \in [n-1]$, denote by $f(i)$ the minimum index (subscription) of the nodes in N_i, let the parent of B_i be $B_{f(i)}$ on T.

Without losing clarity and for simplicity, we also call $v_{f(i)}$ the parent of v_i, for each $i \in [n-1]$.

Due to Property (3) of a tree decomposition (Definition 2), T^{md} can derive a stronger property that has been extensively used in H2H-labeling (that we shall see in Section 3.3) and shall be used in our techniques as well.

Example 4. *Figure 2 shows an MDE-based tree decomposition of G on bags B_1 to B_{12}. Since $\{v_1, v_2, \cdots, v_{12}\}$ is the order of removing nodes from G in the MDE process, it can be verified that for each bag B_i, v_i is the vertex with the minimum index (subscription). Besides, for each B_i, the minimum index $f(i)$ of N_i decides the parent $B_{f(i)}$ of B_i. For example, $B_8 = \{v_8, v_{10}, v_{12}\}$ whose $f(8) = \min\{10, 12\} = 10$. Thus, the parent of B_8 is B_{10} on the tree.*

Lemma 2. *[19] Given an MDE-based tree decomposition T^{md} of graph G whose node order is $v_1, v_2, \cdots, v_n$ and bags are $\{B_1, B_2, \cdots, B_n\}$, for each $i \in [n]$, $v_i \in B_j$ if and only if B_i is an ancestor of B_j, i.e., all nodes in N_i are ancestors of v_i.*

MDE-based treewidth. We call $tw(T^{md})$ the *MDE-based treewidth* of G. This parameter provides an upper bound of $tw(G)$ which will capture the index size of the following two state-of-the-art 2-hop labeling methods: H2H [19] for road network and PLL [2] for scale-free networks.

3.3 Hierarchical 2-Hop Labeling [19]

H2H-Labeling creates distance labelings during the construction of MDE-based tree decomposition T^{md}. Specifically, consider Step (4) in the MDE process (Section 3.2.1), when adding a $clique(N_i)$ to the graph G_i, H2H-Labeling assigns a weight to each clique edge (u, w). The weight is the length of the wedge between u and w via v_i, an upper bound of $dist(u, w)$. If (u, w) already exists in the graph G_{i-1}, H2H-Labeling only keeps the edge with the smaller weight and when an edge is removed, the weight of its incident edges are recorded. The recorded weights are used to construct its index structure.

Index. The H2H-labeling mainly[4] includes the following two arrays for each node $v_i \in V(G)$ and its bag B_i:

(1) Ancestor array stores the ancestors of B_i on T^{md},
(2) Distance array records the distance $dist_G(v_i, v_j)$, for each ancestor B_j of B_i on T^{md}.

Query Processing. Given two nodes v_i and v_j, $dist_G(v_i, v_j)$ can be reported in two cases:

(1) If B_j is an ancestor of B_i (or B_i is an ancestor of B_j): answer with the distance array of v_i (or v_j) directly;
(2) Otherwise, let B_k be the lowest common ancestor of B_i and B_j on T, report $\min_{v_l \in B_k} dist_G(v_i, v_l) + dist_G(v_j, v_l)$.

Lemma 1 ensures the correctness of the query processing: B_k is a v_i-v_j separator on G. Lemma 2 shows that the distance from v_i (or v_j) to v_l is already in the distance array of v_i (or v_j): $v_l \in B_k$ means either $v_l = v_k$ or v_l is an ancestor of v_k, and thus, v_l is a common ancestor of v_i and v_j.

[4]There is an additional position array whose aim is to reduce the query time by a factor of $\log n$ – we won't cover this part in the paper.

Remarks: The index size of H2H-Labeling is $O(nh)$ where h is the height of T^{md}. h is larger than $tw(T^{md})$ since the nodes in N_i are all ancestors of v_i (Lemma 2). *Thus, the index size is no less than* $\Omega(n \times tw(T^{md}))$.

3.4 Pruned Landmark Labeling [2]

Without resorting to tree decomposition, pruned landmark labeling (PLL) constructs 2-hop labeling by

(1) Firstly defining a node order $\{u_1, u_2, \cdots, u_n\}$, then
(2) Performing pruned breadth-first-search (BFS) sourced from each node u_i sequentially.

In the BFS from u_i, when a node u_j is reached while the existing labels can already answer $dist_G(u_i, u_j)$ successfully, the BFS will prune the expansion branch from v_j (the pruned-BFS); otherwise, u_i will be added to the label set of L_{u_j} if u_i has a higher rank than u_j, that is, $i > j$.

Remarks. The size of PLL index is largely dependent on the node order. With the tree decomposition T^{md} of G, one can generate a node order such that the index size of PLL is bounded by $O(n \log n \times tw(T^{md}))$ while the maximum label size is $\log n \times tw(T^{md})$ (Theorem 4.4 [2]).

4 CORE TREE INDEX

The state-of-the-art 2-hop labeling methods introduced in Section 3.1-3.4 have their index-size related to the term of $n \times tw(T^{md})$ — the lower bound of H2H and the upper bound of PLL. *This is not a coincidence.* Section 4.1 demonstrates that the smallest index size over all 2-hop labeling methods is $\Theta(n \times tw(G))$ up to a factor of $\log n$ in the worst-case. This suggests that 2-hop labeling has an *intrinsic bottleneck* on the index size on big graphs with considerable treewidth.

4.1 Limitation of 2-Hop Labeling

We first construct a graph with treewidth d such that any 2-hop labeling must have an index size no less than $\Omega(nd)$.

Lemma 3. *A graph with n nodes and treewidth d has index size of $\Omega(nd)$ in the worst-case.*

Proof. Without loss of generality, assume that n is dividable by d while d is even, that is, there is an integer k such that $n = kd$ while $d/2$ is an integer. We create G as a sequence of "rolling" cliques of size d (Figure 3). Specifically, group the n nodes in $2k$ disjoint groups $C_0, C_1, \cdots, C_{2k-1}$, each having $d/2$ nodes. Let the edge set $E(G)$ include, for i taking values from 0 to $2k-1$, the edges in $clique(C_i \cup C_{i+1 \bmod 2k})$. Since G includes a clique of size d, $tw(G) \geq d-1$. Besides, because the graph has $n(\frac{3}{2}d - 1)$ edges, there are at least $\Omega(nd)$ labels in the index to satisfy the 2-hop cover constraint. Therefore, the index size is $\Omega(nd)$ in the worst-case. □

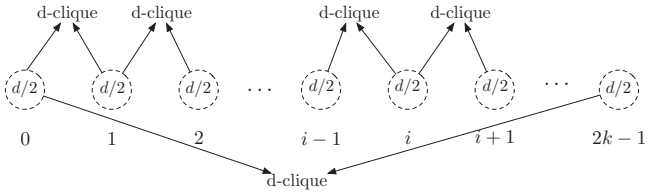

Figure 3: The Proof of Lemma 3

Lemma 4. *[2] Given a tree decomposition T with treewidth d, there is a 2-hop labeling of size $O(nd \log n)$.*

Theorem 1. *Given the number n of nodes of a graph and its treewidth d, the 2-hop complexity $h(G)$ of 2-hop labeling is $\Theta(d)$ up to a factor of $\log n$ in the worst-case.*

Remarks. Theorem 1 is negative to the attempt of scaling 2-hop labeling to graphs such as social networks and web graphs with considerable treewidth.

4.2 Core-Periphery Property

The core-periphery structure has been identified in various community-based graphs such as social networks and web graphs. In general, this property shows that these graphs have a densely connected "core" and by removing this "core", the other nodes, called the "periphery", are of limited connectivity. This structure can potentially reduce the size of distance labeling: building a 2-hop labeling, e.g., PLL, on the core and "tree-like" index, e.g., tree-decomposition based index, on the periphery, can reduce the factor of n in the term $n \times tw(G)$ of the index size. In this sense, a core of size considerably smaller than the graph size would be preferred.

The "core" can be defined in different ways. Leskovec et al. [16] define the "core" as the maximum 2-edge-connected component, which, however, tends to produce a large core that contains 60% of the graph nodes [16], not ideal for reducing the index size of the distance labeling.

Our paper follows the core [18] defined on the MDE-based tree decomposition which partitions a graph G into a core and a set of "trees", called core-tree decomposition.

4.3 Core-Tree Decomposition [18]

Recall the MDE-based tree decomposition with n bags $\{B_1, B_2, \cdots, B_n\}$ of nodes and a tree T^{md} on the n bags. Given a parameter d, the core-tree decomposition divides the tree decomposition T^{md} into a giant core and a forest.

In T^{md} (Figure 4), given a parameter d, the **boundary** λ with $|B_\lambda| \geq d+1$ separates the core $B^c = \bigcup_{i \in [\lambda+1, n]} B_i$ from the remainder in which all the bags have their sizes bounded by $d+1$. Each bag B_i, $i \leq \lambda$, has a **root** B_{r_i} — the highest ancestor of B_i that is not in the core. The index $r(i)$ of the root node $B_{r(i)}$ of B_i is included in the **root set** R. The **interface**

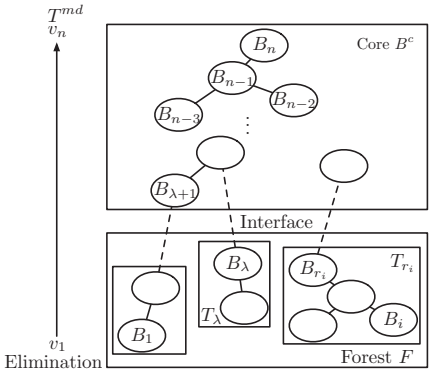

Figure 4: Core Tree Decomposition

of B_i includes the nodes in $N_{r_i} = \{B_{r_i} \setminus r_i\}$ of the root of B_i, that are, the nodes shared by root $B_{r(i)}$ and the father of $B_{r(i)}$ in the core. The interface of any bag has no more than d nodes, and thus, d is called the **bandwidth**. Formally,

- **Bandwidth** d: a user-defined parameter of the core-tree decomposition, a non-negative integer.
- **Boundary** λ: The first λ bags each have at most $d+1$ nodes while $|B_{\lambda+1}| > d+1$. That is, B_λ locates right before the first bag with width greater than d.
- **Core** B^c: the union $B^c = \bigcup_{i\in[\lambda+1,n]} B_i$.
- **Root set** $R = \{i | i \le \lambda, f(i) > \lambda\}$: ids of the non-core bags whose fathers are in the core.
- **Tree** $T_i, i \in [n]$: the subtree of T^{md} rooted at B_i.
- The **root** function $r(i)$ is defined for each $i \le \lambda$: $r(i) = \max\{j \le \lambda | B_j \text{ is an ancestor of } B_i \text{ on } T^{md}\}$.
- **Forest** $F = \{T_i | i \in R\}$: the subtrees with root ids.
- The **interface** of tree T_i, for each $i \in R$: the neighbor set N_i of v_i, in Line (2), the MDE (Section 3.2.1).

Example 5. *For the tree decomposition T^{md} in Figure 2, if the **bandwidth** $d = 2$, then the **boundary** $\lambda = 8$ since $|B_9| = 4 > 3 = d+1$ while all the bags before B_9 had no more than 3 nodes. The **core** $B^c = B_9 \cup B_{10} \cup B_{11} \cup B_{12} = \{v_9, v_{10}, v_{11}, v_{12}\}$. The **roots** $R = \{4, 8\}$. A **tree** T_8 is the subtree of T^{md} rooted at B_8, which contains $\{B_5, B_6, B_7, B_8\}$. The **forest** F contains the subtree with root ids in Roots, i.e., $F = \{T_4, T_8\}$. The **interface** of all the bags in T_8 is the neighbor set $N_8 = \{v_{10}, v_{12}\}$ whose size is no more than d.*

4.4 CT-Index Structure

This section introduces the structure of the index followed by the query processing. CT-Index includes two parts, the **core-index** and the **tree-index**. Two concepts, k-local-path and k-local-distance, that are essential to the reduction of the query delay and index time are defined below.

Definition 4 (k-Local-Path). *Given two nodes s and t in graph G, a path is a k-local-path if all the intermediate nodes on the path are from $\{v_1, v_2, \cdots, v_k\}$.*

Definition 5 (k-Local-Distance). *Given two nodes s and t in graph G, the k-local-distance is the minimum length over all the k-local-paths from s to t.*

By default, $k = \lambda$, unless k is specified in the context. The λ-local-distance between s and t is denoted as $\delta^T(s,t)$.

Example 6. *Consider the tree decomposition T^{md} (Figure 2). Given bandwidth $d = 2$, then $\lambda = 8$ (Example 5). The path $p_1 = \langle v_7, v_6, v_8, v_5, v_{12}\rangle$ from v_7 to v_{12} is the 8-local-path because all the **intermediate** nodes on the path p_1, v_6, v_8 and v_5, are from $S = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. The path $p_2 = \langle v_7, v_{10}, v_{12}\rangle$ is not a 8-local-path because there is an intermediate node v_{10} not in S. Since p_1 is the path with the minimum length in all 8-local-paths from v_7 to v_{12}, $\delta^T(v_7, v_{12}) = |p_1| = 4$.*

Core-index. A 2-hop labeling on the core B^c – not in graph G but in an intermediate graph $G_{\lambda+1}$ generated in MDE process (Section 3.2.1). Note that in our discussion, $G_{\lambda+1}$ is a weighted graph: we associate each edge $(u,v) \in E(G_{\lambda+1})$ a weight of the local distance $\delta^T(u,v)$. We adopt PLL (Section 3.4) for the core-index. Therefore, the core-intex contains, for each vertex $v \in V(G_{\lambda+1})$, a label set $L_v \subseteq V(G_{\lambda+1})$ and the distance from v to each node in L_v : $\{dist_{G_{\lambda+1}}(u,v) | u \in L_v\}$ in $G_{\lambda+1}$. Due to the correctness of PLL, for any two nodes $s, t \in V(G_{\lambda+1})$, we have

$$dist_{G_{\lambda+1}}(s,t) = \min_{v \in L_s \cap L_t} dist_{G_{\lambda+1}}(s,v) + dist_{G_{\lambda+1}}(v,t).$$

Tree-index. For each node v_i with $i \le \lambda$, locate the tree $T_{r(i)}$ of v_i in the forest F. The index on v_i includes:

- The local distance $\delta^T(v_i, v_j)$ for each v_j that is an ancestor of v_i on $T_{r(i)}$, and
- The local distance $\delta^T(v_i, u)$ for each u in $N_{r(i)}$, the interface of the tree $T_{r(i)}$.

Index size analysis. Since the core diminishes under $d \ge tw(T^{md})$, we assume that $d < tw(T^{md})$.

Lemma 5. *The index size of the core-index under $d < tw(T^{md})$ is $size_{core} = O(|B^c| \log(|B^c|) \times tw(T^{md}))$.*

Proof. Since the first λ bags in T^{md}, the MDE-based tree decomposition of G, are of sizes no more than $d+1$ and $d < tw(T^{md})$, the MDE-based treewidth of $G_{\lambda+1}$ is exactly $tw(T^{md})$. Denote by $n_c = |B^c|$, the number of nodes in $G_{\lambda+1}$. The index size of PLL on $G_{\lambda+1}$ is $O(n_c \log n_c \times$ (the MDE-based tree width of $G_{\lambda+1}$)), according to Theorem 4.4 [2], this is exactly $O(n_c \log n_c tw(T^{md}))$. □

Lemma 6. *Tree-index has $size_{tree} = O((h_F + d)(n - |B^c|))$ where h_F is the maximum height of the trees in forest F.*

Proof. For each node in the forest, the tree-index includes its distances to its h_F ancestors and d interface nodes. □

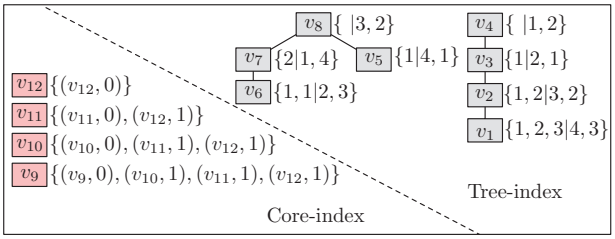

Figure 5: The CT-Index

Theorem 2. *The index size of CT-Index is*

$$O((h_F + d)(n - |B^c|) + |B^c| \log(|B^c|) \times tw(T^{md})).$$

In increasing d, h increases and $|B^c|$ decreases. In other words, in increasing d, $size_{tree}$ increases while $size_{core}$ decreases. As suggested by empirical study [18], as the increase of d, $G_{\lambda+1}$ becomes more and more like a dense random graph, that is, after a certain time, most of the bags that shall be generated in the MDE-process will have sizes similar to $tw(T^{md})$. In such a plateau stage in the elimination of the nodes, $size_{core}$ decreases very slowly while $size_{tree}$ keeps increasing with both h and $n - |B^c|$ and thus $h_F(n - |B^c|)$ in a faster pace. This analysis on one way shows in more detail why H2H-labeling does not work for core-periphery graphs; on the other way suggests that the bandwidth d should be adjusted to effectively tune the index size.

Example 7. *Figure 5 shows the core-index and tree-index under $d = 2$ and $\lambda = 8$. In the core-index, each node in $B^c = \{v_9, v_{10}, v_{11}, v_{12}\}$ bears the 2-hop labeling in graph $G_{\lambda+1} = G_9$. In the tree-index, each node contains its λ-local-distance to two types of nodes: its ancestors in its tree in F and its interface nodes. For example, v_5 has only one ancestor $\{v_8 : 1\}$ and two interface nodes $\{v_{10} : 4, v_{12} : 1\}$ (Figure 2).*

4.5 CT-Index Query Processing

CT-Index answers a query $Q(s, t)$ in 4 cases.

Case 1: Two query nodes are in the core, i.e., $s, t \in B^c$.

- **Report** $dist_{G_{\lambda+1}}(s, t)$ with the core-index.
- **Complexity**: 1 query to the core-index.

Lemma 7. *For $s, t \in B^c$, $dist_{G_{\lambda+1}}(s, t) = dist_G(s, t)$.*

Proof. The length of an edge (u, v) in $E(G_{\lambda+1}) \setminus E(G)$ is an upper bound of $dist_{G_{\lambda+1}}(s, t) \geq dist_G(s, t)$. It thus remains to prove that $dist_{G_{\lambda+1}}(s, t) \leq dist_G(s, t)$. Let p be the shortest path from s to t on G. If all the nodes on p are in B^c then the lemma is proved; otherwise, recursively find the node v_i on p with the minimum index, remove v_i and concatenate the predecessor pre of v_i and the successor suc of v_i with an edge whose length is the summation of the distances from pre to suc via v_i (thus $\delta(p)$ remains unchanged) until all the nodes on p are in B^c. The revised path p is a path on $G_{\lambda+1}$ whose length is $\delta(p) = dist_G(s, t)$. □

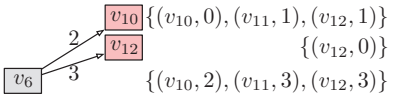

Figure 6: The Extended Label Set

Example 8. *In Figure 5, $s = v_{11}, t = v_{12}$ have $dist(v_{11}, v_{12}) = 1$ from a query on the core-index: $L_{v_{11}} = \{v_{11} : 0, v_{12} : 1\}, L_{v_{12}} = \{v_{12} : 0\}$, the distance is 1 (via v_{12}).*

Case 2: Only one query node is in the core, i.e., $s = v_i$, $i \leq \lambda$ while $t \in B^c$.

- Let $r = r(i)$, the id of B_i's root.
- **Report** $\min_{u \in N_r} \delta^T(v_i, u) + dist_{G_{\lambda+1}}(u, t)$.
- **Complexity**: $O(d)$ queries to the core-index.

A node $u \in N_r$ has its distance $\delta^T(v_i, u)$ in the tree-index and $dist_{G_{\lambda+1}}(u, t)$ in the core-index.

Lemma 8. *For v_i with $i \leq \lambda$ while $t \in B^c$, $dist_G(v_i, t) = \min_{u \in N_r} \delta^T(v_i, u) + dist_{G_{\lambda+1}}(u, t)$.*

Proof. Let p be the shortest path from v_i to t on G. Let B_r be the root of the tree of v_i. Consider the edge from B_r to its parent $B_{f(r)}$. If $t \notin N_r$, according to Lemma 1, $N_r = B_r \cap B_{f(r)}$ is a separator of v_i and t. Therefore, $V(p) \cap N_r \neq \emptyset$; let u be the first such node on p from v_i. Since N_r is a separator between v_i and any node in $B^c \setminus N_r$, p has no node in B^c before u, that is, the segment of p from v_i to u has a length of $\delta^T(v_i, u)$. The length of p is thus $\delta^T(v_i, u) + dist_G(u, t)$. According to Lemma 7, $dist_G(u, t) = dist_{G_{\lambda+1}}(u, t)$. □

Example 9. *In Figure 5, when $s = v_6 \notin B^c$ and $t = v_{11} \in B^c$, the root of B_6 is $r = r(6) = 8$. Enumerate nodes in $N_8 = \{v_{10}, v_{12}\}$ of v_8, and report the minimum value 3 from $\{\delta^T(v_6, v_{10}) + dist_{G_9}(v_{10}, v_{11}), \delta^T(v_6, v_{12}) + dist_{G_9}(v_{12}, v_{11})\} = \{2 + 1, 3 + 1\}$ as the result.*

The following extension operation will be used as a building block in optimizing the query time of Case 3-4.

Extension. For a node v_i with $i \leq \lambda$, its extended label set $L^{ext}_{v_i}$ with distances is constructed as below.

- Let $r = r(i)$ be the id of the root of B_i.
- Let **extended label set** $L^{ext}_{v_i} = \bigcup_{u \in N_r} L_u$ be the union of the core-index label-set of the interface nodes in T_r, and the **extended distances** for each $v \in L^{ext}_{v_i}$ be $dist^{ext}(v_i, v) = \min_{u \in N_r \ s.t. \ v \in L_u} \{\delta^T(v_i, u) + dist_{G_{\lambda+1}}(u, v)\}$.
- **Complexity.** $O(d)$ queries to the core-index.

Example 10. *In Figure 6, for v_6, interface $N_{r(6)} = N_8 = \{v_{10}, v_{12}\}$. $L^{ext}_{v_6} = \bigcup_{u \in N_{r(6)}} L_u = \{v_{10}, v_{11}, v_{12}\}$. The extended distances $dist^{ext}(v_6, v_{10}) = \min\{2 + 0\} = 2$, $dist^{ext}(v_6, v_{11}) = \min\{2 + 1\} = 3$, $dist^{ext}(v_6, v_{12}) = \min\{2 + 1, 3 + 0\} = 3$.*

Case 3-4 of CT-Index shall evaluate 4-hop distances (Equation 1), Lemma 9 is thus important for the query optimization.

Lemma 9. *For any two nodes v_i and v_j with $i, j \leq \lambda$,*

$$\min_{u \in N_{r(i)}, w \in N_{r(j)}} \delta^T(v_i, u) + dist_{G_{\lambda+1}}(u, w) + \delta^T(w, v_j) \quad (1)$$

$$= \min_{v \in L_{v_i}^{ext} \cap L_{v_j}^{ext}} dist^{ext}(v_i, v) + dist^{ext}(v, v_j). \quad (2)$$

Proof. PLL guarantees that for $u, w \in B^c$,

$$dist_{G_{\lambda+1}}(u, w) = \min_{v \in L_u \cap L_w} dist_{G_{\lambda+1}}(u, v) + dist_{G_{\lambda+1}}(v, w).$$

Thus, Equation 1= $\min_{u \in N_{r(i)}, w \in N_{r(j)}, v \in L_u \cap L_w} \delta^T(v_i, u)$ + $dist_{G_{\lambda+1}}(u, v) + dist_{G_{\lambda+1}}(v, w) + \delta^T(w, v_j)$. The condition $\{u \in N_{r(i)}, w \in N_{r(j)}$ and $v \in L_u \cap L_w\}$ can be divided into $\{u \in N_{r(i)}, v \in L_{v_i}^{ext}\ s.t.\ v \in L_u\}$ and $\{w \in N_{r(j)}, v \in L_{v_j}^{ext}\ s.t.\ v \in L_w\}$. This, combined with the definitions of $dist^{ext}(v_i, v)$ and $dist^{ext}(v_j, v)$, completes the proof. □

Lemma 9 turns 4-hop to 2-hop (upon the extended label set), reducing the query processing cost by a factor of d. Specifically, Equation 1 enumerates over the Cartesian product of $N_{r_i} \times N_{r_j}$, leading to d^2 queries on the core-index. By constructing the extended label sets of v_i and v_j in $O(d)$ queries on the core-index, Equation 2 can equivalently report the result with a simple intersection whose cost is dominated by the $O(d)$ queries on the core-index.

Case 3: Two query nodes are in different trees in the forest, i.e., $s = v_i, t = v_j$ where $i, j \leq \lambda$ and $r(i) \neq r(j)$.

- Perform the **extension** operations on v_i and v_j.
- **Report** $\min_{\substack{u \in N_{r(i)} \\ w \in N_{r(j)}}} \delta^T(v_i, u) + dist_{G_{\lambda+1}}(u, w) + \delta^T(w, v_j)$.
- **Complexity**: $O(d)$ queries to the core-index (computed based on Lemma 9).

Lemma 10. *The distance reported by Case 3 is $dist_G(s, t)$.*

Proof. Let p be a shortest path from v_i to t. Let u be the first node on the path p in $N_{r(i)}$. Let w be the last node on the path p in $N_{r(j)}$. According to Lemma 1, the segment of p on (v_i, u) has length $\delta^T(v_i, u)$ while that on (w, v_j) has length $\delta^T(w, v_j)$. Therefore, $dist_G(s, t) = \delta^T(v_i, u) + dist_G(u, w) + \delta^T(w, v_j)$ while $dist_G(u, w) = dist_{G_{\lambda+1}}(u, w)$ (Lemma 7), which completes the proof. □

Example 11. *when $s = v_6$ and $t = v_1$. $r(6) = 8 \neq r(7) = 4$. $L_s^{ext} = \{v_{10} : 2, v_{11} : 3, v_{12} : 3\}$ and $L_t^{ext} = \{v_{11} : 4, v_{12} : 3\}$. Report the minimum value 6 from $\min_{v \in L_s^{ext} \cap L_t^{ext}} dist^{ext}(s, v) + dist^{ext}(t, v)$ as the result.*

Case 4: Two query nodes are in the same tree in the forest, i.e., $s = v_i$, $t = v_j$, $i, j \leq \lambda$ and $r(i) = r(j)$.

- B_k: the lowest common ancestor[5] of B_i and B_j on $T_{r(i)}$.
- Perform the **extension** operations to v_i and v_j.
- **Report** $\min\{d_2, d_4\}$ where
  (1) Two hops $d_2 = \min_{u' \in B_k} \delta^T(v_i, u') + \delta^T(v_j, u')$, and
  (2) Four hops $d_4 = \min_{u, w \in N_{r(i)}} \delta^T(v_i, u) + dist_{G_{\lambda+1}}(u, w) + \delta^T(w, v_j)$. Apply Lemma 9 to compute d_4.
- **Complexity**: $O(d)$ queries to the core-index.

Since what we record on the tree-index is the "local" instance, therefore, searching over the s-t separator B_k may not be sufficient in reporting the "global" distance: a path going through B_k and to some nodes in B^c and then back to another node in B_k can be the actual shortest path.

Lemma 11. *The distance reported by Case 4 is $dist_G(s, t)$.*

Proof. Let p be a shortest path from s to t. If p is a local path, d_2 returns the shortest distance. If p is not a local path, let u be the first node on p in B^c and w the last. According to Lemma 10, the shortest distance can be captured by d_4 in concatenating two local paths with a global distances between a pair of nodes in N_r. □

Example 12. *In Figure 5, $s = v_5$, $t = v_6$, $5, 6 \leq 8$ and $r(5) = r(6) = 4$. B_8 is the lowest common ancestor of B_5 and B_6. $d_2 = \min\{\delta^T(s, u') + \delta^T(t, u') | u' \in B_8\} = \min\{1 + 1, 4 + 2, 1 + 3\} = 2$. $d_4 = 4$ since $L^{ext}(s) = \{v_{10} : 4, v_{11} : 5, v_{12} : 1\}$ and $L^{ext}(t) = \{v_{10} : 2, v_{11} : 3, v_{12} : 3\}$. Report $\min\{d_2, d_4\} = 2$.*

Query Complexity. In the 4 cases above, the query complexity is given by $O(d)$ pairwise shortest distance queries on the core-index. The query time can thus be bounded.

Lemma 12. *The query time on the core-index is $O(tw(T^{md}) \log |B^c|)$.*

Proof. Recall that the core-index is a PLL index on $G_{\lambda+1}$ with a particular node order such that the index size is $O(n \log n \times tw(T^{md}))$. Follow the construction (Theorem 4.4 [2]) of the PPL based on T^{md}, the maximum label size is $tw(T^{md}) \log |B^c|$. Therefore, the query time on the core-index is $O(tw(T^{md}) \log |B^c|)$. □

Theorem 3. *The query complexity of CT-Index is $O(d \log |B^c| \times tw(T^{md}))$.*

Remarks. Theorem 2-3 shows that d provides a trade-off mechanism of the index size and query time.

5 CT-INDEX CONSTRUCTION

This section focuses on an effective construction of the CT-Index. We start with an adapted MDE process (Section 3.2.1) which assigns each clique edge a weight before inserting it into the graph and records the length of an edge upon its removal. Then we build up the distance labels, including the core-index and tree-index described in Section 4.4, based on these deliverables of the MDE process. Algorithm 1 describes the construction process.

[5] Answering a lowest common ancestor on a tree can be done in $O(1)$ time with a linear space index on the tree [12].

MDE-based tree decomposition. In Algorithm 1, Line 1-16 is essentially the tree decomposition which terminates whenever the node to be eliminated has d or more than d neighbors (Line 17); right before the ending of the decomposition, we record the boundary and core (Line 7-8). As explained in Section 3.2.1, upon the elimination of a node v_i, we remove all the affiliated edges while reinvesting a clique of the neighbors of v_i into the graph (Line 13-14). Distance information is maintained constantly. Most of the time, the edge weight retains through the elimination; when a clique edge (u, w) is inserted, it will associated with a weight $\delta_i^-(u) + \delta_i^-(w)$, the length of the wedge of the two edges from u and w to v_i — as an upper bound of $dist_G(u, w)$ — to the graph (Line 16). If the graph already has (u, w), then the distance will be updated with the smaller distance among the two without duplicating the edge. This effort is harvested when edge (v_i, u) is deleted upon the deletion of v_i: we write down the edge weight as $\delta_i^-(u)$ (Line 12). As we shall see, this distance will play an important role in index construction.

Next we show an important property of the distance that we have written down, $\delta_i^-(u)$. Recall the definition of a k-local-path, the paths with intermediate nodes in $v_1, \cdots, v_k$, and k-local-distance defined in Definition 4-5.

Lemma 13. *For $i \in [\lambda]$ and $u \in V(G)$, $u \in N_i$ if and only if there is, between v_i and u, an $(i-1)$-local path.*

Proof. Let G^+ be the graph which includes all the edges in G_0 to G_n generated in the MDE process. Edge $(v_i, v_j) \in E(G^+)$ if and only if $(v_i, v_j) \in E(G_{\min\{i,j\}})$. The lemma is proved according to Lemma 2.1 [5]. □

Based on a similar rationale of Floyd-Warshall algorithm in computing the shortest distances, $\delta_i^-(u)$ is the shortest distance between v_i and v over all the $(i-1)$-local-paths.

Lemma 14. *$\delta_i^-(u)$ is the $(i-1)$-local-distance between v_i and u.*

Proof. According to Lemma 13, there is at least one $(i-1)$-local-path from v_i to u. Let p be the shortest $(i-1)$-local-path from v_i to u. Consider p in the MDE process: the intermediate nodes on p will be contracted in the ascending order of their indexes. Specifically, in contracting a node v_j, MDE connects its predecessor $pred(v_j)$ and successor $suc(v_j)$ directly with an edge weighted as the summation of edges from $pred(v_j)$ and $suc(v_j)$ to v_j — such an edge weight will not be updated by other edges since they are already the shortest distances between the corresponding nodes on p. In this sense, the length of p sustains during MDE until it becomes an edge. Therefore, $\delta_i^-(u)$ is the length of p. □

Tree-Index Construction. Lemma 14 shows that for a root v_i and any neighbor $u \in N_i$ of u, the λ-local-distance $\delta^T(v_i, u)$ is exactly $\delta_i^-(u)$ (Line 25). The tree-index on the root is thus built. For a node v_i that is not a root, we compute its λ-local-distances (Line 27-32) with Lemma 15.

Algorithm 1: CT-Index Construction

Input: Graph G, bandwidth parameter d
Output: CT-Index

1 $G_0 \leftarrow G$ with default edge weight $\delta_0(e) = 1, \forall e \in E(G_0)$;
2 $i \leftarrow 1$;
3 **repeat**
4 $v_i \leftarrow$ the node with the minimum degree in G_{i-1};
5 $N_i \leftarrow$ the neighbor set of v_i in G_{i-1};
6 **if** $|N_i| \geq d$ **then**
7 Boundary $\lambda \leftarrow i - 1$;
8 Core $B^c \leftarrow V \setminus \{v_1, \cdots, v_\lambda\}$;
9 **else**
10 Bag $B_i \leftarrow \{v_i\} \cup N_i$;
11 **for** *each* $u \in N_i$ **do**
12 $\delta_i^-(u) \leftarrow$ the edge weight of (u, v_i) in G_{i-1};
13 $V(G_i) \leftarrow V(G_{i-1}) \setminus \{v_i\}$;
14 $E(G_i) \leftarrow E(G_{i-1}) \cup E(clique(N_i)) \setminus (\{v_i\} \times N_i)$;
15 **for** *each* $e(u, w) \in E(G_i)$ **do**
16 $\delta_i(e) = \begin{cases} \delta_{i-1}(e), & \text{if } e \in E(G_{i-1}) \setminus E(clique(N_i)) \\ \delta_i^-(u) + \delta_i^-(w), & \text{if } e \in E(clique(N_i)) \setminus E(G_{i-1}) \\ \min\{\delta_i^-(u) + \delta_i^-(w), \delta_{i-1}(e)\}, & \text{if otherwise} \end{cases}$
17 **until** $|N_i| \geq d$;
18 Roots $R \leftarrow \emptyset$;
19 **for** *each* $i = \lambda$ *downto* 1 **do**
20 Parent $f(i) \leftarrow \min_{v_j \in N_i} j$;
21 **if** $f(i) > \lambda$ *or* $N_i = \emptyset$ **then**
22 Roots $R \leftarrow R \cup \{i\}$;
23 Root function $r(i) \leftarrow i$;
24 **for** *each* $u \in N_i$ **do**
25 Local distance $\delta^T(v_i, u) \leftarrow \delta_i^-(u)$;
26 **else**
27 Root function $r(i) \leftarrow r(f(i))$;
28 $N_i^T \leftarrow N_i \setminus B^c$;
29 **for** *each* $u \in N_i$ **do**
30 $\delta^T(v_i, u) \leftarrow \min\{\delta^-(u), \min_{v_j \in N_i^T} \delta_i^-(v_j) + \delta^T(v_j, u)\}$;
31 **for** *each* $u \notin N_i$ *but in* $ancestor^T(v_i) \cup N_{r(i)}$ **do**
32 $\delta^T(v_i, u) \leftarrow \min_{v_j \in N_i^T} \delta_i^-(v_j) + \delta^T(v_j, u)$;
33 Construct the PLL (or PSL equivalently) index on $G_{\lambda+1}$;

Lemma 15. *Given λ, for each $i \in [\lambda]$ and $r = r(i)$ the root of i, let N_i^T be the nodes in N_i that are ancestors of v_i on tree T_r rooted at r. The following properties hold:*

(1) $N_i^T = N_i \setminus B^c$, and
(2) For each node $u \in V(G) \setminus \{v_1, v_2, \cdots, v_{i-1}\}$ and a λ-local-path p from v_i to u with the shortest length $\delta(p) = \delta^T(v_i, u)$, either $\delta(p) = \delta_i^-(u)$ or there is a node $v_j \in N_i^T$ such that $\delta(p) = \delta_i^-(v_j) + \delta^T(v_j, u)$.

Proof. According to Lemma 2, all the nodes in N_i are ancestors of v_i on T^{md}. Besides, all the ancestors of v_r are in B^c, thus $N_i^T = N_i \setminus B^c$. Let w be the first node on p from v_i such that $w \notin \{v_1, v_2, \cdots, v_{i-1}\}$. w exists since u is such a node. If $w \in B^c$, then according to the definition of λ-local-path, $w = u$. Thus, the length of p is $\delta(p) = \delta^-(u)$. If $w \notin B^c$, then w is a node v_j with $j \leq \lambda$ and $v_j \in N_i$ (Lemma 13). Therefore, v_j is an ancestor of v_i on $T_{r(i)}$. The segment on p from v_i to v_j has length $\delta_i^-(v_j)$ while the rest part of p has length $\delta^T(v_j, u)$, namely, $\delta(p) = \delta_i^-(v_j) + \delta^T(v_j, u)$. □

Note that the loop in Line 19 is in *reverse* order while the ancestors of a node v_i have their indexes larger than i. Therefore, when we use $\delta^T(v_j, u)$ in Line 30 and 32, it has already been correctly computed since $j > i$. Line 19-32 can, thus, construct the CT-Index correctly.

Index Time Complexity. In decomposing the core and the trees, we spend $O(d^2)$ time in eliminating each node, which takes $O(d^2(n - |B^c|))$ time in total. In computing each local distances that are needed for the Tree-Index (Line 30 and 32) we spend time no more than the size of $|N_i^T|$, which takes $O((h_F + d)(n - |B^c|))$ time in total where h_F is the maximum tree height in the forest F.

Theorem 4. *The index time for constructing everything apart from the core-index is* $O(d(d + h_F)(n - |B^c|))$.

Decide the bandwidth d**.** d manages the trade-off between the index size and query time which is essential when PSL fails in constructing the index in exceeding the memory limit. If the memory is large enough, d should be 0 that provides the best query efficiency; otherwise, d should be as small as possible. This initiates a binary search. The upper bound d_{ub} of the binary search takes an experienced value to reduce the search time; double d_{ub} when a feasible d cannot be found.

Remarks. The concepts of local-path and the corresponding local-distance dramatically reduce the index time since computing the "global" distance from each node to its ancestors on the tree incurs $O(d \times tw(T^{md}))$ query cost on the core-index. d is typically set to 100 while $tw(T^{md})$ of a big core-periphery graph is more than 1000. In this sense, CT-Index reduces the index time of "global" construction of the core-index and tree-index by a factor of 10^5.

6 RELATED WORK

2-hop Labeling. 2-hop labeling [10] has been adapted to graphs with different properties. For road networks, planar graph properties have been extensively exploited to reduce the index size and query time. For example, Abraham et al. proposed a contraction hierarchies algorithm and then used hubs to improve efficiency [1]. The state-of-the-art 2-hop labeling on road network [19] is explained in Section 3.1. It uses the small treewidth of a road network to achieve a superior index and query efficiency. These approaches do not apply to graphs without the designated properties.

For small-world networks such as social networks and web graphs, the state-of-the-art approach is PLL [2] (Section 3.4). Due to its immense index size, Jiang et al. [13] designed a disk labeling algorithm that gradually covers all the shortest paths in a graph by hop doubling; to improve its index time, Li et al. PSL [17] parallelized PLL's labeling process a multi-core environment which achieves a near-linear speedup. In this scenario, the index size remains the bottleneck.

Tree-Decomposition. Tree-decomposition was first studied by Halin [11] and later explored by Robertson et al. [20] as part of their graph minor theories. Arnborg et al. [4] proved the NP-Completeness in determining whether the treewidth of a graph is more than a given value. Bodlaender et al. [6] generate a tree decomposition with the minimum treewidth in time exponential to the treewidth. For graphs with treewidth more than a constant, researchers study heuristic-based tree decomposition algorithms.

Xu et al. [24] listed several heuristics for tree decomposition, including the widely adopted [18, 19, 22] Minimum Degree Elimination (MDE) heuristic [5]. On graphs with large treewidth, core-tree decomposition [3, 18, 22] terminates MDE once the minimum degree exceeds a threshold.

Distance Computation with Tree Decomposition. Table 1 compares the labeling approaches using tree decomposition techniques. Wei et al. [22] were the first in this category. For a tree decomposition T of width w and height h, they use the bags along the tree path that connects the two query nodes to perform dynamic programming online, the query time is $O(w^2h)$. Following this work, Chang et al. [9] presented a multi-hop labeling approach whose query time is $O(w(w + h))$; Xiang [23] proposed an approach with index size $O(nw \log n)$. The state-of-the-art approach of this line [19] has an index size of $\Omega(nw)$ (Section 3.3). On real graphs with potentially large treewidth, these approaches entail either excessive query time or index size.

Core-tree decomposition has been used [3, 22] to partition the graph into a core and a forest. It treats the core as a special bag and computes, for each bag, pairwise shortest distance of nodes in the bag, which incurs $O(mn)$ time for indexing. It processes a query based on the stored distances with a special treatment to the nodes in the core. In contrast, CT-Index stores local distance for nodes on the tree while builds a PSL index for nodes in the core to achieve a smaller index size and shorter index time.

7 EXPERIMENTAL RESULTS

Algorithms. The proposed CT-Index is compared with the state-of-the-art 2-hop labeling approach PSL [17] which is a multi-core parallelization of PLL [2]. PSL further provides two index reduction techniques. *Equivalence relation elimination* keeps only one node for those who share the same set of neighbors while *local minimal set elimination* removes (since PSL keeps a global order over all the graph nodes) the label sets of the nodes who bear the minimum order among their neighbors — these label sets are restores in querying time. These two techniques derive two baseline approaches.

- PSL$^+$: PSL with equivalence relation elimination.
- PSL*: PSL with equivalence relation elimination plus local minimal set elimination.

Since the *equivalence relation elimination* shrinks the graph and is not depending on any specific technique, we have integrated it into our proposed CT-Index. Bandwidth d is a parameter in our algorithm, we denote by "CT-d" as our algorithm with a specific bandwidth d. Note that, when d becomes zero, our method CT-0 is the same as PSL$^+$ because the whole graph is treated as a core.

All algorithms were implemented in C++ and compiled with GNU GCC 4.8.5 and -O3 level optimization. All experiments were conducted on a machine with 48 CPU cores and 384 GB main memory running Linux (Red Hat Linux 4.8.5, 64bit). Each CPU core is Intel Xeon 2.1GHz. The parallelized programs are supported by the openMP framework. When a program runs out of memory, we mark the index size, and the corresponding index time and query time as "OM".

Datasets. The experiments were conducted on 30 real graphs (Table 2) including social networks, web graphs, coauthorship graphs, communication networks, and interaction networks that were downloaded from Network Repository[6] [21], Stanford Large Network Dataset Collection[7] [15], Laboratory for Web Algorithms[8] [7], and the Koblenz Network Collection[9] [14]. The largest graph has over 5.5 billion edges.

Exp 1: Index Size. Figure 7 shows the index size of CT-Index with the bandwidth $d = 20$ (CT-20), CT-Index with bandwidth $d = 100$ (CT-100), PSL$^+$ (CT-0) and PSL*.

CT-100 is the only algorithm that completed the index construction on all the graphs. CT-20 failed on UK0705 and UK07 due to the oversized index; on other graphs, the index size of CT-100 is 1.41 times smaller than CT-20 on average. PSL$^+$ failed on 6 out of 30 graphs due to the oversized index. The index size of CT-100 is 4.79 times smaller than PSL$^+$ on average and 23.72 at a maximum (on SINA). PSL* failed on 3 out of 30 graphs due to the oversized index; on other graphs, CT-100 reduces the index size of PSL* by a factor of 2.31 on average and 5.66 at a maximum; this reduction is critical especially when the bottleneck is in the index size.

[6] http://networkrepository.com/index.php
[7] http://snap.stanford.edu/data/
[8] http://law.di.unimi.it
[9] http://konect.uni-koblenz.de/

Table 2: The Description of Dataset

Name	Dataset	n	m	Type
TALK	Wikitalk[7]	2,394,385	5,021,410	Communication
AMAZ	Amazon[8]	735,323	5,158,388	Social Network
YOUT	Youtube[9]	3,223,589	9,375,374	Social Network
EPIN	Epinions[6]	755,762	13,396,320	Social Network
DBPE	Dbpedia[9]	3,966,924	13,820,853	Web Graph
HUDO	Hudong[6]	1,984,485	14,869,484	Web Graph
BAID	Baidu[6]	2,141,301	17,794,839	Web Graph
DBLP	DBLP[9]	1,314,050	18,986,618	Coauthorship
TOP	Topcats[7]	1,791,489	28,511,807	Web Graph
POK	Pokec[7]	1,632,803	30,622,564	Social Network
FLIC	Flickr[9]	2,302,925	33,140,017	Social Network
FRIE	Friendster[6]	8,658,745	55,170,227	Social Network
STAC	Stack[7]	6,024,271	63,497,050	Interaction
LJ	Ljournal[8]	5,363,260	79,023,142	Social Network
FB	Facebook[6]	58,790,783	92,208,195	Social Network
ENWI	Enwiki[8]	5,616,717	128,835,798	Social Network
INDO	Indochina[8]	7,414,866	194,109,311	Web Graph
HOLL	Hollywood[8]	2,180,759	228,985,632	Social Network
SINA	Sinaweibo[6]	58,655,850	261,321,071	Social Network
TWIT	Twitter[6]	21,297,772	265,025,809	Social Network
UK02	UK-2002[8]	18,520,486	298,113,762	Web Graph
WIKI	Wikipedia[9]	12,150,976	378,142,420	Web Graph
ARAB	Arabic[8]	22,744,080	639,999,458	Web Graph
UK05	UK-2005[8]	39,459,925	936,364,282	Web Graph
WB	Webbase[8]	118,142,155	1,019,903,190	Web Graph
IT04	IT-2004[8]	41,291,594	1,150,725,436	Web Graph
SK05	SK-2005[8]	50,636,154	1,949,412,601	Web Graph
UK06	UK-2006[8]	77,741,046	2,965,197,340	Web Graph
UK0705	UK-07-05[8]	105,896,555	3,738,733,648	Web Graph
UK07	UK-2007[8]	133,633,040	5,507,679,822	Web Graph

Exp 2: Index Time. Figure 8 shows the index time for the algorithms, CT-20, CT-100, PSL$^+$, and PSL*, ran on 45 threads.

CT-100 has a slightly (4% on average) longer index time than CT-20. CT-100 speeds up PSL$^+$ by a factor of 3.26 on average (on the graphs where PSL$^+$ can finish the labeling) and 21.85 at a maximum (on SINA), and speeds up PSL* by a factor of 1.68 on average and 4.64 at a maximum (on UK02).

Exp 3: Query Time. Figure 9 shows the average query time of CT-20, CT-100, PSL$^+$, and PSL* on 10^6 random queries.

The query time of CT-100 is comparable to the other approaches. CT-100 has query time slightly longer than (2.37× on average and 4.48× at the maximum) CT-20. CT-100 is on average 7.55 times slower than PSL$^+$ and on average 3.17 times slower than PSL*. Even on the largest graph UK07, the average query time of CT-100 is 0.39 millisecond while all other methods failed to complete the index construction.

Exp 4: The Effect of bandwidth d. Figure 10(a) shows the index size of CT-Index under bandwidth 0, 2, 5, 10, 20, 50, and 100 on 6 data graphs. The index size drops with an increasing d. On UK05, when d is 0 or 2, the index cannot be constructed

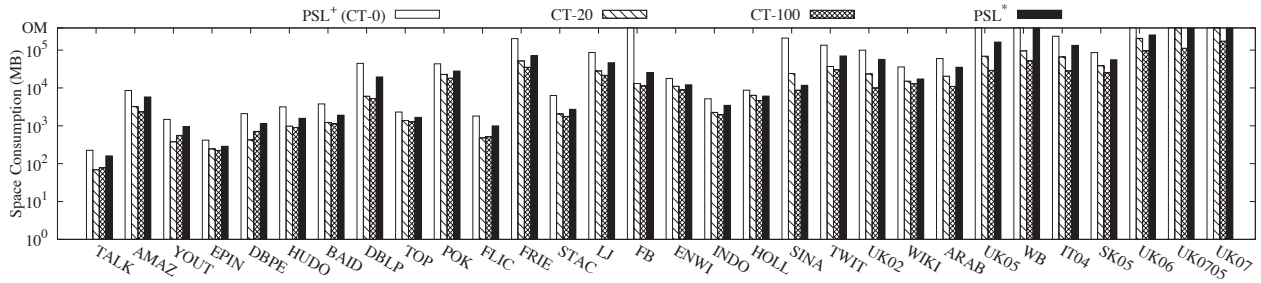

Figure 7: The Comparison of the Index Size

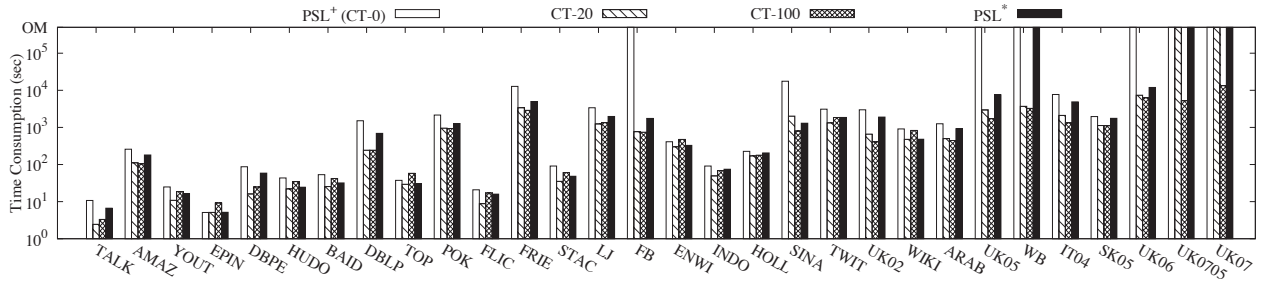

Figure 8: The Comparison of the Index Time

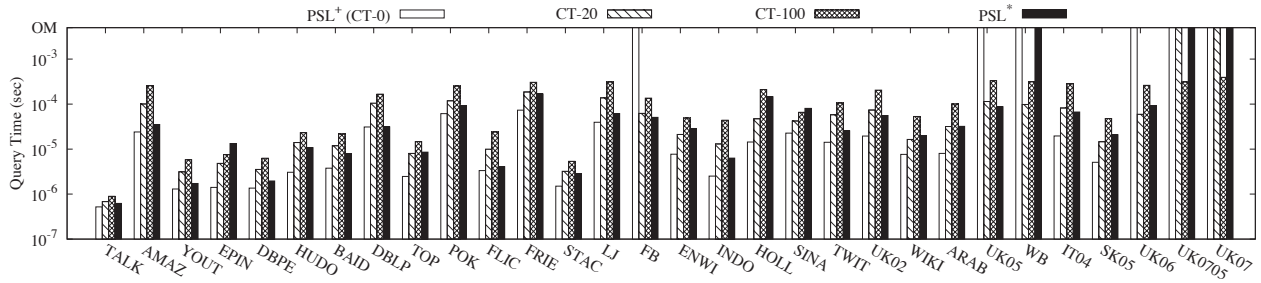

Figure 9: The Comparison of the Query Time

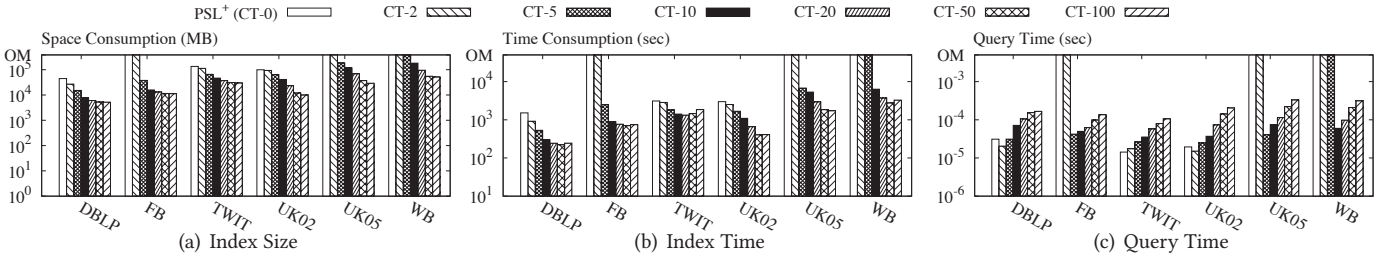

Figure 10: The Effect of bandwidth d

due to the oversized index. When $d = 5$, the index size is 182.94GB which decreases to 117.94GB when $d = 10$, 69.02 under 20, 37.09 under 50, and 29.11 under 100. The marginal gain (index size reduction) decreases in increasing d and becomes very small when $d = 100$.

Figure 10(b) presents the index time of CT-Index under a varying d. With the increase of bandwidth d, the index time normally (on graphs such as DBLP, FB, UK02, and UK05) decreases while showing some fluctuates on TWIT and WB. This can be explained by the trade-off between the core-tree decomposition time and the core index construction time.

Figure 10(b) shows the average query time of CT-Index under a varying d. The query time of CT-Index mildly increases when d is increasing but even when $d = 100$, the query time is constantly below 0.4ms, 8 times slower than PSL* (see Figure 9). This is cost-effective for the improved scalability: PSL* cannot be constructed on WB.

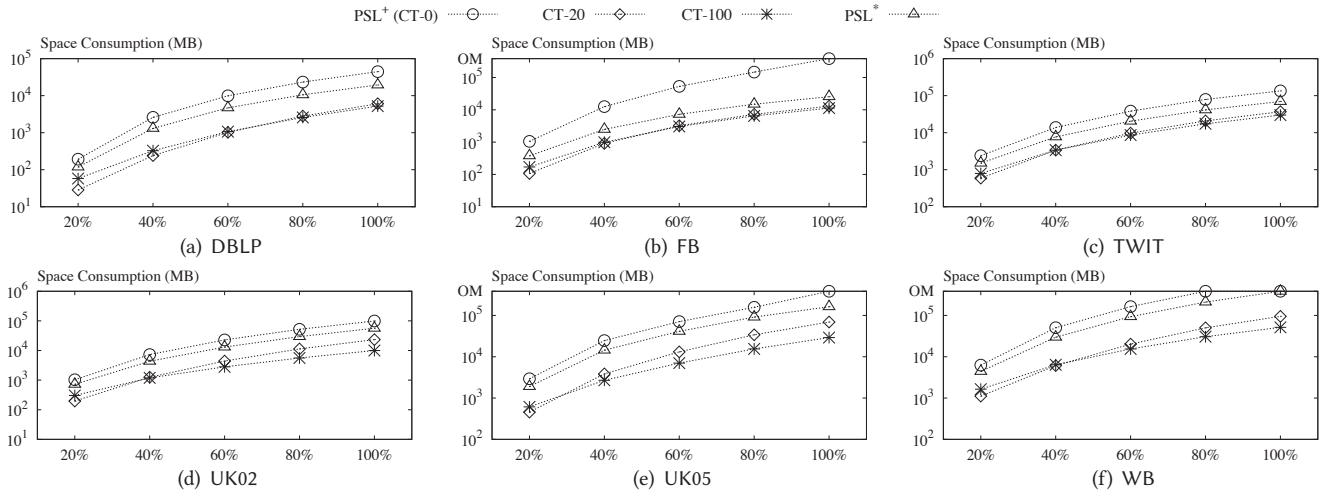

Figure 11: The Test of Scalability for the Index Size

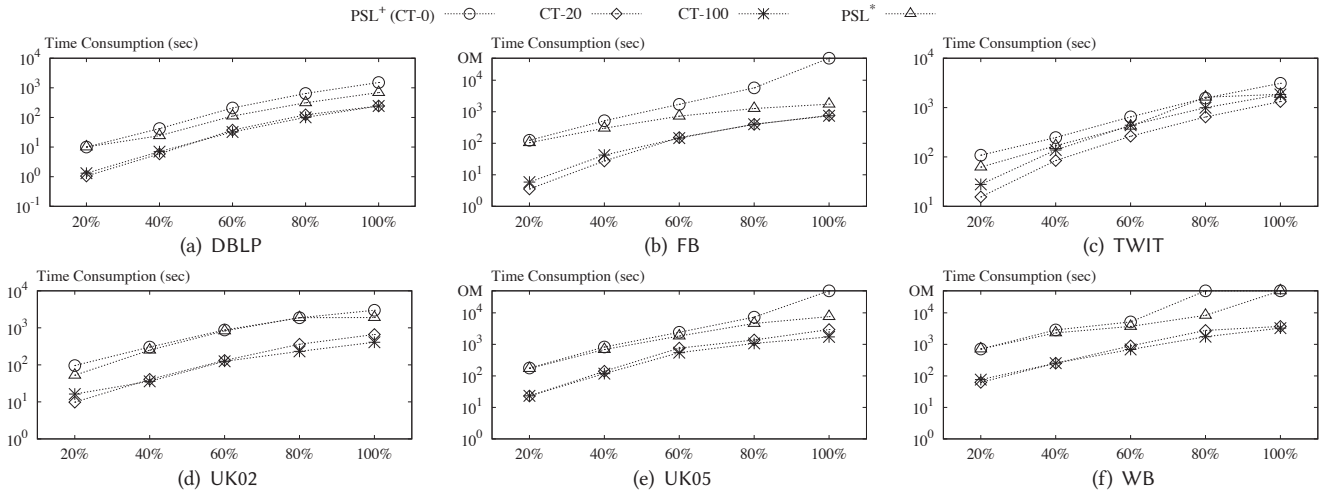

Figure 12: The Test of Scalability for the Index Time

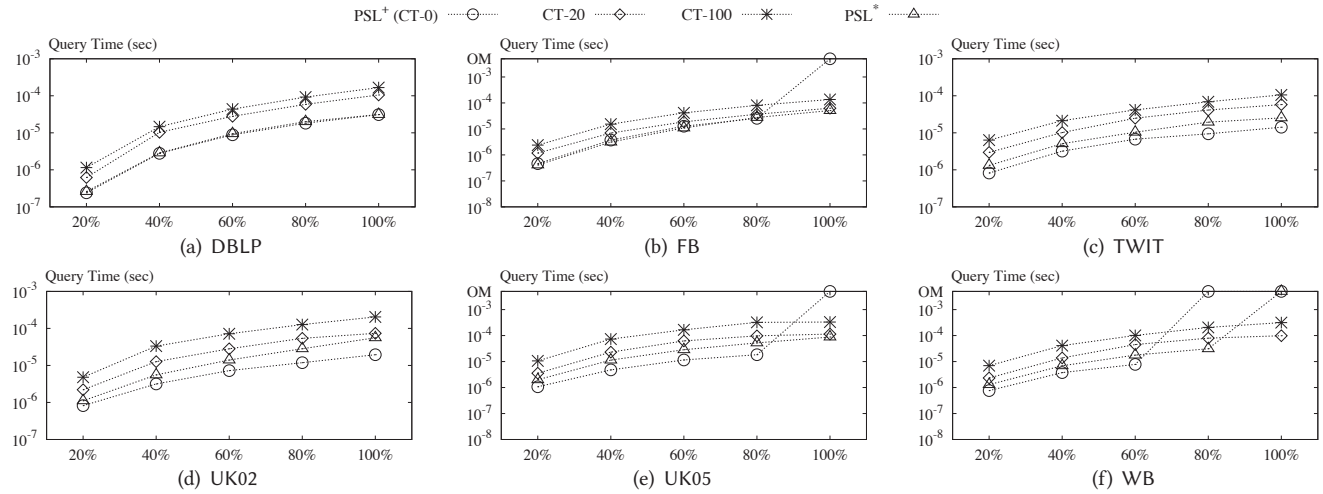

Figure 13: The Test of Scalability for the Query Time

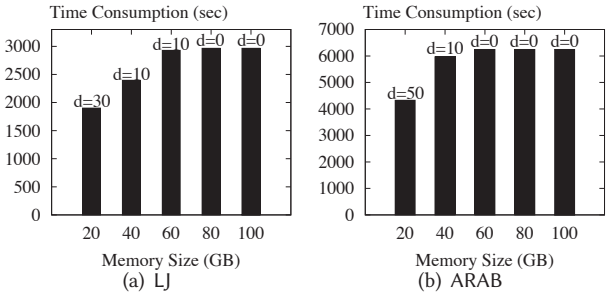

Figure 14: The Determination of d

Table 3: The Comparison Between CT-Index and CD

	Index Time (sec)		Index Size (MB)		Query Time (sec)	
	CD	CT-Index	CD	CT-Index	CD	CT-Index
TALK	6653.524	3.32	521.94	77.13	1.84E-6	8.93E-7
EPIN	35865.233	9.14	5839.77	223.82	9.261E-6	7.56E-6

Exp 5: The Test of Scalability. We randomly divided the nodes of a graph into 5 equally sized node groups and created 5 graphs for the cases of 20%, 40%, 60%, 80%, and 100%: the i-th graph is the induced subgraph on the first i node groups.

Figure 11 shows that the index size of all approaches increases smoothly with the number of nodes of the graph. CT-Index constantly has a smaller index size compared with PSL$^+$ and is better than PSL* in most of the cases. These results justify the scalability of CT-Index in the index size.

Figure 12 shows that the index time grows smoothly with the graph size for all four methods. The proposed approach CT-20 and CT-100 spends less time than PSL$^+$ in all cases and are better than PSL* on most of the cases.

Figure 13 shows that the query time of all approaches grows smoothly with the graph size for all four methods. The query time of PSL$^+$ (CT-0) smaller than that of CT-20 while that of CT-20 is smaller than that of CT-100. This conforms to the query time analysis of CT-Index. But even for CT-100, our query time is no slower than 10 times with PSL* on all test cases, which shows the superiority and scalability of the proposed method CT-Index.

Exp 6: The Comparison with other Approaches with Core Tree Decomposition. Denote by CD the algorithm of [3] (see Section 6 for details). Table 3 compares CT-Index with CD under bandwidth $d = 100$. Since CD ran out of memory on 28 out of 30 graphs tested, Table 3 shows only two graphs, TALK and EPIN.

CD has the index size 10× larger than CT-Index on EPIN. Moreover, CD is *four orders of magnitude slower* than CT-Index in index construction. Such large index size and slow index construction lead to the failure of CD on the other 28 graphs. The results conform to Table 1: CT-Index outperforms CD in all aspects.

Exp 7: The Efficiency of Finding d. The efficiency was evaluated on two real-world graphs, LJ and ARAB, with the memory limit mem_{lmt} ranging from 20 GB to 100 GB and upper bound $d_{ub} = 200$. Figure 14 shows the running time of finding d and the resulting bandwidth d.

On both graphs, a larger memory limit produces a smaller d; once the memory is large enough to hold the PSL index of the graph, $d = 0$ will be reported. A smaller d means larger index size and longer indexing time: on LJ, $d = 0$ (when the memory limit is 100GB) has the running time longer than $d = 30$ (under the memory limit 20GB) by only 56%. To conclude, a small d that is optimal for the current memory setting can be reported within a reasonable time especially when PSL ($d = 0$) fails to create the index. CT-Index can be used for distance queries in all settings.

Summary of the findings in the experiments. CT-Index reduces the *index size* and thus can index massive graphs such as UK0705 and UK07 that no other approaches can process. The *index time* of CT-Index is up to 21.85 times shorter than PSL$^+$. The *query time* of CT-Index comparable to PSL$^+$: CT-100 is on average 7.55 times slower than PSL$^+$ and on average 3.17 times slower than PSL*. Note that this is the pay off for the scalability of CT-Index that is tuned by the bandwidth d. The construction of CT-Index is 4 orders of magnitudes faster than other distance labeling based on core-tree decomposition. Overall, CT-Index shows that a slight sacrifice of the query time of CT (controlled under milliseconds) can lead to a cutting-edge scalability.

8 CONCLUSION

On big graphs with large treewidth (e.g., social networks and web graphs), 2-hop distance labeling approaches demand a massive index whose size becomes the bottleneck. We leverage the core-periphery property of these graphs and then propose a Core-Tree (CT) index to scale up, by reducing the index size, the distance labeling to the billion-scale graphs that can not be processed with existing 2-hop solutions. More importantly, such scalability comes at a negligible cost in query time. Extensive experiments verify the superiority of CT-Index on real graphs. This paper also provides treewidth-based theoretical analysis on distance labeling that may be of independent interest.

Acknowledgements. Miao Qiao is supported by Marsden Fund UOA1732, Royal Society of New Zealand. Lu Qin is supported by ARC DP160101513. Ying Zhang is supported by ARC DP180103096 and FT170100128. Lijun Chang is supported by ARC DP160101513 and FT180100256. Xuemin Lin is supported by NSFC61232006, 2018YFB1003504, ARC DP200101338, DP180103096 and DP170101628.

REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. F. Werneck. 2012. Hierarchical Hub Labelings for Shortest Paths. In *ESA*. 24–35.

[2] T. Akiba, Y. Iwata, and Y. Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*. 349–360.

[3] T. Akiba, C. Sommer, and K. Kawarabayashi. 2012. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*. 144–155.

[4] S. Arnborg, D. G. Corneil, and A. Proskurowski. 1987. Complexity of finding embeddings in ak-tree. *SIAM J. Algebraic Discrete Methods* 8, 2 (1987), 277–284.

[5] Anne Berry, Pinar Heggernes, and Geneviève Simonet. 2003. The Minimum Degree Heuristic and the Minimal Triangulation Process. In *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*. 58–70.

[6] H. L Bodlaender. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. on Computing* 25, 6 (1996), 1305–1317.

[7] P. Boldi and S. Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *WWW*. ACM Press, 595–601.

[8] L. Chang. 2019. Efficient Maximum Clique Computation over Large Sparse Graphs. In *SIGKDD*. 529–538.

[9] L. Chang, J. X. Yu, L. Qin, H. Cheng, and M. Qiao. 2012. The exact distance to destination in undirected world. *VLDB J.* 21, 6 (2012), 869–888.

[10] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. 2003. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[11] Rudolf Halin. 1976. S-functions for graphs. *J. of Geometry* 8, 1-2 (1976), 171–186.

[12] D. Harel and R. Endre Tarjan. 1984. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J. Comput.* 13, 2 (1984), 338–355.

[13] M. Jiang, A. W. Fu, R. C. Wong, and Y. Xu. 2014. Hop Doubling Label Indexing for Point-to-Point Distance Querying on Scale-Free Networks. *PVLDB* 7, 12 (2014), 1203–1214.

[14] J. Kunegis. 2013. Konect: the Koblenz Network Collection. In *WWW*. ACM, 1343–1350.

[15] J. Leskovec and A. Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[16] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. 2009. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[17] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin. 2019. Scaling Distance Labeling on Small-World Networks. In *SIGMOD*. 1060–1077.

[18] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. 2014. Computing Personalized PageRank Quickly by Exploiting Graph Structures. *PVLDB* 7, 12 (2014), 1023–1034.

[19] D. Ouyang, Lu Q., L. Chang, X. Lin, Y. Zhang, and Q. Zhu. 2018. When hierarchy meets 2-hop-labeling: efficient shortest distance queries on road networks. In *SIGMOD*. ACM, 709–724.

[20] N. Robertson and P. D. Seymour. 1986. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms* 7, 3 (1986), 309–322.

[21] R. A. Rossi and N. K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. http://networkrepository.com

[22] F. Wei. 2010. TEDI: efficient shortest path query answering on graphs. In *SIGMOD*. 99–110.

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

[24] J. Xu, F. Jiao, and B. Berger. 2005. A Tree-Decomposition Approach to Protein Structure Prediction. In *Fourth International IEEE Computer Society Computational Systems Bioinformatics Conference, CSB 2005, Stanford, CA, USA, August 8-11, 2005*. 247–256.