Scaling Distance Labeling on Small-World Networks

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

Distance labeling approaches are widely adopted to speed up the online performance of shortest distance queries. The construction of the distance labeling, however, can be exhaustive especially on big graphs. For a major category of large graphs, small-world networks, the state-of-the-art approach is Pruned Landmark Labeling (PLL). PLL prunes distance labels based on a node order and directly constructs the pruned labels by performing breadth-first searches in the node order. The pruning technique, as well as the index construction, has a strong sequential nature which hinders PLL from being parallelized. It becomes an urgent issue on massive small-world networks whose index can hardly be constructed by a single thread within a reasonable time. This paper scales distance labeling on small-world networks by proposing a Parallel Shortest-distance Labeling (PSL) scheme and further reducing the index size by exploiting graph and label properties. PSL insightfully converts the PLL's node-order dependency to a shortest-distance dependence, which leads to a propagation-based parallel labeling in D rounds where D denotes the diameter of the graph. Extensive experimental results verify our efficiency on billion-scale graphs and near-linear speedup in a multi-core environment.

KEYWORDS

Shortest Distance; Indexing; 2-hop Labeling; Parallel; Small-World Network; Compression; Algorithm

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319877

ACM Reference Format:
Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling Distance Labeling on Small-World Networks. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3319877

1 INTRODUCTION

Given a graph G, a shortest distance query $q(s,t)$ reports a minimized length of an s-t path on G. It is a fundamental primitive as either a main function or a building block of applications such as geographic navigation, Internet routing, socially tenuous group finding [25], influential community searching [16] and event detection [24]. Many of these applications cannot afford frequent online distance computations, and therefore, 2-hop labeling [9] and its variations prevail as indexing techniques.

The index size of 2-hop labeling, however, can be quadratic to the number n of the nodes in the graph. For each node v, 2-hop labeling selects a set of nodes as hubs and tags v with its distances to its hubs as labels. A query $q(s,t)$ minimizes, over all hubs r shared by s and t, the 2-hop distances from s to t via r, i.e., $dist(s,r) + dist(r,t)$. To report a precise distance, the shared hubs of s and t must hit — have a common node with — a shortest path between s and t. Such a requirement over all pairs, s and t, of nodes is called the 2-hop cover constraint. A label set that satisfies the 2-hop cover constraint can have a cardinality quadratic to n, especially on dense graphs. For example, a clique necessitates $\Omega(n^2)$ labels.

Finding a global minimum index size of 2-hop labeling, unfortunately, is NP-hard [9]. A local minimum, instead, can be reached by iteratively pruning redundant labels[1]. A label of a node v is redundant if the remaining labels in the label set still satisfy the 2-hop cover constraint. The pruning technique, however, has a strong sequential nature

[1]In many labeling approaches, the labels are pruned in an implicit way — a label will not be generated if pruning it is guaranteed to be safe.

— pruning one label will affect the redundancy of another label. Consider two nodes u and v on the same shortest path between two nodes s and t. The moment when both s and t have the hub set of $\{u, v\}$, all labels on s and t are redundant. After pruning the label on s with the hub u, however, both labels on s and t with the hub v become critical. Due to such a dependency, the order of the pruning has a great influence on the pruning outcome and effectiveness.

The optimization of the pruning order is based on graph properties. For example, the planarity and hierarchical structure of road networks have been well explored to reach a scalable solution (see [19] as an entrance). For a major category of real graphs, small-world [26, 28] networks, the state-of-the-art approach is Pruned Landmark Labeling (PLL) [3].

The key to PLL's success on small-world networks is to encode the highly clustered topology into a node order and construct/prune labels strictly following the node order.

(1) PLL prunes labels based on a node order that prioritizes the high-centrality[2] nodes. The label on a node s to its hub t is pruned if their distance can be answered by labels from s and t to a higher ranked hub. Therefore, a high-centrality hub r is able to prune labels along a large number (due to the clustered topology of the graph) of shortest paths hit by r.

(2) PLL prunes a majority of labels in an implicit way. In other words, PLL constructs *pruned* labels directly as opposed to following a construct-and-then-prune paradigm. This is done by performing a *pruned* Breadth-First-Search (BFS) sourced from a hub r with the assignment of r *sequentially* following the node order.

It is worth noting that the index construction of PLL is highly node-order dependent: the pruning procedure in the BFS of hub r is dependent on the pruned labels constructed for the predecessor, in the node order, of r. Such a strong sequential nature of PLL hinders its parallelization.

On the other hand, the index time becomes an urgent issue for massive small-world networks whose index can hardly be constructed by a single thread within a reasonable time. For example, for the graph SINA[3] with 58 million nodes and 261 million edges, PLL cannot finish the indexing within 3 days. The same situation applies to UK[4] which has 77 million nodes and 2.9 billion edges.

This paper focuses on the scalability issue of the 2-hop distance labeling of small-world networks. We propose non-trivial algorithms to parallel the indexing process of PLL and further reduce the index size. The scalability of our proposed approach is confirmed by extensive experiments. Our contributions are summarized as follows.

- We propose a Parallel Shortest distance Labeling approach PSL upon a novel and insightful conversion from the node-order label dependency of PLL to a shortest-distance label dependency. This conversion leads to a non-trivial propagation based labeling process. The algorithm completes in D rounds where D denotes the diameter of the graph — small for small-world networks. The resulting labels are identical to those constructed in the sequential algorithm of PLL.
- We provide a graph compression technique for all 2-hop labeling approaches and also exploit the property of PLL to further reduce the index size.
- We conduct extensive experiments to verify the performance of the proposed techniques. In a single-core environment, our index reduction technique dramatically shrinks the index size and improves the index time. In a multi-core environment, our PSL approach shows near-linear speed-up in parallelism. The proposed techniques jointly enable the index construction on networks with billion scale offline, which verifies the efficiency of the proposed approach.

The rest of the paper is organized as follows. Section 2 introduces the state-of-the-art 2-hop labeling approach on small-world networks. Section 3 devises a distance labeling algorithm. Section 4 introduces two index reduction techniques. Section 5 summarizes related works. Section 6 experimentally evaluates our proposed approaches on real small-world networks and Section 7 concludes the paper.

2 PRELIMINARY

2.1 Shortest Distance Problem

Let G be a graph with vertex set V_G and edge set E_G. Denote by n and m the number $|V_G|$ of nodes and $|E_G|$ of edges in the graph, respectively. For each node $v \in V_G$, denote by $N_G(v) = \{u|(u, v) \in E_G\}$ the **neighbors** of v and $deg_G(v) = |N_G(v)|$ the degree of node v in G. We mainly use undirected graphs in the paper; Appendix C extends our techniques to directed graphs. Without loss of generality, we assume a connected graph G. Our techniques can be extended to disconnected graphs easily.

Let $p(s, t) = \{v_1, v_2, \cdots, v_k\}$ with $s = v_1$ to $t = v_k$. p is a path on G if, for $\forall 1 \leq i \leq k$, edge $(v_i, v_{i+1}) \in E_G$. For an $i \in [1, k]$, denote by $p(s, t) = p(s, v_i) + p(v_i, t)$ the concatenation of two paths. The length of a path $p(s, t)$ is the number of edges on the path, i.e., $|p(s, t)| = k - 1$. The shortest path between s and t is the path with shortest length. The shortest length is the length of the shortest path, denoted

[2]The centrality can be defined with degree, closeness and betweenness [17].
[3]http://networkrepository.com/index.php
[4]http://law.di.unimi.it

as $dist_G(s,t)$. Given a graph G, a **point-to-point distance query** $q(s,t)$ with $s,t \in V$ returns the shortest distance $dist_G(s,t)$ between s and t. When the context is clear, we use $V, E, N(v), deg(v), dist(s,t)$ to represent the node set, edge set, neighbor set of v, the degree of v and the shortest distance from s to t, respectively, for simplicity.

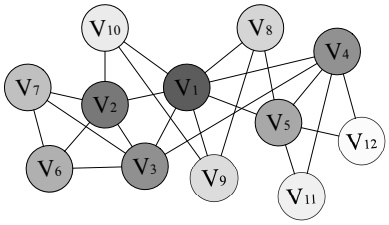


Figure 1: Graph G

Example 2.1. Fig. 1 shows a network $G = (V, E)$ with 12 nodes and 23 edges. The neighbors of v_6 are $N(v_6) = \{v_2, v_3, v_7\}$. Two paths between v_4 and v_6 are $p_1(v_4, v_6) = \{v_4, v_3, v_6\}$, $p_2(v_4, v_6) = \{v_4, v_1, v_2, v_6\}$. The shortest path $p_1(v_4, v_6)$ has the shortest length 2.

2.2 2-Hop Labeling for Distance Queries

To efficiently process point-to-point distance queries, 2-hop labeling approach [9] precomputes the distances from each node to preselected hub nodes and uses the 2-hop distances via hubs to answer a query. Below we introduce the 2-hop labeling approach that has been slightly updated [3, 11, 13] to enable label reduction.

A **labeling function** L maps each node $v \in V$ to a **label set** $L(v)$. $L(v)$ consists of a set of **label entries** where each entry is a key/value pair $(u, dist(v,u))$ with a node $u \in V$ and the distance between v and u. The **hub nodes** of v are the projections of $L(v)$ on the key, i.e., $C(v) = \{u | (u, dist(v,u)) \in L(v)\}$. C is called the **hub function** of L. $\{L(v) | v \in V\}$ is a 2-hop labeling if L satisfies the **2-hop cover constraint** below.

Definition 2.2 (2-hop Cover Constraint [9]). A labeling function L satisfies the 2-hop cover constraint if for any node pair s and t, $C(s) \cap C(t)$ shares a node with a shortest path from s to t.

For a 2-hop labeling L, the **label size** $|L(v)|$ of a node v is the number of entries in $L(v)$. Denote by δ the largest label size of G, i.e., $\delta = max_{v \in V}(|L(v)|)$.

Given a 2-hop labeling L, a distance query $q(s,t)$ is answered with $Query(s,t,L)$ defined below.

$$Query(s,t,L) = \min_{u \in C(s) \cap C(t)} dist(s,u) + dist(u,t)$$

Lemma 2.3. *For a 2-hop labeling L that satisfies the 2-hop cover constraint, $Query(s,t,L) = dist(s,t)$.*

Proof. See Appendix A. □

Assume that the label entries in each label set are stored in the ascending order of the key. The online cost of answering $q(s,t)$ is on retrieving and merging the entries in $L(s)$ and $L(t)$. Thus, the query time complexity is $O(|L(s)| + |L(t)|)$.

2.3 Prune Landmark Labeling Approach

Prune Landmark Labeling Approach (PLL) is the state-of-the-art 2-hop labeling approach on small-world networks.

Node Order. The effectiveness of PLL heavily relies on a node order and the corresponding ranking functions $r(v)$ on nodes $v \in V$. A typical ranking function prioritizes nodes with higher degrees and resorts to original ID to break ties. Specifically, for two nodes v and v', $r(v) > r(v')$ if

- $deg(v) > deg(v')$ or
- $deg(v) = deg(v')$ and $ID(v) > ID(v')$.

Without loss of generality, we rename the nodes such that

$$r(v_1) > r(v_2) > \cdots > r(v_n).$$

Example 2.4. We rank the nodes in Fig. 1 according to their degrees. When two nodes have the same degree, the tie is broken by the original ID of the node. We re-order the nodes such that $r(v_1) > r(v_2) > \cdots > r(v_{12})$.

The node order can similarly be selected based on other node centralities, e.g., betweenness centrality and closeness centrality. Please see [17] as a comprehensive comparison.

PLL with Pruned BFS. Algorithm 1 shows the process of PLL. Given a graph G and a node order $v_1, v_2, \cdots, v_n$, PLL constructs a pruned 2-hop labeling L^{PLL} in n rounds (Line 1). In the i-th round, $i \in [1,n]$, PLL performs a pruned BFS search (a standard BFS search apart from Line 6-8) sourced from v_i. To prune the BFS, PLL tests if the existing index can already report the distance between v_i and a node u (Line 6). If yes, u will neither be labeled nor expanded in this round (Line 7); otherwise, a label with hub v_i will be added to u (Line 8) and u will be expanded right away (Line 9-12). Obviously, on the nodes that are either unexpanded or unreached, the labels with hub v_i are conceptually pruned.

Lemma 2.5 ([3]). *The index of* PLL *satisfies the 2-hop cover constraint.*

We analyse the index time of PLL in Theorem 2.6.

Theorem 2.6. *The time complexity of* PLL *is $O(\delta^2 \cdot m)$ where δ denotes the maximum label size over all nodes in the graph.*

Proof. Every execution of Line 8 will add a new label entry to PLL and will expand a node u to its neighbors (Line 9-12), for each neighbor of u, when it is popped, Line 6 will call the query function to calculate the distance between v_i and u. The time complexity of $Query(v_i, u, L^{\text{PLL}})$ is $O(\delta)$.

Algorithm 1: PLL

Input: Graph $G(V, E)$
Output: The index L^{PLL}

```
for i = 1, 2, ··· , n do
    Q ← a queue with only one element v_i;
    dist(v_i) ← 0 and dist(v) ← ∞, ∀v ∈ V \ v_i;
    while Q ≠ ∅ do
        u ← Q.pop();
        if Query(v_i, u, L^PLL) ≤ dist(u) then
            continue;
        L^PLL(u) ← L^PLL(u) ∪ {(v_i, dist(u))};
        for w ∈ N(u) do
            if dist(w) = ∞ then
                dist(w) ← dist(u) + 1;
                Q.push(w);
return L^PLL ;
```

The number of function calls is $\Sigma_{u\in V}\Sigma_{r\in C(u)} deg(u) \leq \delta m$. Therefore, the overall index time of PLL is $O(\delta^2 \cdot m)$. □

We show index time of PLL by investigating δ on two representative small-world networks. Youtube, a social network, has $\delta = 1665$. UK-Tpd, a web graph, has $\delta = 2866$. When the number of edges is 1 billion, $\delta^2 m = 10^{15}$. This amount of work for constructing L^{PLL}, therefore, can hardly be finished on a single thread within a reasonable time.

3 PARALLELIZED DISTANCE LABELING

Section 3.1 -3.2 revisit PLL to identify the label properties and order dependency. Section 3.3 transforms the order dependency in PLL to distance dependency. By utilizing the distance dependency, Section 3.4 proposes a practical approach in constructing the index in parallel.

3.1 Label Property

The labels of PLL show an important node-order property.

Theorem 3.1. *For any two nodes $\forall u, v \in V$, v is a hub of u under* PLL*, i.e., $(v, dist(v, u)) \in L^{\text{PLL}}(u)$, if and only if v is the highest ranked node on all the shortest paths from u to v.*

Proof. Let S be the set of nodes on all the shortest paths from u to w. Let w be the highest ranked node in S.

We prove that all nodes in S have w as their hubs in L^{PLL} by contradiction. Assume that there is a node z in S such that z does not have a hub of w in L^{PLL}. Consider the round of Algorithm 1 where the pruned BFS sourced w is performing. Let L' be the snapshot of the PLL label set right before the round begins. Given that z has no hub of w, then either

- z is explicitly pruned: $Query(z, w, L') = dist(w, z)$, or
- z is implicitly pruned: z is not reached since there is at least a node z' on the shortest path from w to z explicitly pruned with $Query(z', w, L') = dist(w, z')$.

In either case, it requires a common hub between w and z (or z') to produce the query result, which is impossible since i) $z, z' \in S$ and ii) w has the highest rank in S and iii) L' does not include any hub ranked higher than w. Contradiction.

Since all nodes in S have w as their hubs in L^{PLL}, we prove the two directions of the theorem in two cases: 1) if $w = v$, that is, v is the highest ranked node in S, then v is a hub of $u \in S$ and 2) if $r(w) > r(v)$, when before the pruned BFS sourced from v is performed, w is already a common hub of u and v. As w is on the shortest path between u and v, the label with hub v on u is pruned and not in PLL. □

Lemma 3.2-3.4 are derived from Theorem 3.1.

Lemma 3.2. *If v is a hub of u, $r(v) > r(u)$.*

Proof. Since v has the highest rank on a shortest path from v to u (Theorem 3.1), $r(v) > r(u)$. □

Lemma 3.3. *For $\forall u \in V$, $(u, 0) \in L^{\text{PLL}}(u)$.*

Proof. We make the path as $p(u, u)$ and according to Theorem 3.1, $(u, 0)$ will be always inserted to $L^{\text{PLL}}(u)$. □

Lemma 3.4. *For $\forall (u, v) \in E$, $(u, 1) \in L^{\text{PLL}}(v)$, if $r(u) > r(v)$; otherwise,$(v, 1) \in L^{\text{PLL}}(u)$.*

Proof. Let $p(u, v)$ be the path with an edge. According to Theorem 3.1, the higher ranked node is the hub node. □

3.2 Order Dependency

To see the dependency among the labels, we partition the labels in L^{PLL} according to their hub nodes. Let $v_1, v_2, \cdots, v_n$ be the node order under which label set L^{PLL} was constructed.

We define two sets with particular meanings. Recall that PLL has n rounds where the i-th round performs a pruned BFS sourced from v_i. We denote by $L^{\text{PLL}}_{<i}(u)$ the snapshot of $L^{\text{PLL}}(u)$ at the beginning of the i-th round and by $L^{\text{PLL}}_i(u)$ the incremental label of u built in the i-th round.

Definition 3.5 (Order Specific Label Set).

$$L^{\text{PLL}}_i(u) = \{(v_i, dist(v_i, u)) \in L^{\text{PLL}}\},$$

for $\forall i \in [1, n]$, $u \in V$. Let $L^{\text{PLL}}_i = \bigcup_{u\in V} L^{\text{PLL}}_i(u)$.

Definition 3.6 (Order Partial Label Set).

$$L^{\text{PLL}}_{<i}(u) = \{(v_j, dist(v_j, u)) \in L^{\text{PLL}} | j < i\},$$

for $\forall i \in [1, n+1]$, $u \in V$. Let $L^{\text{PLL}}_{<i} = \bigcup_{u\in V} L^{\text{PLL}}_{<i}(u)$. $L^{\text{PLL}}_{<n+1} = L^{\text{PLL}}$.

The following lemma shows that the pruning condition in Algorithm 1 leads to an order dependency among labels.

Lemma 3.7 (Order Dependency). *$L_i^{\mathsf{PLL}}(u)$ depends on $L_{<i}^{\mathsf{PLL}}(u)$. Specifically, $L_i^{\mathsf{PLL}}(u) =$*

$$\begin{cases} \{(v_i, dist(v_i,u))\} & Query(v_i,u,L_{<i}^{\mathsf{PLL}}) > dist(v_i,u); \\ \emptyset & otherwise. \end{cases}$$

Proof. Let S be the set of nodes on the shortest path from v_i to u (including v_i and u). Let w be the node with the highest rank in S. If $v_i = w$, according to Theorem 3.1, i) v_i is a hub of u and ii) for $\forall v \in S \backslash v_i$, v is a not a hub of v_i, and thus $Query(v_i,u,L_{<i}^{\mathsf{PLL}}) > dist(v_i,u)$. If $r(v_i) < r(w)$, then v_i is not a hub of u and label $(w, dist(w,v_i)), (w, dist(w,u)) \in L_{<i}^{\mathsf{PLL}}$ and thus $Query(v_i,u,L_{<i}^{\mathsf{PLL}}) = dist(v_i,u)$. □

Lemma 3.7 shows that $L_i^{\mathsf{PLL}}(u)$ depends on $L_{<i}^{\mathsf{PLL}}(u)$ while $L_{<i}^{\mathsf{PLL}}(u)$ depends on $L_{i-1}^{\mathsf{PLL}}(u)$. Such a convolved dependency can hardly be removed as long as the labels are built in the node order.

Example 3.8. Table 1 illustrates how PLL constructs the index. A cell at the row of v_i and the column of v_j records the order specific label of v_i at the j-th round. In column v_1, pruned BFS inserts v_1 into $L_1^{\mathsf{PLL}}(u)$, $\forall u \in V$. In column v_2, PLL performs pruned BFS and v_2 becomes the hub of $\{v_2, v_3, v_6, v_7, v_{10}\}$ due to the pruning condition of $L_1^{\mathsf{PLL}} = \{L_1^{\mathsf{PLL}}(u) | u \in V\}$. The order dependency in the column v_7: partial set $L_{<7}^{\mathsf{PLL}} = \bigcup_{i<7, u \in V} L_i^{\mathsf{PLL}}(u)$ prunes the labels on all nodes except on v_7.

3.3 Distance Dependency

To break the order dependency in the label construction, consider the pruning condition of Line 6, Algorithm 1. When $Query(v_i,u,L_{<i}^{\mathsf{PLL}}) = dist(u,v_i)$ prunes a node label on u, there must be two labels on u and v_i, respectively, to a common hub w such that $dist(u,w) + dist(w,v_i) = dist(u,v_i)$. Therefore, $dist(u,w)$ and $dist(w,v_i)$ must be no greater than $dist(u,v_i)$. In other words, all the labels with distances greater than $dist(u,v_i)$ have no effect on the query result of $Query(v_i,u,L_{<i}^{\mathsf{PLL}})$ and the corresponding pruning outcomes.

From the above intuition, we group the label entries in L^{PLL} based on their label distances. The rearranged label sets will pave the way to our Parallel Shortest distance Labeling (PSL) approach (Section 3.4) and are thus called PSL label sets. Let D be the diameter of the graph G.

Definition 3.9 (Distance Specific Label Set).

$$L_d^{\mathsf{PSL}}(u) = \{(v, dist(v,u)) \in L^{\mathsf{PLL}}(u) | dist(v,u) = d\},$$

for $\forall u \in V, d \in [1, D]$. Let $L_d^{\mathsf{PSL}} = \{L_d^{\mathsf{PSL}}(u) | u \in V\}$.

Similarly, the partial label of a node then becomes the set of label entries with distance less than a certain distance and is defined in Definition 3.10.

Definition 3.10 (Distance Partial Label Set).

$$L_{<d}^{\mathsf{PSL}}(u) = \{(v, dist(v,u)) \in L^{\mathsf{PLL}}(u) | dist(v,u) < d\},$$

for $\forall u \in V, d \in [1, D+1]$. Let $L^{\mathsf{PSL}} = \bigcup_{u \in V} L^{\mathsf{PSL}}(u)$. In particular, $L^{\mathsf{PSL}}(u) = L_{<D+1}^{\mathsf{PSL}}(u)$.

The equivalence of the index L^{PLL} and the novel index L^{PSL} is given in Theorem 3.11.

Theorem 3.11. $L^{\mathsf{PSL}} = L^{\mathsf{PLL}}$.

Proof. Since all the label $(v, dist(v,u))$ in L^{PLL} has $dist(v,u) \le D$, L^{PSL} includes all labels in L^{PLL} and has no other labels according to the definition. □

Example 3.12. Table 1 shows a rearrangement of labels in PLL. A cell with row v_i and column j denotes label set of $L_j^{\mathsf{PSL}}(v_i)$ — the PLL labels of v_i whose distances are j.

Distance Dependency. Definition 3.9 and Definition 3.10 provide us an opportunity in removing the order dependency in the label construction process.

Theorem 3.13 (Distance Dependency). *$L_d^{\mathsf{PSL}}(u)$ depends on $L_{<d}^{\mathsf{PSL}}$. Specifically, given a node u, for a node $v \in V$ with $r(v) > r(u)$ and $dist(u,v) = d$, $(v, dist(v,u)) \in L_d^{\mathsf{PSL}}(u)$ if and only if $Query(u,v,L_{<d}^{\mathsf{PSL}}) > d$.*

Proof. Consider a node v with $dist(u,v) = d$. Denote by S the set of nodes on the shortest paths from u to v and let w be the highest ranked node in S. According to Theorem 3.1, we have two exclusive cases:

- $w = v$ *if and only if* v is the hub of u;
- $w \neq v$ means that
  - w is the hub of both u and v, and
  - $dist(u,w), dist(w,v) < d$,

  and therefore, $Query(u,v,L_{<d}^{\mathsf{PSL}}) = d$.

Therefore, if $(v, dist(v,u)) \notin L_d^{\mathsf{PSL}}(u)$, namely, v is not a hub of u, then $w \neq v$, and then $Query(u,v,L_{<d}^{\mathsf{PSL}}) = d$. Besides, if $(v, dist(v,u)) \in L_d^{\mathsf{PSL}}(u)$, namely, v is a hub of u, v is the highest ranked node in S and therefore, no other node in S can be a hub of v, that is, $Query(u,v,L_{<d}^{\mathsf{PSL}}) > d$. □

By transforming the order dependency to distance dependency, it is possible to complete the index construction in D rounds where D denotes the diameter of the graph.

Example 3.14. In Table 1, each row corresponds to the partial label of a node while each column corresponds to the incremental labels regarding each distance value. When $d = 0$, each node add to itself since the distance between itself is zero. When $d = 1$, we either add nodes that are 1-hop

Table 1: The Index of PLL and PSL

	PLL												PSL		
ID	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	0	1	2
v_1	$(v_1, 0)$	-	-	-	-	-	-	-	-	-	-	-	$(v_1, 0)$	-	-
v_2	$(v_1, 1)$	$(v_2, 0)$	-	-	-	-	-	-	-	-	-	-	$(v_2, 0)$	$(v_1, 1)$	-
v_3	$(v_1, 1)$	$(v_2, 1)$	$(v_3, 0)$	-	-	-	-	-	-	-	-	-	$(v_3, 0)$	$(v_1, 1)$ $(v_2, 1)$	-
v_4	$(v_1, 1)$	-	$(v_3, 1)$	$(v_4, 0)$	-	-	-	-	-	-	-	-	$(v_4, 0)$	$(v_1, 1)$ $(v_3, 1)$	-
v_5	$(v_1, 1)$	-	-	$(v_4, 1)$	$(v_5, 0)$	-	-	-	-	-	-	-	$(v_5, 0)$	$(v_1, 1)$ $(v_4, 1)$	-
v_6	$(v_1, 2)$	$(v_2, 1)$	$(v_3, 1)$	-	-	$(v_6, 0)$	-	-	-	-	-	-	$(v_6, 0)$	$(v_2, 1)$ $(v_3, 1)$	$(v_1, 2)$
v_7	$(v_1, 2)$	$(v_2, 1)$	$(v_3, 1)$	-	-	$(v_6, 1)$	$(v_7, 0)$	-	-	-	-	-	$(v_7, 0)$	$(v_2, 1)$ $(v_3, 1)$ $(v_6, 1)$	$(v_1, 2)$
v_8	$(v_1, 1)$	-	-	-	$(v_5, 1)$	-	-	$(v_8, 0)$	-	-	-	-	$(v_8, 0)$	$(v_1, 1)$ $(v_5, 1)$	-
v_9	$(v_1, 1)$	-	-	-	-	-	-	$(v_8, 1)$	$(v_9, 0)$	-	-	-	$(v_9, 0)$	$(v_1, 1)$ $(v_8, 1)$	-
v_{10}	$(v_1, 1)$	$(v_2, 1)$	-	-	-	-	-	-	$(v_9, 1)$	$(v_{10}, 0)$	-	-	$(v_{10}, 0)$	$(v_1, 1)$ $(v_2, 1)$ $(v_9, 1)$	-
v_{11}	$(v_1, 2)$	-	$(v_3, 2)$	$(v_4, 1)$	$(v_5, 1)$	-	-	-	-	-	$(v_{11}, 0)$	-	$(v_{11}, 0)$	$(v_4, 1)$ $(v_5, 1)$	$(v_1, 2)$ $(v_3, 2)$
v_{12}	$(v_1, 2)$	-	$(v_3, 2)$	$(v_4, 1)$	$(v_5, 1)$	-	-	-	-	-	-	$(v_{12}, 0)$	$(v_{12}, 0)$	$(v_4, 1)$ $(v_5, 1)$	$(v_1, 2)$ $(v_3, 2)$

away to a node u or prune the 1-hop away nodes. Note that according to Lemma 3.2, only higher ranking nodes can be hubs of lower ranking nodes. When $d = 2$, we either add nodes that are 2-hop away to a node u or prune the 2-hop away nodes. For instance, if $u = v_{11}$, the node v_1 that is 2-hop away is added into $L_2^{\text{PSL}}(v_{11})$. But node v_8 is pruned since we can make use of v_5, which is less than two hops away with v_8, to prune it.

3.4 The Parallelized Labeling Method

To apply Theorem 3.13 to generate $L_d^{\text{PSL}}(u)$, all the node pairs with distance equal to d are to be examined which is also expensive. This section provides a practical algorithm, Parallel Shortest distance Labeling (PSL), to construct the index L^{PSL} in label propagations.

Propagation-Based Label Construction. This section provides a positive answer to the following question: can we build the distance specific label $L_d^{\text{PSL}}(u)$ by gathering the labels of its neighbors, namely, $L_{d-1}^{\text{PSL}}(v)$, for $v \in N(u)$? We formally show that $\bigcup_{v \in N(u)} L_{d-1}^{\text{PSL}}(v)$ is sufficient to create $L_d^{\text{PSL}}(u)$ in Lemma 3.15

Lemma 3.15. *All the hub nodes of labels in $L_d^{\text{PSL}}(u)$ appear in labels $\bigcup_{v \in N(u)} L_{d-1}^{\text{PSL}}(v)$ as hub nodes.*

Proof. We show that if a node is not a hub of any node $v \in N(u)$ in $L_{d-1}^{\text{PSL}}(v)$, then it is not a hub of u in $L_d^{\text{PSL}}(u)$. Let $w \neq u$ be a hub of u in $L_d^{\text{PSL}}(u)$ but is not a hub of any node $v \in N(u)$ in $L_{d-1}^{\text{PSL}}(v)$. Note that the PLL was built in a BFS search. Consider the round when the pruned BFS search is sourced from w. Since $w \neq u$ and w is a hub of u, there is a shortest path from w to u such that w is a hub of all nodes on the path. Let s be the predecessor of u on the shortest path. $s \in N(v)$ and $(w, dist(w, s)) \in L^{\text{PLL}}$. Since $dist(w, s) = d - 1$, w is a hub of $L_{d-1}^{\text{PSL}}(s)$, contradiction. □

Pruning Conditions. From Lemma 3.15, we can construct $L^{\text{PSL}}(u)$ in an iterative way and the initial condition is given in Lemma 3.3 by inserting u to the label $L_0^{\text{PSL}}(u)$ as its own hub. However, pouring all nodes in $\bigcup_{v \in N(u)} L_{d-1}^{\text{PSL}}(v)$ directly into $L_d^{\text{PSL}}(u)$ produces a large set of candidate labels. Therefore, we propose two rules to prune unnecessary label entries.

Lemma 3.16. *A hub w in the label set $\bigcup_{v \in N(u)} L_{d-1}^{\text{PSL}}(v)$ is not a hub of u if $r(w) < r(u)$.*

Proof. Lemma 3.2. □

Lemma 3.17. *A hub w in the label set $\bigcup_{v \in N(u)} L_{d-1}^{\text{PSL}}(v)$ is not a hub of u in $L_d^{\text{PSL}}(v)$ if $Query(w, u, L_{<d}^{\text{PSL}}) \leq d$.*

Proof. If $Query(w, u, L_{<d}^{\text{PSL}}) < d$, then $dist(w, u) < d$, w is not a hub of u with distance $dist(w, u) = d$. If $Query(w, u, L_{<d}^{\text{PSL}}) = d$, we discuss in two cases.

- $dist(w, u) < d$, w is not a hub of u with distance d.
- $dist(w, u) = d$, there is a node z on the shortest path between w and u with $r(z) > r(w)$. According to Theorem 3.1, w is not be a hub of u in L^{PLL}.

Therefore, w is not a hub of u if $Query(w, u, L_{<d}^{\text{PSL}}) \leq d$. □

Based on the above pruning rules, we propose our label propagation function to find the exact $L_d^{\text{PSL}}(u)$, $\forall u \in V$.

Denote by $C_d(v)$ the set of hub nodes in label set $L_d^{\text{PSL}}(v)$, for $\forall v \in V$ and $d \in [1, D + 1]$.

Theorem 3.18 (Label Propagation Function).

$$L_d^{\text{PSL}}(u) = \bigcup_{w \in C_{d-1}(v),\ for\ \forall v \in N(u)} L_d^{\text{PSL}}(u, w) \tag{1}$$

$$where\ L_d^{\text{PSL}}(u, w) = \begin{cases} \emptyset,\ if\ r(w) < r(u)\ or\ Query(w, u, L_{<d}^{\text{PSL}}) \leq d; \\ \{(w, dist(w, u))\},\ otherwise. \end{cases} \tag{2}$$

Proof. Denote by L' the label set computed from Equation (1). We show that $L' = L_d^{\mathsf{PSL}}(u)$ in two directions. Due to the correctness of Lemma 3.15 and the pruning conditions, the label set $L_d^{\mathsf{PSL}}(u) \subseteq L'$. The follow parts prove $L' \subseteq L_d^{\mathsf{PSL}}(u)$. Let $(w, dist(w,u))$ be a label in L'. Equation (2) shows that $r(w) > r(u)$ and $Query(w, u, L_{<d}^{\mathsf{PSL}}) > d$.

If in L^{PLL}, w is not a hub of u, then according to Theorem 3.1, there is a node s that in S — the set of all nodes in the shortest path between w and u — with $r(s) > r(w) > r(u)$. Therefore, $dist(w,s), dist(s,u) < d$ and $dist(w,u) \leq d$, and thus, $Query(w, u, L_{<d}^{\mathsf{PSL}}) \leq d$, contradiction.

Therefore, w is a hub of u in L^{PLL}. Besides, if $dist(w,u) < d$, $Query(w, u, L_{<d}^{\mathsf{PSL}}) = dist(w,u) < d$, contradiction. Thus, $dist(w,u) = d$. Now we have proved that w is a hub of u in L^{PLL} with $dist(w,u) = d$, i.e., w is a hub of u in $L_d^{\mathsf{PSL}}(u)$ which completes the proof. □

The PSL Algorithm. Algorithm 2 puts all parts of PSL together. $L_0^{\mathsf{PSL}}(u)$ is obtained by add u to itself (Line 1). Then, for each edge, the higher ranked node v is added into lower ranked node u to form $L_1^{\mathsf{PSL}}(u)$ according to Lemma 3.4 (Line 2-4). If L_{d-1}^{PSL} is empty — the path with length $d-1$ is covered by $L_{<d-1}^{\mathsf{PSL}}$ — we find the final index (Line 6). Otherwise, nodes are parallelly processed to build L_d^{PSL} for $d > 1$ (Line 7-12): each node u first finds its superset $cand(u)$ (Lemma 3.15) (Line 8) and then, pruning conditions 3.16-3.17 apply (Line 10-11). Entry $(w, dist(w,u))$ is then added to $L_d^{\mathsf{PSL}}(u)$ (Line 11-12).

Algorithm 2: PSL

```
   Input: Graph G(V, E)
   Output: The index L^PSL
1  Insert (u, 0) into L_0^PSL(u), ∀u ∈ V;
2  for (u, v) ∈ E do
3      if r(u) > r(v) then Insert (u, 1) into L_1^PSL(v);
4      else Insert (v, 1) into L_1^PSL(u);
5  d ← 2;
6  while L_{d-1}^PSL is not empty do
7      for u ∈ V in parallel do
8          cand(u) ← hubs in L_{d-1}^PSL(v), ∀v ∈ N(u);
9          for each node w ∈ cand(u) do
               // Pruning Condition Lemma 3.16
10             if r(w) < r(u) then continue;
               // Pruning Condition Lemma 3.17
11             if Query(w, u, L_{<d}^PSL) ≤ d then continue;
12             Insert (w, d) into L_d^PSL(u);
13     d ← d + 1;
14 return L^PSL;
```

Example 3.19. In Fig. 2(a), each node $u \in V$ is added to $L_0^{\mathsf{PSL}}(u)$ for $d = 0$. In Fig. 2(b), for each edge (u,v), v is added to $L_1^{\mathsf{PSL}}(u)$ if $r(v) > r(u)$. For instance, $L_1^{\mathsf{PSL}}(v_3) = \{(v_1,1),(v_2,1)\}$, $L_1^{\mathsf{PSL}}(v_2) = \{(v_1,1)\}$, $L_1^{\mathsf{PSL}}(v_7) = \{(v_2,1),(v_3,1),(v_6,1)\}$, In Fig. 2(c), for each node u, hubs in $\{L_1^{\mathsf{PSL}}(w) | w \in N(u)\}$ are candidate hubs and then added to $L_2^{\mathsf{PSL}}(u)$ if the pass pruning conditions. v_6 has three neighbors v_2, v_3, v_7. Then, candidate nodes are $\{v_1, v_2, v_3, v_6, v_7\}$. $(v_1, 2)$ will be put into $L_2^{\mathsf{PSL}}(v_6)$ since the current index gives the answer ∞ and $r(v_1) > r(v_6)$. $\{v_2, v_3, v_6\}$ will be pruned by the current index while v_7 will be pruned since $r(v_7) < r(v_6)$. Therefore, $L_2^{\mathsf{PSL}}(v_6) = \{(v_1, 2)\}$.

Theorem 3.20. *The time complexity of* PSL *under one core environment is* $O(\delta^2 \cdot m)$.

Proof. Let $L^{\mathsf{PSL}} = L_{<D+1}^{\mathsf{PSL}} = L^{\mathsf{PLL}}$. For each label in $L^{\mathsf{PSL}}(v)$, it has been collected by each of v's neighbors once as candidates (Line 11). For each candidate, a query (Line 15) is called in $O(\delta)$ time. The total cost is $\Sigma_{v \in V} \delta d(v) \times \delta = O(\delta^2 m)$. □

4 INDEX SIZE REDUCTION

Parallel index construction reduces the index time while leaving the index size $L^{\mathsf{PSL}} = L^{\mathsf{PLL}}$ unchanged. This section improves the scalability of the PSL by reducing the index size. Section 4.1 reduces the graph size using the equivalence relationships among nodes. Section 4.2 optimizes the index size of PSL based on an observation on the label distribution.

4.1 Equivalence Relation Reduction

We consider the equivalence of two nodes u and v based on their neighbors. Depending on whether u and v have an edge, we define two types of equivalence relations.

Definition 4.1 (Node Equivalence Relations). For $u, v \in V$,

- $u \simeq_1 v$ if $N(u) = N(v)$;
- $u \simeq_2 v$ if $N(u) \cup \{u\} = N(v) \cup \{v\}$.

It can be verified that $\simeq_1$ and $\simeq_2$ are equivalence relations. Their reflexive, symmetric and transitive properties are inherited from the equality operator over node sets.

Since $u \notin N(u)$, $u \simeq_1 v$ requires that u and v have no edge while $u \simeq_2 v$ requires that u and v must have an edge.

Each equivalence relation partitions V into disjoint equivalent classes: the equivalent class of a node v includes all the nodes that are equivalent to v. We say an equivalent class non-trivial if it includes at least two nodes. Definition 4.2 obtains nodes in non-trivial equivalent classes under the two equivalence relations and Lemma 4.4 shows that these non-trivial equivalent classes are disjoint.

Definition 4.2. Define three vertex sets V_1, V_2 and V_3 with

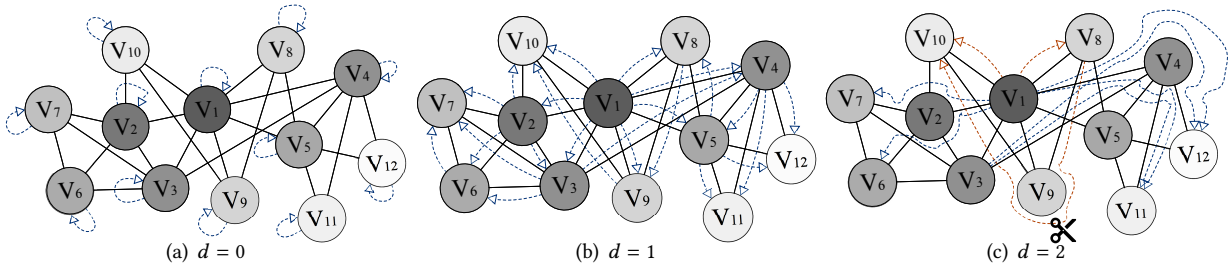


Figure 2: The execution of PSL from $d = 0$, $d = 1$ to $d = 2$

- $V_1 = \{u \in V | \text{there exists } v \neq u \text{ such that } u \simeq_1 v\}$
- $V_2 = \{u \in V | \text{there exists } v \neq u \text{ such that } u \simeq_2 v\}$
- $V_3 = V \setminus V_1 \setminus V_2$.

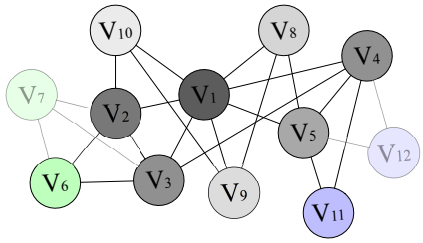


Figure 3: Equivalence Relation Reduction

Example 4.3. In Fig. 3, $V_1 = \{v_{11}, v_{12}\}$ since $N(v_{11}) = N(v_{12}) = \{v_4, v_5\}$; $V_2 = \{v_6, v_7\}$ since $\{N(v_6) \cup v_6\} = \{N(v_7) \cup v_7\} = \{v_2, v_3, v_6, v_7\}$.

Lemma 4.4. *V_1,V_2 and V_3 is a partition of the graph G.*

Proof. Since V_3 is the complement of $V_1 \cup V_2$, the three vertex sets jointly cover V. It remains to prove that $V_1 \cap V_2 = \emptyset$. Let u be a node $u \in V_1 \cap V_2$. According to the definition, there exist two nodes $v \neq u$ and $w \neq u$ such that $u \simeq_1 v$ and $u \simeq_2 w$. In other words, $N(u) = N(v)$ and $N(u) \cup \{u\} = N(w) \cup \{w\}$. Since v has no edge to u while w has an edge to u, $v \neq w$. Thus, $w \in N(u) = N(v)$, namely, there is an edge between w and v. Since $v \in N(w) \setminus \{u\} \subseteq N(u)$, u and v have an edge, contradiction. Therefore, $V_1 \cap V_2 = \emptyset$. □

According to Lemma 4.4, each node belongs to at most one non-trivial equivalence class constructed under the two equivalence relations. Therefore, we define the mapping function f that maps a node to the node with the smallest node ID in the corresponding non-trivial equivalent class.

Definition 4.5.

$$f(u) = \begin{cases} min\{v | v \simeq_1 u\}, \ if \ u \in V_1; \\ min\{v | v \simeq_2 u\}, \ if \ u \in V_2; \\ u, \ if \ u \in V_3; \end{cases} \quad (3)$$

Example 4.6. In Fig. 3, $f(v_{11}) = f(v_{12}) = v_{11}$; $f(v_6) = f(v_7) = v_6$; $f(u) = u$, for $u \in V_3$.

Graph Reduction. We compress the graph by eliminating all the nodes u in V_1 and V_2 and their incident edges unless $f(u) = u$. This operation transforms G to its subgraph G^s.

Example 4.7. In Fig. 3, $f(v_7) \neq v_7$, we delete v_7. Similarly, $f(v_{12}) \neq v_{12}$, we delete v_{12}. Nodes u with $f(u) = u$ are kept.

Lemma 4.8. *For any two nodes s, t with $f(s) \neq f(t)$, $dist_G(s, t) = dist_{G^s}(f(s), f(t))$.*

Proof. Let $p(s, t) = \{v_1, v_2, \cdots, v_k\}$ be a shortest path on G from s to t and let $p^s(s, t) = \{f(v_1), f(v_2), \cdots, f(v_k)\}$.

This paragraph proves that for any nodes x and y on p with $x \neq y$, $f(x) \neq f(y)$. We first show that for all $v \neq t$ on p, $f(v) \neq f(t)$: if otherwise the predecessor $pre(v)$ of v on the path $p - pre(v)$ exists since $f(s) \neq f(t)$ – can link to t directly and then reduces the path length, contradiction. Therefore, any node v with $f(v) \neq f(t)$ has a successor on p. Secondly, let $u \neq t$ be a node on p; denote by S the equivalent class of u; let z be the last node in S on the path. $suc(z)$, the successor of z on the path exists since $f(u) = f(z) \neq f(t)$ (from the first point). There is an edge from u to $suc(z)$ since 1) z has an edge to $suc(z)$, 2) $u, z \in S$ and 3) $suc(z) \notin S$. Thus, if $suc(z)$ is not the successor of u then p is not a shortest path. Therefore, all nodes on p have different $f(\cdot)$ values.

It is easy to verify that if $f(u) \neq f(v)$ and there is an edge between u and v, then there is an edge between $f(u)$ and $f(v)$. Thus, $p^s(s, t)$ is a path on G^s. Since G^S is a subgraph of G, $dist_G(s, t) \leq dist_{G^s}(s, t) \leq dist_G(s, t)$. □

Table 2: Reduce Index Size with Equivalence Relations

	Number of Reduced Nodes			Index Space (MB)	
Dataset	$\|V\|$	$\|V_1 \setminus F(V_1)\|$	$\|V_2 \setminus F(V_2)\|$	Before	After
YOUT	3,223,590	1,068,666	14,405	2141.512	1474.86
TPD	1,766,010	312,166	11,912	1783.192	1495.05

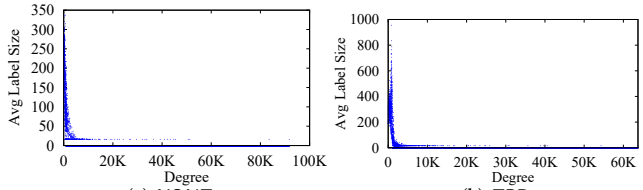
Example 4.9. Denote by $F(V') = \{v \in V' | v = f(v)\}$ the remained nodes in a set under equivalence reduction. Table 2 shows the effectiveness of the equivalence relations on index reduction. For YOUT (TPD), around 33.15% (17.67%) and 0.45% (0.67%) of nodes are eliminated by the first and the second equivalence relation, respectively and the index size are reduced by 31.13% (16.16%).

Query Processing. With the compressed graph, the query processing has to be adapted. We answer query $q(s,t)$ in the following four cases. 1) If $s = t$, return 0. 2) If $f(s) = f(t)$ under $s \simeq_1 t$ then return 2. 3) If $f(s) = f(t)$ under $s \simeq_2 t$, return 1. 4) Otherwise, return $q(f(s), f(t))$ in G^s.

4.2 Local Minimum Set Elimination

The index reducing technique in this section is motivated by an observation on the PLL label distribution.

For PLL with nodes ordered in node degrees, Fig. 4 shows the label size distribution of two representative small-world networks: Youtube (denoted by YOUT) is a social network and UK-Tpd (denoted by TPD) is a web graph. The maximum degrees of YOUT and TPD are 91751 and 63731, respectively. It can be observed that low degree nodes have significantly larger label sizes than the high degree nodes. This observation motivates the elimination of node labels on the nodes ranked lowest among its neighbors.



(a) YOUT (b) TPD

Figure 4: PLL: Degree and Label Size

Definition 4.10 (Local Minimum Set). A node is local minimum node if it has the lowest rank among its neighbors. Local minimum set constitutes of local minimum nodes:

$$\mathsf{M}(\mathrm{G}) = \{u \in V | \text{for } \forall v \in N(u), r(u) < r(v)\}.$$

Example 4.11. In Fig. 5, $\mathsf{M}(\mathrm{G}) = \{v_7, v_{10}, v_{11}, v_{12}\}$. For example, node v_7 has the lowest rank among its neighbors.

An ideal property of a local minimum node v is that v is referred to by no node other than v itself as a hub.

Lemma 4.12. *For any node $\forall v \in \mathsf{M}(\mathrm{G})$ and any node $\forall u \in V$, v is a hub of u in L^{PSL} if and only if $v = u$.*

Proof. According to Theorem 3.1, v is a hub of u if v is the highest ranked node in S — the set of all nodes on the shortest path from u to v. Unless $u = v$, for any shortest path from u to v, there is a node $w \in N(v)$ on the path. If v is a local minimum node, $r(v) < r(w)$ and v cannot be a hub of u. □

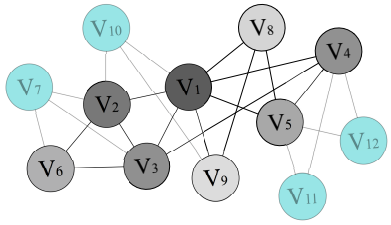


Figure 5: Local Minimum Set

Construct Labels for $V \setminus \mathsf{M}(\mathrm{G})$. Lemma 4.12 shows that removing nodes in $\mathsf{M}(\mathrm{G})$ does not affect the label set of any node in $V \setminus \mathsf{M}(\mathrm{G})$. However, in our propagation based label construction, $L_d^{\mathsf{PSL}}(v)$ is built from $L_{d-1}^{\mathsf{PSL}}(u)$, $\forall u \in N(v)$. In other words, for a node $u \in N(v) \cap \mathsf{M}(\mathrm{G})$, without $L_{d-1}^{\mathsf{PSL}}(u)$ we cannot construct $L_d^{\mathsf{PSL}}(u)$ using Theorem 3.18.

To tackle the above problem, the key finding is that nodes in $\mathsf{M}(\mathrm{G})$ are independent. That is, there is no edge between nodes in $\mathsf{M}(\mathrm{G})$. Thus, a node u with some of its neighbor from $\mathsf{M}(\mathrm{G})$ can be saved by resorting to u's two-hop neighbors via nodes in $\mathsf{M}(\mathrm{G})$. These 2-hop neighbors will certainly fall in $V \setminus \mathsf{M}(\mathrm{G})$ and their labels are ready for use.

Definition 4.13 (Generalized Neighbors). Given a node $v \in V \setminus \mathsf{M}(\mathrm{G})$, we define two neighbor sets. $N^1(v) = N(v) \setminus \mathsf{M}(\mathrm{G})$ includes the neighbors of v that fall in $V \setminus \mathsf{M}(\mathrm{G})$ and $N^2(v) = \{w | w \in (N(u) \setminus \{v\}), \forall u \in (N(v) \cap \mathsf{M}(\mathrm{G}))\}$ includes the two-hop neighbors of v connected via nodes in $\mathsf{M}(\mathrm{G})$.

Example 4.14. In Fig. 5, since $v_9 \in V \setminus \mathsf{M}(\mathrm{G})$, $N^1(v_9) = \{v_1, v_8\}$, $N^2(v_9) = \{v_1, v_2\}$.

We show that the generalized neighbors are not in $\mathsf{M}(\mathrm{G})$.

Lemma 4.15. *Given a node $v \in V \setminus \mathsf{M}(\mathrm{G})$, $N^1(v) \cap \mathsf{M}(\mathrm{G}) = \emptyset$ and $N^2(v) \cap \mathsf{M}(\mathrm{G}) = \emptyset$.*

Proof. $N^1(v) \cap \mathsf{M}(\mathrm{G}) = \emptyset$ by Definition 4.13. Let $x \in N^2(v)$ be a node expanded from $y \in N(v) \cap \mathsf{M}(\mathrm{G})$. If $x \in \mathsf{M}(\mathrm{G})$, then $r(y) < r(x)$ and $r(x) < r(y)$, contradiction. □

Example 4.16. In Fig. 5, $N^2(v_9) = \{v_1, v_2\}$, which are all in the set $V \setminus \mathsf{M}(\mathrm{G})$.

We show a label propagation function on $V \setminus M(G)$ below.

For $\forall v \in V$ and $d \in [1, D+1]$, denote, by $C_d(v)$, the set of hub nodes in label set $L_d^{\mathsf{PSL}}(v)$.

Theorem 4.17. *For each node $u \in V \setminus \mathsf{M}(\mathrm{G})$*

$$L_d^{\mathsf{PSL}}(u) = \bigcup_{\substack{w \in C_{d-1}(v), \text{ for } \forall v \in N^1(u) \\ w \in C_{d-2}(v'), \text{ for } \forall v' \in N^2(u)}} L_d^{\mathsf{PSL}}(u, w), \qquad (4)$$

$$where\ L_d^{\mathrm{PSL}}(u,w) =$$
$$\begin{cases} \emptyset,\ if\ r(w) < r(u)\ or\ Query(w,u,L_{<d}^{\mathrm{PSL}}) \le dist(w,u); \\ (w, dist(w,u)),\ otherwise. \end{cases}$$

Proof. Let L'' be the labels drawn from Equation (4). We reuse the proof of Theorem 3.18 by showing that the hubs L' constructed in Equation (1) is a subset of the hubs in L''. According to Lemma 3.15, $\bigcup_{v' \in N^2(u)} C_{d-2}(v')$ is a super set of $\bigcup_{v \in N(v) \cap \mathsf{M(G)}} C_{d-1}(v)$, besides, $N(u) = N^1(u) \cup (N(u) \cap \mathsf{M(G)})$, thus $\bigcup_{v \in N(u)} C_{d-1}(v) \subseteq \bigcup_{v \in N^1(u)} C_{d-1}(v) \cup \bigcup_{v' \in N^2(u)} C_{d-2}(v')$ which completes the proof. □

Table 3: Reduced Index Size with Local Minimum Set

	Node Number		Index Space (MB)	
Dataset	$\|V\|$	$\|\mathsf{M(G)}\|$	Before	After
YOUT	3,223,590	2,287,357	2141.512	1234.377
TPD	1,766,010	1,151,224	1783.192	989.567

Example 4.18. Table 3 shows the effectiveness on reducing the index size using local minimum set. For YOUT (TPD), the local minimum set eliminates about 70.95% (65.18%) nodes and reduces the index size by 42.4% (44.5%).

Query Processing. The reduced index provides the labels for nodes in $V \setminus \mathsf{M(G)}$. When it comes to query processing, we can recover the labels of nodes in $\mathsf{M(G)}$ with the union of the labels of neighbors. For a query $q(s,t)$, without loss of generality, if $s \in \mathsf{M(G)}$ and $t \in V \setminus \mathsf{M(G)}$, we swap s and t. To reduce the online cost, we use a hash join to produce the 2-hop distances. Let H be a table of size $|V \setminus \mathsf{M(G)}|$ where $H(w)$ records the labelled distance in $L^{\mathrm{PSL}}(s)$ with hub w. $H(w) = \infty$ if w is not a hub of s. Since the label set $L^{\mathrm{PSL}}(s)$ may not be available, we construct H in two cases.

- If $s \in V \setminus \mathsf{M(G)}$, we hash the labels in $L^{\mathrm{PSL}}(s)$ by letting $H(v) = dist(s,v)$ for each hub v of s.
- Otherwise, we construct labels of s by visiting neighbors $w \in N(s)$ of s and update $H(v)$ with $dist(v,w)+1$ for each hub v of $w - H(v)$ only keeps the minimum value along the updates.

After H being constructed, we generate labels of t in a similar way and instead of updating the table H, we fetch the value stored in the table H under the same hub node and then compose a 2-hop distance.

Note that, the hash table H can be maintained across different queries without initialization: we keep a dirty log and recover H after processing each query.

Lemma 4.19. *When $s,t \in \mathsf{M(G)}$, the time cost of distance query is $O(\Sigma_{a \in N(s)}|L^{\mathrm{PSL}}(a)| + \Sigma_{b \in N(t)}|L^{\mathrm{PSL}}(t)|)$.*

Proof. For s, we store nodes in $\{L^{\mathrm{PSL}}(a) | a \in N(s)\}$ in H. For t, we scan the nodes in $\{L^{\mathrm{PSL}}(b) | b \in N(t)\}$ to gain the distance. The linear scan takes $O(|\{L^{\mathrm{PSL}}(a)|a \in N(s)\}| + |\{L^{\mathrm{PSL}}(b)|b \in N(s)\}|)$ time in total. □

Table 4: Local Minimum Set: Index and Query Time

	Index Time (sec)		Query Time (sec)	
Dataset	Before	After	Before	After
YOUT	23.805	15.786	1.13E-06	1.71E-06
TPD	18.997	13.71	1.80E-06	3.71E-06

Example 4.20. Table 4 shows the index time and query time in a 45-core environment. Local minimum set technique reduces, for YOUT (TPD), the index time by 33.69% (27.83%) at a cost of 1.5× (2.06×) query time. The trade-off is worthwhile since the query time is still in micro-seconds.

5 RELATED WORK

Indexing shortest distances for fast online query processing has been extensively studied. A recent experimental comparison on distance labeling algorithms can be found in [17].

Distance Labeling on Small-world Networks. To index shortest distances for small-world networks, existing solutions either build a partial index to assist the online search algorithms [5, 11, 12] or build a complete index to fully support the distance query [3, 13]. The solutions in the latter category require a larger index but will obtain much faster query processing time. In the first category, Is-label approach first determines the vertex hierarchy through the independent set and then creates the label for each node by this hierarchy structure [11]. Tree decomposition is used in [5] to discover the core-fringe structure of social-networks and then index is created on these two separate parts. Shortest path trees of high degree nodes are used [12] as index to guide the online searching to process the distance query. In the second category, PLL [3] constructs the index by performing pruned BFS whose detail is given in Section 2.3. The hop doubling approach in [13] applies generation rules to join the short paths to long paths, until the whole paths are covered. Compared to PLL, the algorithm proposed in [13] uses less memory but will spend much more index time.

Distance Labeling on Road Networks. For distance indexing approaches on road networks, the approach in [1] constructs the index by eliminating the high ranking nodes and add it to the labels of its neighbors. The approach proposed by Wei [29] first decomposes the graph into a tree as the index, and then the distance of two nodes are answered through this index using dynamic programming. The pruned highway labeling approach proposed by Akiba et al. [2] decomposes the road network into disjoint paths and the label of a node include the distance to some nodes of the paths. A hierarchical hop-based index is proposed in [19] to answer

Table 5: The Description of the Datasets

Name	Dataset	n	m	Type
DELI	Delicious[10]	536,109	1,365,961	Social Network
GP	GPlus[10]	211,188	1,506,896	Social Network
LAST	Lastfm[10]	1,191,806	4,519,330	Social Network
GOOG	Google[11]	875,713	5,105,039	Web Graph
AMAZ	Amazon[7]	735,323	5,158,388	Social Network
DIGG	Digg[10]	770,800	5,907,132	Social Network
FLIX	Flixster[8]	2,523,386	7,918,801	Social Network
TREC	Trec[8]	1,601,787	8,063,026	Web Graph
YOUT	Youtube[8]	3,223,589	9,375,374	Social Network
SKIT	Skitter[8]	1,696,415	11,095,298	Internet Topology
TWIT	Twitter[11]	456,631	14,855,875	Social Network
HUDO	Hudong[10]	1,984,485	14,869,484	Web Graph
PET	Petster[8]	623,766	15,699,276	Social Network
BAID	Baidu[10]	2,141,301	17,794,839	Web Graph
TPD	UK-Tpd[7]	1,766,010	18,244,650	Web Graph
DBLP	DBLP[8]	1,314,050	18,986,618	Coauthorship
TOPC	Topcats[11]	1,791,489	28,511,807	Web Graph
POK	Pokec[11]	1,632,803	30,622,564	Social Network
FLIC	Flickr[8]	2,302,925	33,140,017	Social Network
HOST	UK-Host[7]	4,769,354	50,829,923	Web Graph
STAC	Stack[11]	6,024,271	63,497,050	Interaction
LJ	Ljournal[7]	5,363,260	79,023,142	Social Network
FB	Facebook[10]	58,790,783	92,208,195	Social Network
INDO	Indochina[7]	7,414,866	194,109,311	Web Graph
SINA	Sina[10]	58,655,850	261,321,071	Social Network
WIKI	Wiki[8]	12,150,976	378,142,420	Web Graph
ARAB	Arabic[7]	22,744,080	639,999,458	Web Graph
IT	IT-2004[7]	41,291,594	1,150,725,436	Web Graph
SK	SK-2005[7]	50,636,154	1,949,412,601	Web Graph
UK	UK-2006[7]	77,741,046	2,965,197,340	Web Graph

shortest distance queries in a road network with bounded query processing time and index size. More details about the distance query on road networks can be found in [17, 30].

Approximate Distance Labeling. For approximate distance labeling algorithms, the basic idea is to select nodes as landmarks and then precompute the distances from the landmarks to all the other nodes. The distance between any node pair can be estimated using triangle inequality [8, 20]. Online processing on landmarks is used to improve the precision [21, 27]. However, on small-world networks, the relative error becomes significant since the distances are bounded by the small diameter.

6 EXPERIMENTAL RESULTS

Algorithms. We compare our proposed algorithms against the state-of-the-art algorithm PLL [3]. Our techniques include the following three methods:

- PSL: the parallelized distance labeling technique introduced in Section 3.
- PSL$^+$: PSL with the equivalence relation elimination technique as introduced in Section 4.1.
- PSL*: PSL with the equivalence relation elimination technique plus the local minimal set elimination technique as introduced in Section 4.2.

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. We set the cut-off time as 24 hours.

Datasets. We conducted experiments on 30 real-world graphs whose properties are shown in Table 5. The largest graph has more than 2.9 billion edges. The datasets are from various types of small-world networks including social networks, web graphs, internet topology graphs, coauthorship graphs, and interaction networks. All graphs were downloaded from Network Repository[5][23], Stanford Large Network Dataset Collection[6][15], Laboratory for Web Algorithms[7] [6, 7], and the Koblenz Network Collection[8] [14].

Exp 1: Index Time on a Single Core. We compare the index time of PLL with PSL, PSL$^+$ and PSL* on a single core. Note that, the bit-parallel technique introduced in [3] is used for all methods since it is a separate optimization which can be plugged into all distance labeling methods.

Fig. 6 shows that PSL has an index time comparable to PLL while PSL$^+$ and PSL* reduce the index time of PLL—a by-product of the index reduction. For example, on the dataset ARAB, PSL$^+$ and PSL* successfully constructed the index while PLL and PSL failed.

Exp 2: Index Time on Multiple Cores. Fig. 7 shows the index time of PSL, PSL$^+$ and PSL* on 45 cores. Compared to the single-core results shown in Fig. 6, all the three methods have a significant speedup. This speedup allows PSL to index multiple massive graphs, e.g., LJ, ARAB and SK, that cannot be indexed on a single core. PSL* succeeded in indexing all the graphs while both PSL and PSL$^+$ failed on FB and UK—thanks to the index reduction. The results show that the parallelism together with the index reduction techniques scale up the distance labeling to handle larger graphs.

Exp 3: Index Size. Fig. 8 shows the index size of PLL, PSL, PSL$^+$ and PSL*. The label size of PLL and PSL is the same, which conforms to the analysis in Section 3.3. Both index reduction techniques are effective. PSL$^+$ reduces the index size of PSL on SK by more than 50%. Moreover, only PSL* can

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

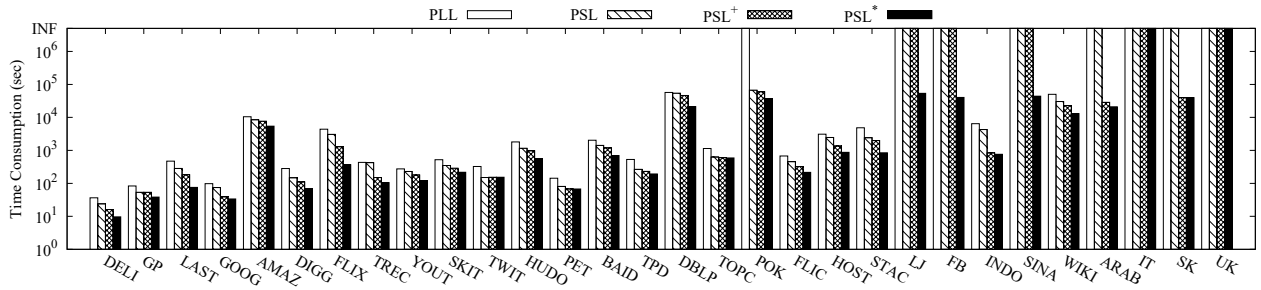


Figure 6: The Comparison of the Index Time on One Core

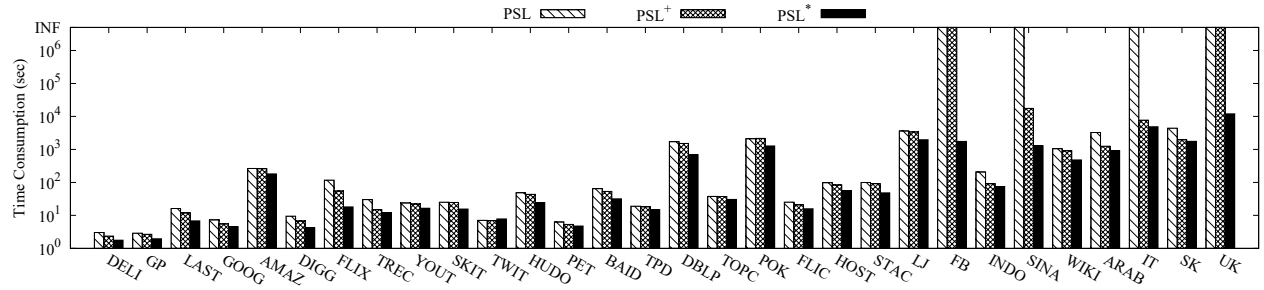


Figure 7: The Comparison of the Index Time on 45 Cores

index massive graphs such as UK while the other approaches ran out of memory. This verified the effectiveness of our index reduction approaches.

Exp 4: Query Time. We compare the average query time of PSL, PSL$^+$ and PSL* on 10^6 random queries. Fig. 9 shows that PSL$^+$ and PSL* have a query time comparable to PSL. For PSL$^+$, the additional query cost on checking equivalence relations is negligible. Since G^s is smaller than G, the query time of PSL$^+$ is sometimes smaller than PSL. For example, the query time of PSL$^+$ on DELI is 1.17E-6 seconds while the query time of PSL is 1.31E-6 seconds. For PSL*, although the labels of nodes in M(G) need to be constructed on-the-fly, the query time of PSL* is within twice the query time of PSL on average, remaining in micro-second level.

Exp 5: Indexing Speedup on Multi Cores. The speedup of the index time of an approach on x cores is calculated by

$$speedup = \frac{\text{the index time of the approach with 1 core}}{\text{the index time of the approach with } x \text{ cores}}. \quad (5)$$

According to Equation 5, when the core number is 1, the speedup is constantly 1; when an approach fails in indexing on 1 core within the time limit, its speedup cannot be derived. Fig. 10 shows the index time speedup of PSL, PSL$^+$ and PSL* with the core number varying from 1, 12, 23, 34, to 45 on six networks, DBLP, POK, LJ, FB, WIKI, and SK, respectively. A near linear speedup has been observed for all the three approaches along with the increasing number of cores. The speedup of each approach is relatively stable over different graphs. On 45 cores, PSL shows, over all datasets, an average speedup of 30 and a maximum speedup of 32, PSL$^+$ shows average 28 and maximum 31 while PSL* shows average 27 and maximum 31. The index reduction techniques have little influence on the speedup: the lines of the three approaches clutter, especially on DBLP. A mild slowdown in the speedup when the core number gets close to 45 can be explained by the imbalance resource allocation introduced by more cores.

The index size reduction techniques can be critical: PSL failed on FB even when 45 cores were engaged while PSL* removed redundant nodes to achieve an completion.

Exp 6: Scalability on Index Time. We randomly divided the nodes of a graph into 5 groups, each group consisted of 1/5 of the nodes. We created 5 graphs while the i-th test case is the induced subgraph on the first i node groups. The experiments were performed on the 5 graphs, respectively.

Fig. 11 shows that the index time of PSL* increases almost linearly with the number of nodes of the graph. For example, the index time is about 48 times on 100% nodes than on 20% nodes of DBLP and is about 8 times for FB. For PSL and PSL$^+$, although there is a situation where these two methods fail to create the index, the index time increases smoothly when the number of nodes increases. Therefore, the above results justify the scalability of PSL for index time.

Exp 7: Scalability on Index Size. The setting is the same as the former experiment. Fig. 12 shows that the space consumption grows smoothly with the graph size for all three methods. For example, the index space on 100% nodes of DBLP is about 184.6, 251.2, 182.5 times larger than that on

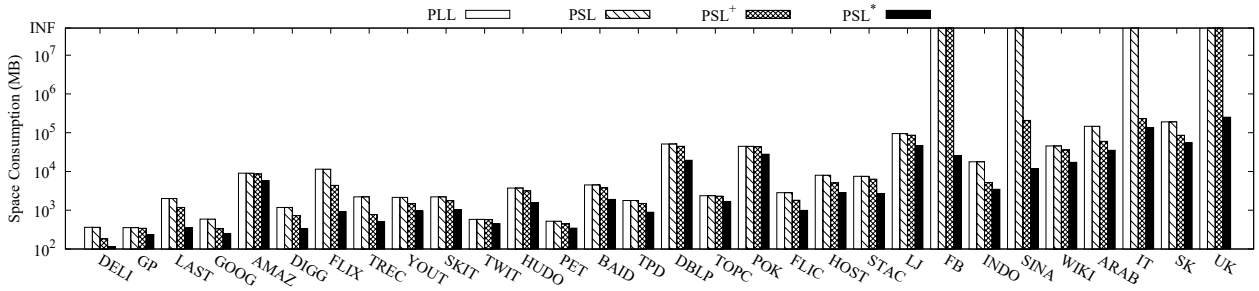


Figure 8: The Comparison of the Index Size

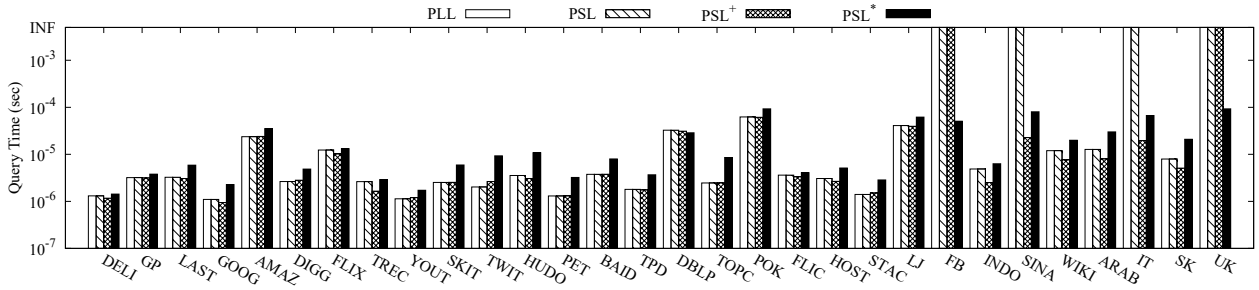


Figure 9: The Comparison of the Query Time

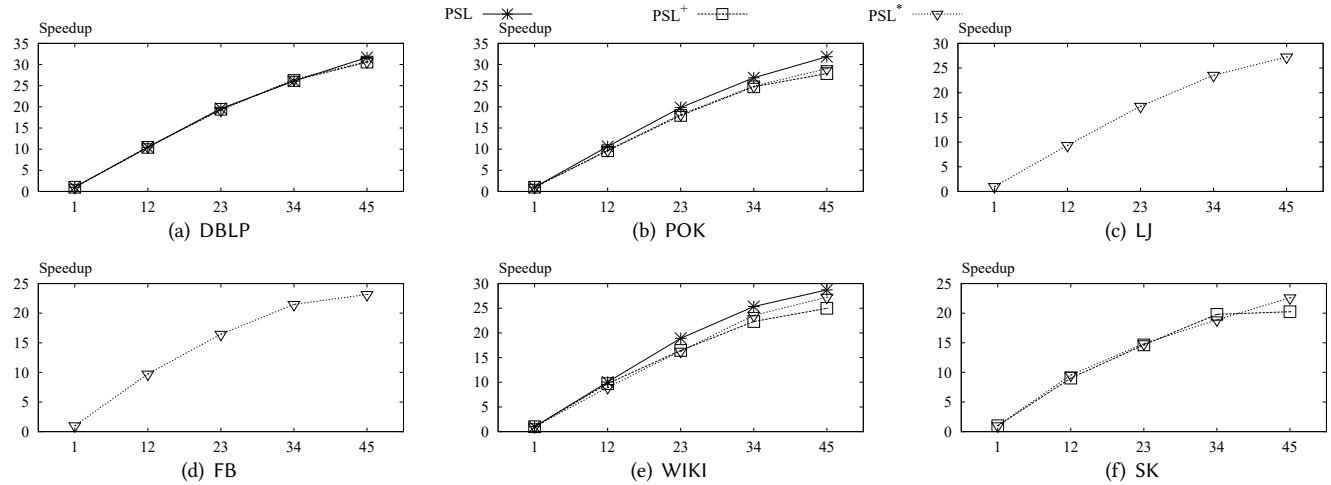


Figure 10: The Effect of Core Number on the Index Time

20% nodes for PSL, PSL$^+$, and PSL* respectively. Therefore, the smooth increase of the index space shows the scalability of PSL for the index size.

Exp 8: Scalability on Query Time. Fig. 13 shows that the query time of the proposed approaches grows smoothly with the graph size. For example, on LJ, the query time on 100% nodes is about 368.34, 372.92 and 546.26 times larger than that on 20% nodes for PSL, PSL$^+$, and PSL* respectively. Other graphs show a similar trend. Combining the above experiments on the scalability test, we draw the conclusion that the proposed methods all show excellent scalability.

For more experiments please see Appendix B.

7 CONCLUSIONS

In this paper, we propose a novel parallelized labeling scheme for distance queries on small-world networks. Our method accelerates the index construction by concurrently creating labels with the same label distances. Moreover, the index size is further reduced by removing redundant nodes from the graph and removing labels of local minimum sets from the index. Extensive experimental results illustrate the superior efficiency of our approach. In particular, our approach enables the building of the index for networks at billion scales. Experimental results verify the near-linear speedup of our algorithms in a multi-core environment.

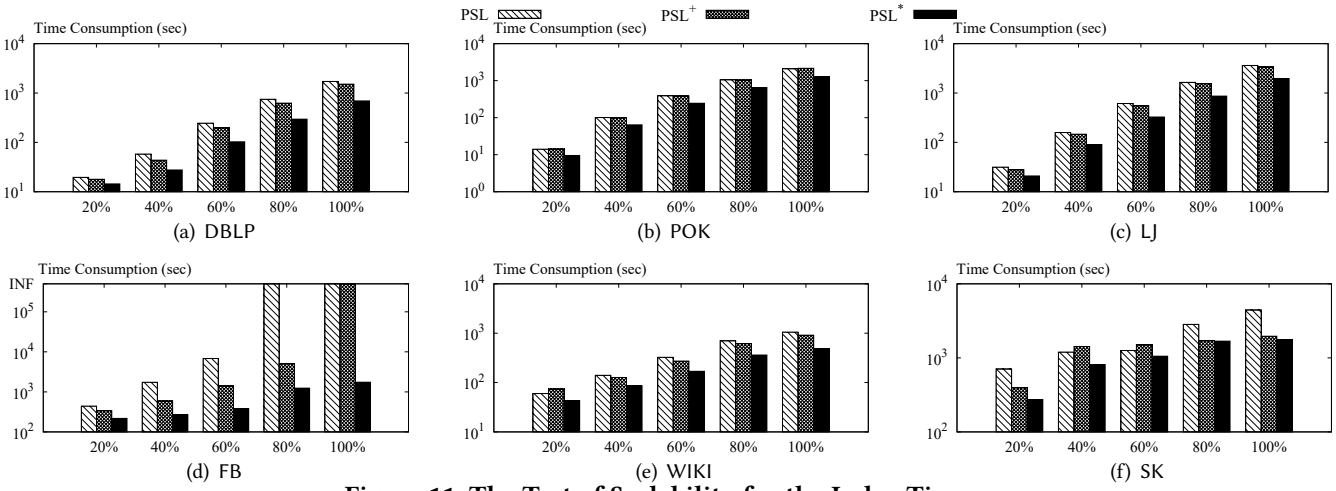


Figure 11: The Test of Scalability for the Index Time

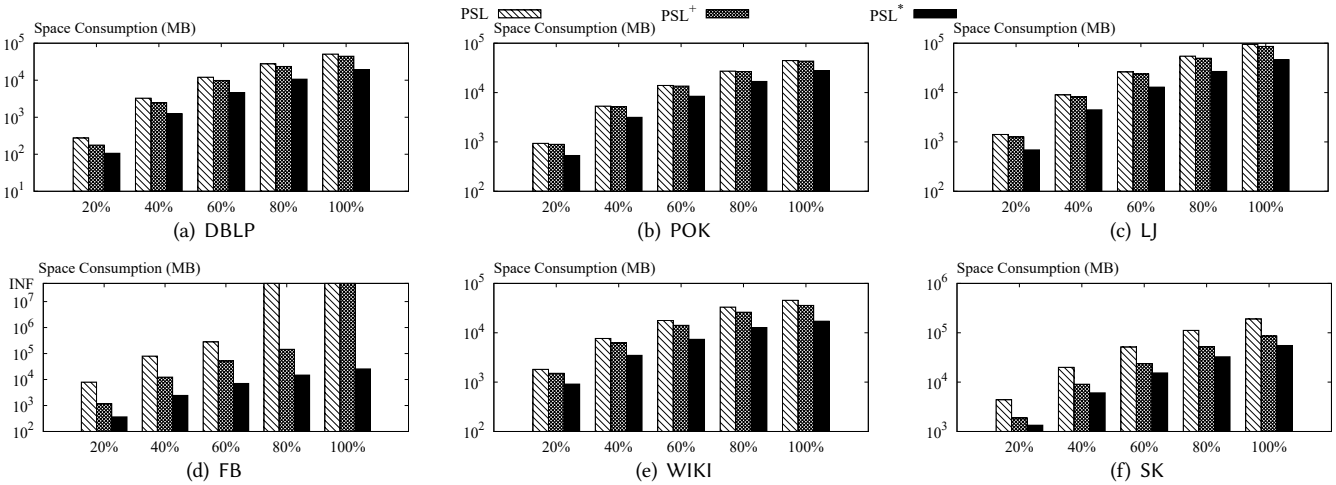


Figure 12: The Test of Scalability for the Index Size

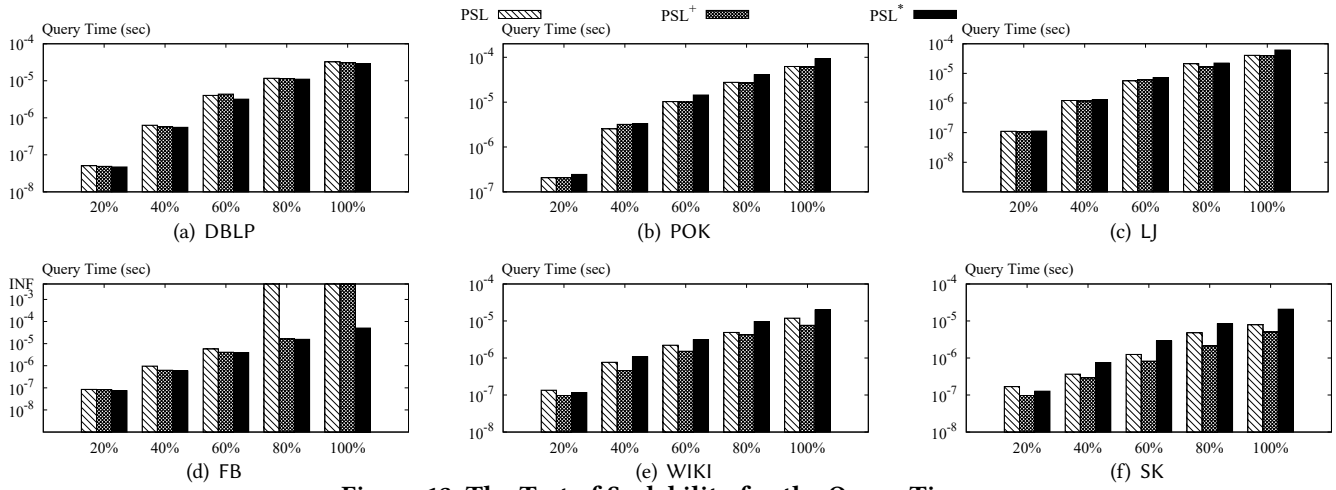


Figure 13: The Test of Scalability for the Query Time

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 FT170100128 and DP180103096. Lijun Chang is supported by ARC DP160101513 and FT180100256. Xuemin Lin is supported by NSFC 61672235, DP170101628 and DP180103096.

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2012. Hierarchical hub labelings for shortest paths. In *European Symposium on Algorithms*. Springer, 24–35.

[2] Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. 2014. Fast shortest-path distance queries on road networks by pruned highway labeling. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 147–154.

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

[4] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2014. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *Proceedings of the 23rd international conference on World wide web*. ACM, 237–248.

[5] Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. 2012. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 144–155.

[6] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th international conference on World Wide Web*, Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM Press, 587–596.

[7] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. ACM Press, Manhattan, USA, 595–601.

[8] Wei Chen, Christian Sommer, Shang-Hua Teng, and Yajun Wang. 2012. A compact routing scheme and approximate distance oracle for power-law graphs. *ACM Transactions on Algorithms (TALG)* 9, 1 (2012), 4.

[9] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2002. Reachability and distance queries via 2-hop labels. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 937–946.

[10] Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2013. Hub label compression. In *International Symposium on Experimental Algorithms*. Springer, 18–29.

[11] 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. *Proceedings of the VLDB Endowment* 6, 6 (2013), 457–468.

[12] Takanori Hayashi, Takuya Akiba, and Ken-ichi Kawarabayashi. 2016. Fully dynamic shortest-path distance query acceleration on massive networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 1533–1542.

[13] 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. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1203–1214.

[14] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1343–1350.

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

[16] Jianxin Li, Xinjue Wang, Ke Deng, Xiaochun Yang, Timos Sellis, and Jeffrey Xu Yu. 2017. Most influential community search over large social networks. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 871–882.

[17] Ye Li, Man Lung Yiu, Ngai Meng Kou, et al. 2017. An experimental study on hub labeling based shortest path algorithms. *Proceedings of the VLDB Endowment* 11, 4 (2017), 445–457.

[18] Mark EJ Newman. 2005. A measure of betweenness centrality based on random walks. *Social networks* 27, 1 (2005), 39–54.

[19] 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 *Proceedings of the 2018 International Conference on Management of Data*. ACM, 709–724.

[20] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. 2009. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 867–876.

[21] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. 2014. Approximate shortest distance computing: A query-dependent local landmark scheme. *IEEE Transactions on Knowledge and Data Engineering* 26, 1 (2014), 55–68.

[22] Yongrui Qin, Quan Z Sheng, Nickolas JG Falkner, Lina Yao, and Simon Parkinson. 2017. Efficient computation of distance labeling for decremental updates in large dynamic graphs. *World Wide Web* 20, 5 (2017), 915–937.

[23] Ryan A. Rossi and Nesreen 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

[24] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. 2014. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1176–1185.

[25] Chih-Ya Shen, Liang-Hao Huang, De-Nian Yang, Hong-Han Shuai, Wang-Chien Lee, and Ming-Syan Chen. 2017. On finding socially tenuous groups for online social networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 415–424.

[26] Jeffrey Travers and Stanley Milgram. 1967. The small world problem. *Phychology Today* 1, 1 (1967), 61–67.

[27] Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. 2011. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 1785–1794.

[28] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *nature* 393, 6684 (1998), 440.

[29] Fang Wei. 2010. TEDI: efficient shortest path query answering on graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 99–110.

[30] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest path and distance queries on road networks: An experimental evaluation. *Proceedings of the VLDB Endowment* 5, 5 (2012), 406–417.

A PROOF OF LEMMA 2.3

According to triangle inequality, for any node $u \in V$, $dist(s,u) + dist(u,t) \geq dist(s,t)$. For a node u' on a shortest path from s to t, $dist(s,t) = dist(s,u') + dist(u',t)$. Since $C(s) \cap C(t)$ shares a node with a shortest path from s to t, $\min_{v \in C(s) \cap C(t)} dist(s,v) + dist(v,t) = dist(s,t)$.

B ADDITIONAL EXPERIMENTS

Exp 9: The Impact of Node Order. Theorem 3.1 shows that the index structure of PSL and PLL is determined by the node order. Apart from the degree-based node order engaged by Exp 1-8 in Section 6, a popular node order is based on the betweenness centrality of each node u — the fraction of shortest paths between node pairs that pass through u [18]. Paper [17] also mentioned a significant-path-based node order generated from an iterative process. For each $i \in [1,n]$, the i-th iteration has the following steps.

(1) C_i denotes the candidate set. $C_1 = V$ is the node set of the graph. For $i \in [2,n]$, C_i is determined in the $(i-1)$-th round. S_i denotes the set of previously selected significant nodes where $S_1 = \emptyset$.
(2) r_i, the significant node, is the highest-degree node in C_i. Denote by $T(r_i)$ the shortest path tree rooted at r_i.
(3) Compute the trimmed shortest path tree T_i, the largest subtree of $T(r_i)$ rooted at r_i without a node in S_i.
(4) Compute p_i, the root to leaf path in T_i where each node v in p_i is the maximum-degree child in T_i of $prec(v, p_i)$ — the predecessor of v in p_i;
(5) If p_i has only one node, then let C_{i+1} be $V \setminus S_{i+1}$, otherwise, let C_{i+1} be the nodes in p_i with r_i excluded.

The iteration terminates in $n = |V|$ rounds. The significant-path-based node order is $r_1, r_2, \cdots, r_n$.

This experiment computes the three node orders:

D Degree-based node order,
B Betweenness-centrality-based node order, and
S Significant-path-based node order,

using existed source code[9] and then compares the performance of our proposed approaches, PSL, PSL$^+$ and PSL* under the three node orders. PSL$_D$ denotes PSL under node order D and this notation similarly applies to other approaches and node orders. Note that it is necessary to list the computation time of a node order — the betweenness centrality of nodes in V and the significant path of the graph requires heavy computation.

Table. 6 shows the index time (with the node order computation time OT), index size, and query time of PSL, PSL$^+$, and PSL* under different node orders D, B, and S, respectively, on four graphs DELI, GP, LAST, GOOG. The node order of a bold number wins the corresponding comparison.

[9] http://degroup.cis.umac.mo/sspexp

In terms of the total index time — the summation of the index time and node order computation time — under one core, node order D wins over all the comparisons. This attributes to the fast computation time of the node order D. Node order B allows smaller index time at a cost of a far more expensive node order computation. The computation of node order S is on average 40% cheaper than that of node order B but is still far more expensive than that of node order D.

Node order D produces index with sizes comparable and even lower than that of node orders B and S. Though for PSL, the index size under node order B is on average 37% smaller than that under node order D; for PSL$^+$ and PSL*, node order D wins the comparisons on, respectively, 2 out of 4 and 3 out of 4 datasets, which is surprising in considering its low computation cost.

The wining node order (typically node orders B and S) in query time spends, compared to node order D on average, 29% less query time. For PSL*, node order D wins the query time on 2 out of 4 datasets.

The degree-based node order has a comparable or even smaller index size and query time than that of betweenness-centrality-based and significant-path-based node orders. Betweenness-centrality-based node order shows better index size and query time but is expensive to compute.

Exp 10: Comparison with other Index Reduction Techniques. This experiment compares the two index reduction techniques proposed in Section 4 (PSL$^+$ utilises the first reduction technique while PSL* applies both) to the existing index reduction technique HLC [10]. HLC compresses the index by coding the common labels into reusable tokens while restoring the labels in query time. The code of HLC was from the authors of [17].

The performance gained by applying an index reduction technique can be captured by the ratio of the costs (index time, index size, or query time) before and after the technique is applied. A ratio greater than 1 if and only if the index reduction technique is reducing the cost.

Fig. 14 shows the index time ratio, index size ratio and query time ratio of the three index reduction techniques on four datasets DELI, GP, LAST and GOOG.

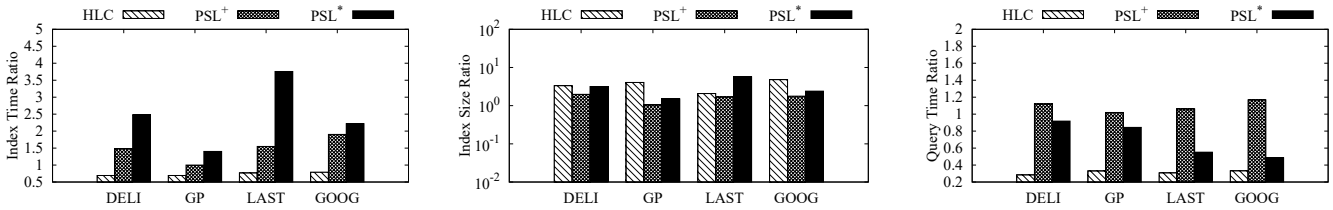
The index time ratio of HLC is constantly smaller than 1: HLC pays 36% more time than PLL in compressing the index of PLL. In contrast, the index time of PSL$^+$ (and PSL*) is 64% (and 75%) less than that of the baseline.

HLC, PSL$^+$ and PSL* can all reduce the index size; however, the index reduction of HLC is not for free — the query time of HLC becomes much longer (ratio much smaller than 1). In contrast, PSL$^+$ reduces the query time of the baseline.

In conclusion, HLC achieves smaller index size at a cost of a much longer index and query time while PSL$^+$ reduces the

Table 6: The Effect of Node Order on Index Time (IT), Index Size (IS), and Query Time (QT)

	Dataset	PSL			PSL$^+$			PSL*		
		PSL$_D$(OT)	PSL$_B$(OT)	PSL$_S$(OT)	PSL^+_D(OT)	PSL^+_B(OT)	PSL^+_S(OT)	PSL$^*_D$(OT)	PSL$^*_B$(OT)	PSL$^*_S$(OT)
IT (sec)	DELI	**23.9(0.1)**	20.9(173.4)	24.6(81.5)	**16.1(0.1)**	17.7(173.4)	19.5(81.5)	**9.6(0.1)**	13.1(173.4)	12.2(81.5)
	GP	**53.8(0.1)**	22.2(118.8)	23.6(62.3)	**53.8(0.1)**	21.0(118.8)	21.5(62.3)	**38.4(0.1)**	24.6(118.8)	20.9(62.3)
	LAST	**282.6(0.3)**	228.9(1481)	287.4(1040)	**182.6(0.3)**	162.8(1481)	209.2(1040)	**75.2(0.3)**	86.9(1481)	85.0(1040)
	GOOG	**74.5(0.2)**	34.7(123.4)	39.2(87.6)	**39.2(0.2)**	26.7(123.4)	33.1(87.6)	**33.6(0.2)**	23.9(123.4)	26.9(87.6)
IS (MB)	DELI	364.0	**319.4**	338.9	**184.0**	244.4	255.9	**114.5**	180.8	183.7
	GP	355.2	**143.1**	158.8	341.8	**141.0**	156.3	235.4	**107.5**	123.9
	LAST	1997.5	**1778.7**	1958.8	**1182.1**	1242.8	1343.6	**351.8**	461.3	473.8
	GOOG	589.1	**366.1**	399.9	331.6	**324.0**	343.0	**245.0**	283.3	291.5
QT (sec)	DELI	1.3E-06	**1.1E-06**	1.2E-06	1.2E-06	**1.0E-06**	1.1E-06	**1.4E-06**	1.6E-06	1.5E-06
	GP	3.2E-06	1.4E-06	**1.3E-06**	3.1E-06	**1.2E-06**	1.3E-06	3.8E-06	2.6E-06	**2.0E-06**
	LAST	3.3E-06	**2.9E-06**	3.2E-06	3.1E-06	**2.8E-06**	3.1E-06	**5.9E-06**	6.4E-06	6.5E-06
	GOOG	1.1E-06	**7.4E-07**	8.6E-07	9.4E-07	**7.7E-07**	8.2E-07	2.3E-06	**1.6E-06**	1.6E-06



(a) Index Time Ratio (b) Index Size Ratio (c) Query Time Ratio

Figure 14: The Comparison of Size Reduction Techniques

Table 7: The Description of the Road Networks

Name	Dataset	n	m	D
BO	road-belgium-osm[10]	1,441,295	1,549,970	1987
CA	roadNet-CA[11]	1,971,281	2,766,607	865
PA	roadNet-PA[11]	1,090,920	1,541,898	794
TX	roadNet-TX[11]	1,393,383	1,921,660	1064

index size, index time and query time simultaneously. PSL* reduces the index time and index size at cost of a moderate increase on the query time.

Exp 11: Performance on Road Networks. We evaluated our approaches on graphs with large diameters. A majority of real graphs with large diameters are weighted, we thus downloaded 4 Road Networks (RN) from Network Repository[10] and Stanford Large Network Dataset Collection[11]and then turned them into unweighted graphs by letting the weight of each edge be 1. Table 7 shows the details of the four RNs.

We performed PSL, PSL$^+$ and PSL* on the four road networks. The index time, index space and query time on 1 core are shown in Fig. 15. PSL$^+$ performed identically with PSL since on road networks, it is hard to find non-trivial equivalent node classes. In contrast, PSL* can still reduce the index time and index size of PSL at a cost of a slight increase of query time. On 45 cores, the speedup of the three approaches stays stably within 16-21. This means that our parallelization also works for graphs with large diameters.

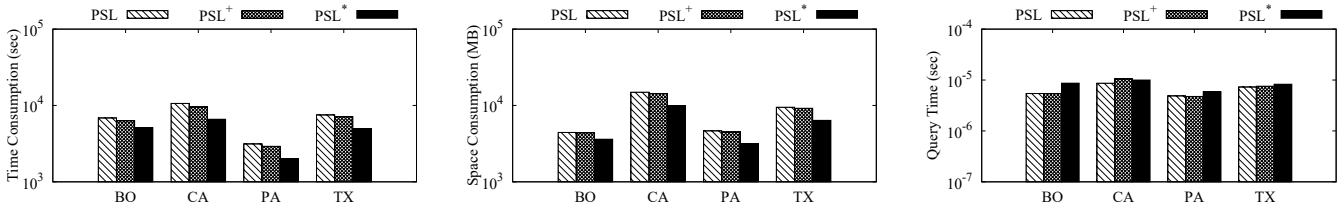
We carried out the state-of-the-art road network distance labeling approach, the hierarchical 2-hop labeling method (H2H) [19]. H2H makes full use of the graph structure of road networks to achieve the superior efficiency in index time and query time by combining the 2-hop labeling with the additional node hierarchy. We obtained the source code from the authors of [19] and adopted default parameters.

Fig. 16 compares the performance of H2H with that of PSL-1 and PSL-45, the PSL on one core and 45 cores, respectively. PSL-1 takes up to two orders of magnitude longer index time than H2H and one order of magnitude longer query time. The index size of PSL is also larger than that of H2H. Even equipped with 45 cores, the index time of PSL-45 is still slower than H2H on 2 out of 4 datasets. This result echoes the fact that PLL was not designed for road networks.

C EXTEND PSL TO DIRECTED GRAPHS

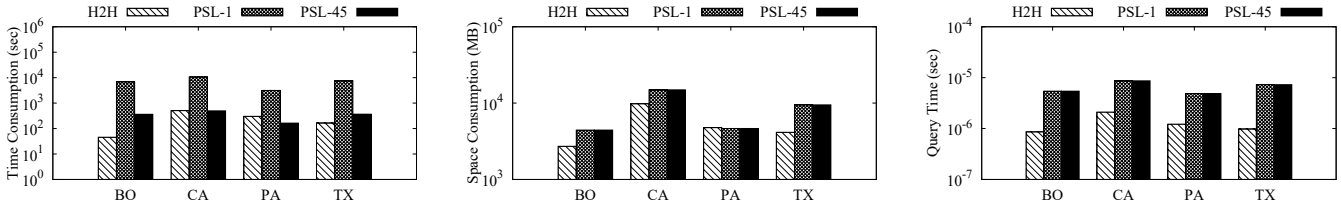
For directed graphs, each node $v \in V$ is associated with a set of hub nodes $C_{\mathsf{IN}}(v)$, where $w \in C_{\mathsf{IN}}(v)$ can reach v and another set of hub nodes $C_{\mathsf{OUT}}(v)$, where v can reach $w \in C_{\mathsf{OUT}}(v)$. Combined with the distance, we obtain two labels $L_{\mathsf{IN}}(v) = \{(u, dist(u, v))|u \in C_{\mathsf{IN}}(v)\}$ and $L_{\mathsf{OUT}}(v) = \{(u, dist(v, u))|u \in C_{\mathsf{OUT}}(v)\}$ for the node v. To compute the labels $L_{\mathsf{OUT}}(v)$, we run PSL on G; To compute $L_{\mathsf{IN}}(v)$, we reverse the edge direction of graph and run PSL on the

[10]http://networkrepository.com/index.php [23]
[11]http://snap.stanford.edu/data/ [15]



(a) Index Time (b) Index Space (c) Query Time

Figure 15: Performance of PSL-Based Approaches on Road Networks



(a) Index Time (b) Index Space (c) Query Time

Figure 16: Compare PSL to the State-Of-The-Art Road Network Labeling Approach

reversed graph. To process the distance query $q(s,t)$, we make use of $Query(s,t,L)$ defined in the following equation.

$$Query(s,t,L) = min_{u \in C_{\text{OUT}}(s) \cap C_{\text{IN}}(t)}(dist(s,u) + dist(u,t)).$$

D DISCUSSIONS

Empower PLL with two index reduction techniques. PLL adopts the equivalence relation reduction technique by initializing with a graph reduction: map each node to the smallest node in the non-trivial equivalent class (Definition 4.5) and then remove the nodes with $f(u) \neq u$. According to Theorem 4.17, PLL on the reduced graph is identical to PSL on the reduced graph which is the index of PSL$^+$.

PLL adopts the local minimum set reduction by adapting the graph $G(V,E)$ to $G'(V',E')$. Specifically, let M(G) be the local minimal set of G. For each node $v \in$ M(G), let $E_v = \{(u,w) \in E | u,w \in N(v)\}$. The new graph G' has $V' = V \setminus S$ and $E' = E \cup \bigcup_{v \in \text{M(G)}} E_v$. We assign to each newly added edge a weight of 2 and perform PLL on G' under the original node order. It can be easily proved that for any two nodes $u, v \notin$ M(G), their shortest distance on G' is the same with that on G. According to Theorem 3.1, u is a hub of v on G' if and only if u is a hub of v on G. Besides, according to the local minimality, nodes in M(G) do not contribute to any label of other nodes on G, thus, the PLL labels of a node v on G' is identical to that of v on G. The index of PLL on G', therefore, is identical to the index of PSL with local minimum set technique on G. The query processing in Section 4.2 can be directly applied. Since the degree of a local minimal node is not larger than that of its neighbors, the size of G' can be well bounded. For example, on GOOG, $|E'| = 2.29|E|$, and the number of $|E'|$ is no more than $3|E|$ over all the datasets in Table 5.

Extend PSL to Dynamic Graphs. Extending PLL to dynamic graphs has been studied in the literature and the existing results show that maintaining the PLL index for both edge insertion and edge deletion is challenging. For the distance label maintenance under edge insertion, existing technique [4] fails in eliminating the outdated labels and thus breaks the minimal property of PLL. For the distance label maintenance under edge deletion, existing approach [22] needs to conduct BFS from affected nodes and the speedup over the baseline of recomputing all the labels is thus marginal (up to one order of magnitude). Due to the challenges to maintain the PLL index, it is non-trivial to extend PSL to handle dynamic graphs using multiple cores. We will further explore how to parallelize PLL to handle dynamic graphs as our future work.

Extend PSL to Weighted Graphs. To handle weighted graphs, we need to modify the pruning condition in Lemma 3.17 as follows:

LEMMA D.1. *A hub w in the label set $\bigcup_{v \in N(u)} L_{d-1}^{\text{PSL}}(v)$ is not a hub of u in $L_d^{\text{PSL}}(u)$ if $Query(w,u,L_{<d}^{\text{PSL}}) \leq \min_{x \in L_{d-1}^{\text{PSL}}(u) \cap N(w)} dist(w,x) + length(Label_{d-1}^{\text{PSL}}(u,x))$.*

Since $x \in L_{d-1}^{\text{PSL}}(u)$, there is a label of u with $d-1$ hops from x. Here its weighted length recorded by the label is denoted as $length(Label_{d-1}^{\text{PSL}}(u,x))$.

Since the labels are generated based on the increasing order of number of hops other than the weighted distances, it is possible that a label with more hops may be shorter than the label with fewer hops. Consequently, some labels that can be pruned by PLL may not be pruned by PSL. As a result, PSL will produce a super set of labels of PLL although PSL still guarantees the correctness of query processing. It remains hard to trim the labels efficiently.