Noname manuscript No.
(will be inserted by the editor)

Distance Labeling: on Parallelism, Compression, and Ordering

Wentao Li[1] · **Miao Qiao**[2] · **Lu Qin**[1] · **Ying Zhang**[1] · **Lijun Chang**[3] · **Xuemin Lin**[4]

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 first scales distance labeling on small-world networks by proposing a Parallel Shortest-distance Labeling (PSL) scheme. 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. To further scale up PSL, it is critical to reduce the index size. This paper proposes effective index compression techniques based on graph properties as well as label properties; it also explores best practices in using betweenness-based node order to reduce the index size. The efficient betweenness estimation of the graph nodes proposed may be of independent interest to graph practitioners. Extensive experimental results verify our efficiency on billion-scale graphs, near-linear speedup in a multi-core environment, and up to 94% reduction in the index size.

Keywords Shortest Distance; 2-hop Labeling; Betweenness; Parallelism; Compression; Ordering

{wentao.li, lu.qin, ying.zhang}@uts.edu.au
miao.qiao@auckland.ac.nz
lijun.chang@sydney.edu.au
lxue@cse.unsw.edu.au
[1]AAII, FEIT, University of Technology Sydney, Australia
[2]University of Auckland, New Zealand
[3]The University of Sydney, Australia
[4]The University of New South Wales, Australia

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 [41], influential community searching [29] and event detection [40]. Many of these applications cannot afford frequent online distance computations, and therefore, 2-hop labeling [17] 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 [17]. 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 — 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 [33] as an entrance). For a major category of real graphs, small-world [43,45] networks, the state-of-the-art approach is Pruned Landmark Labeling (PLL) [4].

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 parallelize 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 two compression techniques to reduce the index size. The first one is based on graph properties and is thus applicable to all 2-hop labeling approaches; the second one explores the property of PSL, which leads to significant index reduction.
– We further explore best practices in using betweenness-based node order to reduce the index size. Given the quadratic time (infeasible for big graphs) in computing exact betweenness, we introduce k-betweenness — betweenness on paths with no more than k hops — to allow i) an efficient sampling-based approximation and ii) a holistic optimization of the node order for index reduction. The novel and efficient sampling-based approximate computation of node betweenness is the key to this reduction and may be of independent interest.
– 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 ap-

[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.

[2] The centrality can be defined with degree, closeness and betweenness [31].

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

[4] http://law.di.unimi.it

proach on small-world networks. Section 3 devises a distance labeling algorithm. Section 4 introduces two index reduction techniques. Section 5 computes the betweenness-based node order by proposing novel approximation algorithms, which further reduced the index size. Section 6 summarizes related works. Section 7 experimentally evaluates our proposed approaches on real small-world networks and Section 8 concludes the paper.

2 Preliminary

2.1 Shortest Distance Problem

Let G be an unweighted 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 $N(v) = \{u|(u,v) \in E_G\}$ the **neighbors** of v and $deg(v) = |N(v)|$ the degree of node v in G. We mainly use undirected graphs in the paper; Appendix B 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 $\forall 1 \le i \le k$, 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 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.

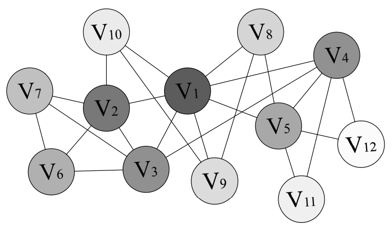


Fig. 1 Graph G

Example 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 [17] 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 [4,20,26] 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 1 (2-hop Cover Constraint [17]) 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 1 *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 Pruned Landmark Labeling Approach

Pruned 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 *total order* r on V, called the node order. For two nodes u and v, if $r(u) > r(v)$, we say u has a higher rank than v. With the node order defined, we can safely rename the nodes such that

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

A highly prevalent node order is degree-based: the order r prioritizes nodes with higher degrees and breaks ties based on original node ID. Specifically, for any two nodes v and v', $r(v) > r(v')$ if

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

This paper uses degree-based node order by default unless another node order is specified in the context.

Example 2 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 reorder the nodes such that $r(v_1) > r(v_2) > \cdots > r(v_{12})$.

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

The runtime of PLL for labeling large graphs can be very long. As shown in Line 6 of Algorithm 1, the query function to calculate the distance between v_i and u (i.e., $Q(v_i, u, L^{PLL})$) takes $O(\delta)$ time. The number of function calls is $\Sigma_{u \in V} \Sigma_{r \in C(u)} deg(u)$, which may reach δm in the worst case. This leads to a rather high time cost in terms of function calls for PLL, which is confirmed by our extensive empirical studies: PLL takes more than 3 days for labeling the graph SINA with 58 million nodes and 261 million edges.

Remarks. Note that PLL can work with any total order on V. Since there are $|V|!$ different total orders on V (the number of permutations of nodes in V), the selection of the node order in optimizing the space and/or temporal efficiency of PLL remains an open problem. It has been suggested by existing literature [31] that

Algorithm 1: PLL

```
   Input: Graph $G(V, E)$
   Output: The index $L^{\mathsf{PLL}}$
1  for $i = 1, 2, \cdots, n$ do
2      $Q \leftarrow$ a queue with only one element $v_i$;
3      $dist(v_i) \leftarrow 0$ and $dist(v) \leftarrow \infty, \forall v \in V \setminus v_i$;
4      while $Q \neq \emptyset$ do
5          $u \leftarrow Q.pop()$;
6          if $Query(v_i, u, L^{\mathsf{PLL}}) \leq dist(u)$ then
7              continue;
8          $L^{\mathsf{PLL}}(u) \leftarrow L^{\mathsf{PLL}}(u) \cup \{(v_i, dist(u))\}$;
9          for $w \in N(u)$ do
10             if $dist(w) = \infty$ then
11                 $dist(w) \leftarrow dist(u) + 1$;
12                 $Q.push(w)$;
13 return $L^{\mathsf{PLL}}$ ;
```

betweenness-centrality-based node order may be better than degree-based node order; however, improving PLL based on betweenness centrality faces the two challenges listed below.

– The computation of the exact betweenness centrality is as expensive as computing pairwise shortest distances, which is unaffordable on large graphs.
– The best practice of optimizing PLL based on approximate betweenness, that is, cost-effectively estimating the betweenness to reduce the index size of PLL is yet to be explored.

Part of this paper will dedicate to exploring betweenness centrality in forming a better distance index. Specifically, Section 2.4 will introduce the definition of betweenness centrality; Section 5 will propose a better algorithm for distance index construction based on a novel sampling-based betweenness centrality computation.

2.4 Node Order: Betweenness Centrality

This paper actively explores the computation and application of betweenness-centrality-based node order in improving the efficiency of distance indexing. This section introduces betweenness centrality related concepts.

Given a graph $G(V, E)$ and two nodes $s, t \in V$, denote by $\sigma_{s,t}$ the number of different shortest paths between s and t (note that two different shortest paths between s and t can overlap on some nodes). Denoted by $\sigma_{s,t}(v)$, for $\forall v \in V$, the number of, among all the s-t shortest paths, shortest paths through v. If $s = t$, then $\sigma_{s,t} = 1$; if $v = s$ or $v = t$, then $\sigma_{s,t}(v) = 0$. We now define, for each node $v \in V$, its betweenness $\mathsf{bc}(v)$.

Definition 2 (Betweenness) Given graph $G(V, E)$, for $\forall v \in V$, betweenness centrality

$$\mathsf{bc}(v) = \sum\nolimits_{s,t \in V} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}.$$

Example 3 For node pair v_3, v_{10} in Fig. 1, there are two shortest paths between them: $p_1(v_3, v_{10}) = \{v_3, v_2, v_{10}\}$, and $p_2(v_3, v_{10}) = \{v_3, v_1, v_{10}\}$. Then, $\sigma_{v_3,v_{10}} = 2$. Since only $p_1(v_3, v_{10})$ passes through v_2, then $\sigma_{v_3,v_{10}}(v_2) = 1$. $\sigma_{v_3,v_{10}}(v_9) = 0$ because there is no v_3-v_{10} shortest path through v_9.

The betweenness centrality costs quadratic [13] time to compute, which is expensive for big graphs. For efficiently estimating betweenness centrality, we also resort to k-betweenness [14], a variation of betweenness centrality. Given a positive integer k, k-betweenness is defined for each node $v \in V$ by considering only shortest paths whose lengths are no more than k.

Definition 3 (k-Betweenness [14]) Given a graph $G(V, E)$ and an integer $k \geq 0$, the k-betweenness

$$\mathsf{kbc}(v) = \sum\nolimits_{s,t \in V, dist(s,t) \leq k} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}, \text{ for each } v \in V.$$

k-betweenness is a meaningful approximation of the betweenness centrality since k-betweenness is exactly the betweenness centrality when k approaches the diameter (the longest shortest path) of the graph. It was proposed since paths of long distances are less likely to form new edges, e.g., friendships in a social network [12] or a joint work in a collaboration network.

Example 4 If k is set to 2, then node pair v_{10}, v_{12} contributes nothing to k-betweenness of other nodes since their shortest distance $dist(v_{10}, v_{12}) = 3$.

$\mathsf{kbc}(v)$ is an aggregation over all the shortest paths of lengths no larger than k. To simplify the discussions on the computation of $\mathsf{kbc}(v)$, we introduce the concept of partial k-betweenness $\mathsf{kbc}_s(v)$ which is the portion of $\mathsf{kbc}(v)$ contributed by paths starting from node s.

Definition 4 (Partial k-Betweenness) Given graph $G(V, E)$, an integer $k \geq 0$ and a node $s \in V$,

$$\text{for } \forall v \in V, \mathsf{kbc}_s(v) = \sum\nolimits_{t \in V, dist(s,t) \leq k} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}.$$

Note that k-betweenness can be easily derived from partial k-betweenness

$$\mathsf{kbc}(v) = \Sigma_{s \in V} \mathsf{kbc}_s(v).$$

Therefore, the computation of $\mathsf{kbc}(v)$ boils down to computing $\mathsf{kbc}_s(v)$ for each node s of G. According to the definition of $\mathsf{kbc}_s(v)$, it can be observed that $\mathsf{kbc}_s(v)$ becomes zero if v is not included in any shortest path sourced at s with $\leq k$ length; thus, we have Lemma 3.

Lemma 3 *If $dist(s, v) \geq k$ or $dist(s, v) = 0$ then* $\mathsf{kbc}_s(v) = 0$.

Proof If $dist(s, v) = 0$, $\mathsf{kbc}_s(v) = 0$ since $\sigma_{v,t}(v) = 0$ for any $t \in V$. If $dist(s, v) = k$, v can only be the end point of any s-t shortest path $p(s, t)$ with $|p(s, t)| \leq k$. Then, $\sigma_{s,v}(v) = 0$. If $dist(s, v) > k$, there is no s-t path via v with $|p(s, t)| \leq k$, and $\sigma_{s,t}(v) = 0$. Thus, $\mathsf{kbc}_s(v) = 0$.

An exact algorithm to compute k-betweenness is presented in [14]. The basic idea is to perform a graph traversal sourced from each $s \in V$ to compute $\mathsf{kbc}_s(v)$, for $\forall v \in V$. Compared to betweenness computation, calculating k-betweenness only needs to visit nodes within a distance k to each source, thus improving the efficiency. However, for small-world graphs, the number of nodes within distance k (k greater than 2) to each source may still be large [42], which makes the exact k-betweenness computation for large graphs undesirable.

3 Parallelized Distance Labeling

Sections 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 1 *For any two nodes $\forall u, v \in V$, v is a hub of u under* PLL*, i.e., $(v, dist(v, u)) \in L^{\mathsf{PLL}}(u)$, 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 4-6 are derived from Theorem 1.

Lemma 4 *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 1), $r(v) > r(u)$.

Lemma 5 *For $\forall u \in V$, $(u, 0) \in L^{\mathsf{PLL}}(u)$.*

Proof We make the path as $p(u, u)$ and according to Theorem 1, $(u, 0)$ will be always inserted to $L^{\mathsf{PLL}}(u)$.

Lemma 6 *For $\forall (u, v) \in E$, $(u, 1) \in L^{\mathsf{PLL}}(v)$, if $r(u) > r(v)$; otherwise,$(v, 1) \in L^{\mathsf{PLL}}(u)$.*

Proof Let $p(u, v)$ be the path with an edge. According to Theorem 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 $L^{\mathsf{PLL}}_{<i}(u)$ the snapshot of $L^{\mathsf{PLL}}(u)$ at the beginning of the i-th round and by $L^{\mathsf{PLL}}_i(u)$ the incremental label of u built in the i-th round.

Definition 5 (Order Specific Label Set)

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

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

Definition 6 (Order Partial Label Set)

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

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

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

Lemma 7 (Order Dependency) *$L^{\mathsf{PLL}}_i(u)$ depends on $L^{\mathsf{PLL}}_{<i}(u)$. Specifically, $L^{\mathsf{PLL}}_i(u) =$*

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

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 1, i) v_i is a hub of u and ii) for $\forall v \in S \setminus v_i$, v is a not a hub of v_i, and thus $Query(v_i, u, L^{\mathsf{PLL}}_{<i}) > dist(v_i, u)$. If $r(v_i) < r(w)$, then v_i is not a hub of u and label $(w, dist(w, v_i)), (w, dist(w, u)) \in L^{\mathsf{PLL}}_{<i}$ and thus $Query(v_i, u, L^{\mathsf{PLL}}_{<i}) = dist(v_i, u)$.

Lemma 7 shows that $L^{\mathsf{PLL}}_i(u)$ depends on $L^{\mathsf{PLL}}_{<i}(u)$ while $L^{\mathsf{PLL}}_{<i}(u)$ depends on $L^{\mathsf{PLL}}_{i-1}(u)$. Such a convolved dependency can hardly be removed as long as the labels are built in the node order.

Example 5 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 $L^{\mathsf{PLL}}_1(u)$, $\forall u \in V$. 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 $L^{\mathsf{PLL}}_1 = \{L^{\mathsf{PLL}}_1(u) | u \in V\}$. The order dependency in the column v_7: partial set $L^{\mathsf{PLL}}_{<7} = \bigcup_{i<7, u \in V} L^{\mathsf{PLL}}_i(u)$ 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 $Query(v_i, u, L^{\mathsf{PLL}}_{<i}) = dist(u, v_i)$ 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 $Query(v_i, u, L^{\mathsf{PLL}}_{<i})$ 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.

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}	$(v_1,2)$	-	$(v_3,2)$	$(v_4,1)$	$(v_5,1)$	-	-	-	-	-	-	$(v_{12},0)$	$(v_{12},0)$	$(v_4,1)$ $(v_5,1)$	$(v_1,2)$ $(v_3,2)$

Definition 7 (Distance Specific Label Set)

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

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

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 8.

Definition 8 (Distance Partial Label Set)

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

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

The equivalence of the index L^{PLL} and the novel index L^{PSL} is given in Theorem 2.

Theorem 2 *$L^{\mathsf{PSL}} = L^{\mathsf{PLL}}$.*

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

Distance Dependency. Definition 7 and Definition 8 provide us an opportunity in removing the order dependency in the label construction process.

Theorem 3 (Distance Dependency) *$L^{\mathsf{PSL}}_d(u)$ depends on $L^{\mathsf{PSL}}_{<d}$. Specifically, given a node u, for a node $v \in V$ with $r(v) > r(u)$ and $dist(u,v) = d$, $(v, dist(v,u)) \in L^{\mathsf{PSL}}_d(u)$ if and only if $Query(u, v, L^{\mathsf{PSL}}_{<d}) > d$.*

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 1, we have two exclusive cases:

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

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

Therefore, if $(v, dist(v,u)) \notin L^{\mathsf{PSL}}_d(u)$, namely, v is not a hub of u, then $w \ne v$, and then $Query(u, v, L^{\mathsf{PSL}}_{<d}) = d$. Besides, if $(v, dist(v,u)) \in L^{\mathsf{PSL}}_d(u)$, 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, $Query(u, v, L^{\mathsf{PSL}}_{<d}) > d$.

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 7 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 away to a node u or prune the 1-hop away nodes. Note that according to Lemma 4, 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 $L^{\mathsf{PSL}}_2(v_{11})$. 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 to generate $L^{\mathsf{PSL}}_d(u)$, 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 $L_d^{\mathsf{PSL}}(u)$ by gathering the labels of its neighbors, namely, $L_{d-1}^{\mathsf{PSL}}(v)$, for $v \in N(u)$? We formally show that $\bigcup_{v\in N(u)} L_{d-1}^{\mathsf{PSL}}(v)$ is sufficient to create $L_d^{\mathsf{PSL}}(u)$ in Lemma 8.

Lemma 8 *All the hub nodes of labels in $L_d^{\mathsf{PSL}}(u)$ appear in labels $\bigcup_{v\in N(u)} L_{d-1}^{\mathsf{PSL}}(v)$ as hub nodes.*

Proof We show that if a node is not a hub of any node $v \in N(u)$ in $L_{d-1}^{\mathsf{PSL}}(v)$, then it is not a hub of u in $L_d^{\mathsf{PSL}}(u)$. Let $w \neq u$ be a hub of u in $L_d^{\mathsf{PSL}}(u)$ but is not a hub of any node $v \in N(u)$ in $L_{d-1}^{\mathsf{PSL}}(v)$. 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 $(w, dist(w,s)) \in L^{\mathsf{PLL}}$. Since $dist(w,s) = d-1$, w is a hub of $L_{d-1}^{\mathsf{PSL}}(s)$, contradiction.

Pruning Conditions. From Lemma 8, we can construct $L^{\mathsf{PSL}}(u)$ in an iterative way and the initial condition is given in Lemma 5 by inserting u to the label $L_0^{\mathsf{PSL}}(u)$ as its own hub. However, pouring all nodes in $\bigcup_{v\in N(u)} L_{d-1}^{\mathsf{PSL}}(v)$ directly into $L_d^{\mathsf{PSL}}(u)$ produces a large set of candidate labels. Therefore, we propose two rules to prune unnecessary label entries.

Lemma 9 *A hub w in the label set $\bigcup_{v\in N(u)} L_{d-1}^{\mathsf{PSL}}(v)$ is not a hub of u if $r(w) < r(u)$.*

Proof Lemma 4.

Lemma 10 *A hub w in the label set $\bigcup_{v\in N(u)} L_{d-1}^{\mathsf{PSL}}(v)$ is not a hub of u in $L_d^{\mathsf{PSL}}(v)$ if $Query(w,u,L_{<d}^{\mathsf{PSL}}) \leq d$.*

Proof If $Query(w,u,L_{<d}^{\mathsf{PSL}}) < d$, then $dist(w,u) < d$, w is not a hub of u with distance $dist(w,u) = d$. If $Query(w,u,L_{<d}^{\mathsf{PSL}}) = d$, 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 1, w is not be a hub of u in L^{PLL}.

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

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

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

Theorem 4 (Label Propagation Function)

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

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

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 8 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 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 6 (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 8) (Line 8) and then, pruning conditions 9-10 apply (Line 10-11). Entry $(w, dist(w,u))$ is then added to $L_d^{\mathsf{PSL}}(u)$ (Line 11-12).

Example 8 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, $L_2^{\mathsf{PSL}}(v_6) = \{(v_1, 2)\}$.

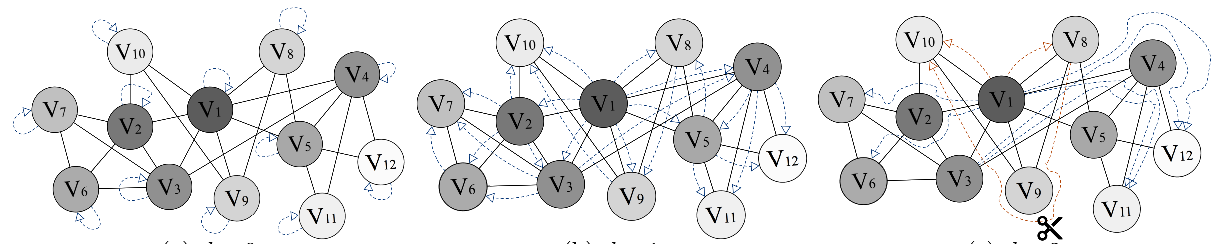


Fig. 2 The execution of PSL from $d = 0$, $d = 1$ to $d = 2$

Algorithm 2: PSL

```
   Input: Graph $G(V, E)$
   Output: The index $L^{PSL}$
1  Insert $(u, 0)$ into $L_0^{PSL}(u)$, $\forall u \in V$;
2  for $(u, v) \in E$ 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  $d \leftarrow 2$;
6  while $L_{d-1}^{PSL}$ is not empty do
7      for $u \in V$ in parallel do
8          $cand(u) \leftarrow$ hubs in $L_{d-1}^{PSL}(v)$, $\forall v \in N(u)$;
9          for each node $w \in cand(u)$ do
               // Pruning Condition Lemma 9
10             if $r(w) < r(u)$ then continue;
               // Pruning Condition Lemma 10
11             if $Query(w, u, L_{<d}^{PSL}) \leq d$ then continue;
12             Insert $(w, d)$ into $L_d^{PSL}(u)$;
13     $d \leftarrow d + 1$;
14 return $L^{PSL}$;
```

Theorem 5 *The time complexity of* PSL *under one core environment is $O(\delta^2 \cdot m)$.*

Proof Let $L^{PSL} = L^{PSL}_{<D+1} = L^{PLL}$. For each label in $L^{PSL}(v)$, 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 $L^{PSL} = L^{PLL}$ 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 9 (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 is non-trivial if it includes at least two nodes. Definition 10 obtains nodes in non-trivial equivalent classes under the two equivalence relations and Lemma 11 shows that these non-trivial equivalent classes are disjoint.

Definition 10 Define three vertex sets V_1, V_2 and V_3 with

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

Example 9 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 11 *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

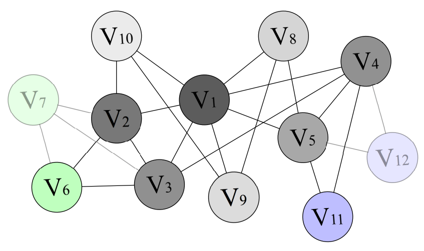


Fig. 3 Equivalence Relation Reduction

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 11, 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 11

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

Example 10 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 11 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 12 *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	$\lvert V \rvert$	$\lvert V_1 \setminus F(V_1) \rvert$	$\lvert V_2 \setminus F(V_2) \rvert$	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

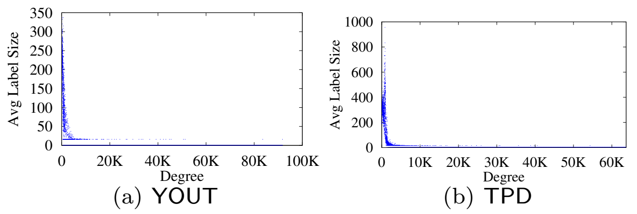
Example 12 Denote by $F(V') = \{v \in V' | v = f(v)\}$ the remained nodes in a set under equivalence reduction. Table 2 shows the effectiveness of the equivalence relations on index reduction. For YOUT (TPD), around 33.15% (17.67%) and 0.45% (0.67%) of nodes are eliminated by the first and the second 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

Fig. 4 PLL: Degree and Label Size

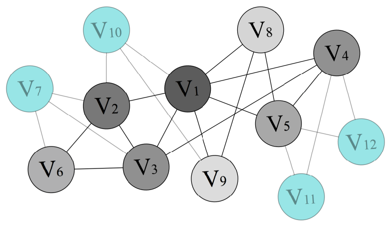


Fig. 5 Local Minimum Set

Definition 12 (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:

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

Example 13 In Fig. 5, $\mathsf{M(G)} = \{v_7, v_{10}, v_{11}, v_{12}\}$. 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 13 *For any node $\forall v \in \mathsf{M(G)}$ 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 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.

Construct Labels for $V \setminus \mathsf{M(G)}$. Lemma 13 shows that removing nodes in $\mathsf{M(G)}$ does not affect the label set of any node in $V \setminus \mathsf{M(G)}$. 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 $u \in N(v) \cap \mathsf{M(G)}$, without $L_{d-1}^{\mathsf{PSL}}(u)$ we cannot construct $L_d^{\mathsf{PSL}}(u)$ using Theorem 4.

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

Definition 13 (Generalized Neighbors) Given a node $v \in V \setminus \mathsf{M(G)}$, we define two neighbor sets. $N^1(v) = N(v) \setminus \mathsf{M(G)}$ includes the neighbors of v that fall in $V \setminus \mathsf{M(G)}$ and $N^2(v) = \{w | w \in (N(u) \setminus \{v\}), \forall u \in (N(v) \cap \mathsf{M(G)})\}$ includes the two-hop neighbors of v connected via nodes in $\mathsf{M(G)}$.

Example 14 In Fig. 5, since $v_9 \in V \setminus \mathsf{M(G)}$, $N^1(v_9) = \{v_1, v_8\}$, $N^2(v_9) = \{v_1, v_2\}$.

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

Lemma 14 *Given a node $v \in V \setminus \mathsf{M(G)}$, $N^1(v) \cap \mathsf{M(G)} = \emptyset$ and $N^2(v) \cap \mathsf{M(G)} = \emptyset$.*

Proof $N^1(v) \cap \mathsf{M(G)} = \emptyset$ by Definition 13. Let $x \in N^2(v)$ be a node expanded from $y \in N(v) \cap \mathsf{M(G)}$. If $x \in \mathsf{M(G)}$, then $r(y) < r(x)$ and $r(x) < r(y)$, contradiction.

Example 15 In Fig. 5, $N^2(v_9) = \{v_1, v_2\}$, which are all in the set $V \setminus \mathsf{M(G)}$.

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 6 *For each node $u \in V \setminus \mathsf{M(G)}$*

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

where $L_d^{\mathsf{PSL}}(u, w) =$

$$\begin{cases} \emptyset, \ if\ r(w) < r(u)\ or\ Query(w, u, L_{<d}^{\mathsf{PSL}}) \leq dist(w, u); \\ (w, dist(w, u)),\ otherwise. \end{cases}$$

Proof Let L'' be the labels drawn from Equation (4). We reuse the proof of Theorem 4 by showing that the hubs L' constructed in Equation (1) is a subset of the hubs in L''. According to Lemma 8, $\bigcup_{v' \in N^2(u)} C_{d-2}(v')$ 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.

Example 16 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%).

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

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 $L^{\mathsf{PSL}}(s)$ with hub w. $H(w) = \infty$ if w is not a hub of s. Since the label set $L^{\mathsf{PSL}}(s)$ may not be available, we construct H in two cases.

- If $s \in V \setminus \mathsf{M(G)}$, we hash the labels in $L^{\mathsf{PSL}}(s)$ 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 15 *When $s,t \in \mathsf{M(G)}$, the time cost of distance query is $O(\Sigma_{a\in N(s)}|L^{\mathsf{PSL}}(a)| + \Sigma_{b\in N(t)}|L^{\mathsf{PSL}}(t)|)$.*

Proof For s, we store nodes in $\{L^{\mathsf{PSL}}(a)|a \in N(s)\}$ in H. For t, we scan the nodes in $\{L^{\mathsf{PSL}}(b)|b \in N(t)\}$ to gain the distance. The linear scan takes $O(|\{L^{\mathsf{PSL}}(a)|a \in N(s)\}| + |\{L^{\mathsf{PSL}}(b)|b \in N(s)\}|)$ 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 17 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 Index Optimization with Betweenness-based Node Order

The PSL proposed in Section 3 parallelizes PLL in a multi-core environment, and the main bottleneck of this labeling method is the unaffordable index size. The two index reduction techniques proposed in Section 4 are built upon a node order which is, by default, degree based. To further reduce the index size, this section investigates the application of betweenness-based node order in PSL. As suggested by [31] and verified by our preliminary experimentation (Exp-9, Section 7), betweenness-based node order leads to a smaller index size. The difficulty in applying the betweenness-centrality to PSL is two-fold. 1) The computation of the betweenness centrality for all the nodes is computationally expensive ($O(mn)$ [13]) for big graphs. 2) The best practice of cost-effectively optimizing PSL with approximate betweenness is unknown. A better estimation of k-betweenness leads to a smaller index size; however, improving estimation precision can be exhaustive. Section 5.1 first proposes a sampling-based approach for estimating k-betweenness; to further improve the estimation efficiency, Section 5.2 presents a pool-based sampling algorithm. Section 5.3 introduces an algorithm in engaging the betweenness estimation in PSL for index reduction.

5.1 Basic Sampling

Exact k-betweenness of a node $v \in V$ summarizes the partial betweenness $\mathsf{kbc}_s(v)$ over all source nodes s in V. However, only a small number of sources s contribute to the computation of $\mathsf{kbc}(v)$: Lemma 3 shows that nodes s with $dist(s,v) = 0$ or $dist(s,v) \geq k$ has $\mathsf{kbc}_s(v) = 0$. These useless sources can be safely removed for v.

Definition 14 (k-Reachable Set) The k-reachable set of a vertex $v \in V$ is defined as $\mathsf{R}(v) = \{s|0 < dist(s,v) < k\}$.

Example 18 To estimate $\mathsf{kbc}(v_9)$ (with $k = 2$) in Fig. 1, we only consider source nodes in $\mathsf{R}(v_9) = \{v_1, v_2, v_3, v_4, v_5, v_8, v_{10}\}$ since nodes w outside $\mathsf{R}(v_9)$ make the partial betweenness $\mathsf{kbc}_w(v_9)$ zero.

Under the framework of betweenness approximation [7], random samples need to be selected from $\mathsf{R}(v)$. Suppose we randomly select some nodes S from $\mathsf{R}(v)$ for a node v. For each sample $s \in S$, $\mathsf{kbc}_s(v)$ can be computed by undertaking a graph traversal from s [14]. We estimate $\mathsf{kbc}(v)$ with

$$\widetilde{\mathsf{kbc}}(v) = (\sum_{s\in S} \mathsf{kbc}_s(v)) \cdot \frac{|\mathsf{R}(v)|}{|S|}.$$

Lemma 16 shows that $\widetilde{\mathsf{kbc}}(v)$ is an unbiased estimator of $\mathsf{kbc}(v)$.

Lemma 16 *$E(\widetilde{\mathsf{kbc}}(v)) = \mathsf{kbc}(v)$, for $\forall v \in V$.*

Proof For $\forall s \in \mathsf{R}(v)$, we define a random variable $X_s = \mathsf{kbc}_s(v) \cdot |\mathsf{R}(v)|$. We have $E(X_s) = \sum_{s\in\mathsf{R}(v)} \frac{1}{|\mathsf{R}(v)|} X_s = \sum_{s\in\mathsf{R}(v)} \frac{1}{|\mathsf{R}(v)|}\mathsf{kbc}_s(v) \cdot |\mathsf{R}(v)| = \mathsf{kbc}(v)$. When aggregating X_s over samples s in S, we have $E(\widetilde{\mathsf{kbc}}(v)) = E(\sum_{s\in S} \mathsf{kbc}_s(v) \cdot \frac{|\mathsf{R}(v)|}{|S|}) = E(\sum_{s\in S} X_s \cdot \frac{1}{|S|}) = E(X_s) = \mathsf{kbc}(v)$.

Lemma 17 *Suppose $K = \max_{s\in S}(\mathsf{kbc}_s(v))$,*

$$P(|\widetilde{\mathsf{kbc}}(v) - \mathsf{kbc}(v)| > \epsilon) \leq 2\exp\left(-2|S| \cdot \left(\frac{\epsilon}{K \cdot |\mathsf{R}(v)|}\right)^2\right)$$

Proof Let $X_1, X_2, \cdots, X_q$ be independent random variables with values in $[a, b]$, and $\overline{X} = \frac{X_1+X_2+\cdots+X_q}{q}$, then $P(|E(\overline{X}) - \overline{X}| > \epsilon) \leq 2\exp(-2n \cdot (\frac{\epsilon}{b-a})^2)$ by Hoeffding's inequality [23]. For any $s \in S$, we define $X_s = \mathsf{kbc}_s(v) \cdot |\mathsf{R}(v)|$, then $E(X_s) = \mathsf{kbc}(v)$, $\overline{X_s} = \widetilde{\mathsf{kbc}}(v)$, $q = |S|$, $a = 0$, $b = K \cdot |\mathsf{R}(v)|$. Plugging these terms into Hoeffding's inequality proves the lemma.

Discussion. Given a node $v \in V$, by Lemma 17, for a given $\epsilon \in R^+$ and $\delta \in (0, 1)$, if $|S| \geq \frac{\ln(\frac{2}{\delta})\cdot K^2 \cdot |\mathsf{R}(v)|^2}{2\epsilon^2}$, we obtain an estimation of $\mathsf{kbc}(v)$ within an additive error ϵ with a probability at least δ [23]. The required sample size, unfortunately, is very large. Although there are techniques to reduce the sample size [11,37,38], there are two drawbacks of this basic sampling approach: i) each node v needs to compute $|\mathsf{R}(v)|$ to reach an unbiased estimator $\widetilde{\mathsf{kbc}}(v)$ of $\mathsf{kbc}(v)$; ii) node v precomputes $\mathsf{R}(v)$ to select samples. The cost of obtaining $\mathsf{R}(v)$ (and $|\mathsf{R}(v)|$) for $\forall v \in V$ by performing $n = |V|$ BFS (with a length limited to k) is no better than the exact k-betweenness computation.

5.2 Pool-based Sampling

Size Estimation. To solve the first drawback of the above sampling method, we select a pool S_{size} of nodes to approximate $|\mathsf{R}(v)|$ for *all* nodes $v \in V$. Specifically, for each node $s \in S_{\mathsf{size}}$, we conduct a **k-bounded BFS** from s which only visits nodes that are $< k$ hops away from s. Suppose v has been visited $n_{\mathsf{size}}(v)$ times by the k-bounded BFS from s (that is, there are $n_{\mathsf{size}}(v)$ samples in S_{size} that belong to $\mathsf{R}(v)$), we estimate $|\mathsf{R}(v)|$ with

$$\widetilde{\mathsf{R}}(v) = n_{\mathsf{size}}(v) \cdot \frac{n}{|S_{\mathsf{size}}|}.$$

Lemma 18 shows that $\widetilde{\mathsf{R}}(v)$ is an unbiased estimator of $|\mathsf{R}(v)|$.

Lemma 18 *$E(\widetilde{\mathsf{R}}(v)) = |\mathsf{R}(v)|$, for $\forall v \in V$.*

Proof For each sample $s \in S_{\mathsf{size}}$, we define a random variable X_s to indicate whether s is in $\mathsf{R}(v)$, that is, $X_s = \begin{cases} 1, \; if\ s \in \mathsf{R}(v) \\ 0, \; otherwise \end{cases}$. Then, $P(X_s = 1) = \frac{|\mathsf{R}(v)|}{n}$ and $E(X_s) = \frac{|\mathsf{R}(v)|}{n}$. Thus, $E(n_{\mathsf{size}}(v)) = \sum_{s\in S_{\mathsf{size}}} E(X_s) = \frac{|\mathsf{R}(v)|}{n} \cdot |S_{\mathsf{size}}|$, and $E(\widetilde{\mathsf{R}}(v)) = |\mathsf{R}(v)|$.

Algorithm 3: Size Estimation

```
   Input: Graph $G(V,E)$, hop $k$, time budget $T_s$
   Output: $S_{size}$, $n_{size}(v)$ for $\forall v \in V$
1  $S_{size} \leftarrow \emptyset$;
2  $n_{size}(v) \leftarrow 0$, for $\forall v \in V$;
3  for sampling time $\leq T_s$ do
4      $s \leftarrow$ a node chosen uniformly at random from $V$;
5      $S_{size} \leftarrow S_{size} \cup \{s\}$;
6      Let $\sigma_{s,s} \leftarrow 1$ and $dist(s) \leftarrow 0$;
7      For $\forall v \in V \setminus \{s\}$, let $\sigma_{s,v} \leftarrow 0$ and $dist(v) \leftarrow -1$;
8      $curr \leftarrow \{s\}$; $next \leftarrow \emptyset$;
9      for $i = 0, 1, \cdots, k-2$ do
10         for $\forall v \in curr$ and $\forall w \in N(v)$ do
11             if $dist(w) = -1$ then
12                 $dist(w) \leftarrow dist(v) + 1$;
13                 $n_{size}(w) \leftarrow n_{size}(w) + 1$;
14                 $next \leftarrow next \cup \{w\}$;
15         $curr \leftarrow next$, $next \leftarrow \emptyset$;
16 return $S_{size}$, $n_{size}(v)$ for $\forall v \in V$;
```

Algorithm 3 estimates, for $\forall v \in V$, the size $n_{\mathsf{size}}(v)$ of $\mathsf{R}(v)$, within the sampling time budget T_s. For a random sample s (Line 4), we append s in S_{size} (Line 5), and perform a k-bounded BFS from s (Line 6-15). For each newly visited node v (i.e., $s \in \mathsf{R}(v)$), $n_{\mathsf{size}}(v)$ is increased by 1 (Line 13). The process continues until the sampling time goes beyond the budget T_s (Line 3).

Partial Betweenness Estimation. To solve the second drawback of the basic sampling approach, we select a pool of nodes S_{bc} to compute k-partial betweenness for *all* nodes $v \in V$. Among the samples in S_{bc}, suppose $n_{\mathsf{bc}}(v)$ nodes (denoted as S_v) are included in $\mathsf{R}(v)$ for a certain v, we summarize the partial k-betweenness of v over S_v to obtain $\kappa(v)$, for each individual node $v \in V$; $\kappa(v)$ shall be used to estimate $\mathsf{kbc}(v)$.

$$\kappa(v) = \sum_{s\in S_v} \mathsf{kbc}_s(v).$$

In this way, we avoid sampling from $\mathsf{R}(v)$ for each node in V.

Algorithm 4 estimate $\kappa(v)$ and $n_{\mathsf{bc}}(v)$, for $\forall v \in V$ (Line 1-27) within time budget T_s. We repeatedly select a sample s (Line 3) uniformly at random, until the time budget T_s is consumed (Line 2). Given s, we follow the method introduced in [14] to compute $\mathsf{kbc}_s(v)$, for $\forall v \in V$. Note that in this process, $\mathsf{kbc}_s(v)$ is added to $\kappa(v)$, and $n_{\mathsf{bc}}(v)$ is increased by 1.

We first conduct a k-bounded BFS from s (Line 1-18), aiming at computing $\sigma_{s,v}$, the number of shortest paths from s to v, for $\forall v \in R(s)$ (equivalently $s \in R(v)$). Specifically, we use *curr* and *next* to store nodes expanded in the current round and the nodes to expand in the next round. $\sigma_{s,v}$ is initialized with zero for all $v \in V$, except for s, whose $\sigma_{s,s}$ is set to 1; $dist(v)$ is initialized to -1 for all v, except for s, which is set to 0 (Line 6-7). We first insert s into *curr* to start the BFS (Line 8), and then we explore nodes within distance k to s (Line 9): for each node $v \in curr$ (Line 10), we check v's neighbor w (Line 11). If w is not visited before (Line 12), $dist(w)$ is updated to $dist(v) + 1$ (Line 13), and w is appended to *next* and S (Line 14-15). If w is one hop farther than v regarding s, we increase $\sigma_{s,w}$ by adding $\sigma_{s,v}$ to it (Line 16-17). Then, *next* is assigned to *curr* for the next round (Line 18).

When all nodes within distance k to s have been stored in stack S, we perform a backward BFS to compute $\mathsf{kbc}_s(v)$, for $\forall v \in V$ (Line 19-27). Specifically, $\mathsf{kbc}_s(v)$ is initialized as zero (Line 19), and we visit nodes w in S reversely – in the order of non-increasing distance to s (Line 21). For each neighbor v of w, if v is one hop closer than w regarding s, $\mathsf{kbc}_s(w)$ is used to update $\mathsf{kbc}_s(v)$ (Line 22-24). For each $v \neq s$, $\mathsf{kbc}_s(v)$ is added to $\kappa(v)$, and $n_{\mathsf{bc}}(v)$ is increased by 1 (Line 26-27).

Algorithm 4: Partial Betweenness Estimation

```
   Input: Graph $G(V,E)$, hop $k$, time budget $T_s$
   Output: $n_{\mathsf{bc}}(v), \kappa(v)$ for $\forall v \in V$
1  $n_{\mathsf{bc}}(v) \leftarrow 0$, $\kappa(v) \leftarrow 0$, for $\forall v \in V$;
2  while $sampling\ time \leq T_s$ do
3      $s \leftarrow$ a random node in $V$;
       // Forward BFS
4      $curr \leftarrow \emptyset, next \leftarrow \emptyset$;
5      $S \leftarrow$ an empty stack;
6      For $\forall v \in V \setminus \{s\}$, $\sigma_{s,v} \leftarrow 0$, $dist(v) \leftarrow -1$;
7      $\sigma_{s,s} \leftarrow 1$, $dist(s) \leftarrow 0$;
8      $curr \leftarrow curr \cup \{s\}$;
9      for $i = 0, 1, \cdots, k-1$ do
10         for $\forall v \in curr$ do
11             for $\forall w \in N(v)$ do
12                 if $dist(w) = -1$ then
13                     $dist(w) \leftarrow dist(v) + 1$;
14                     $next \leftarrow next \cup \{w\}$;
15                     $S \leftarrow S \cup \{w\}$;
16                 if $dist(w) = dist(v) + 1$ then
17                     $\sigma_{s,w} \leftarrow \sigma_{s,w} + \sigma_{s,v}$;
18         $curr \leftarrow next$, $next \leftarrow \emptyset$;
       // Backward BFS
19     $\mathsf{kbc}_s(v) \leftarrow 0$, $\forall v \in V$;
20     while $S \neq \emptyset$ do
21         $w \leftarrow$ pop from $S$;
22         for $v \in N(w)$ do
23             if $dist(w) \neq dist(v) + 1$ then continue;
24             $\mathsf{kbc}_s(v) \leftarrow \mathsf{kbc}_s(v) + \frac{\sigma_{s,v}}{\sigma_{s,w}} \cdot (1 + \mathsf{kbc}_s(w))$;
25             if $v \neq s$ then
26                 $\kappa(v) \leftarrow \kappa(v) + \mathsf{kbc}_s(v)$;
27                 $n_{\mathsf{bc}}(v) \leftarrow n_{\mathsf{bc}}(v) + 1$;
28 return $n_{\mathsf{bc}}(v), \kappa(v)$ for $\forall v \in V$;
```

To analyze the estimation accuracy of the pool-based sampling, we focus on $S_v = \{v \in S_{\mathsf{bc}} | v \in R(v)\}$ and its size $n_{\mathsf{bc}}(v) = |S_v|$, for each $v \in V$. We show that for each v, the size $n_{\mathsf{bc}}(v)$ is proportional to $|R(v)|$.

Lemma 19 $E(n_{\mathsf{bc}}(v)) = |S_{\mathsf{bc}}| \times \frac{|R(v)|}{n}$.

Proof For a node that is chosen uniformly at random from V, it falls in $R(v)$ with probability $\frac{|R(v)|}{n}$. Aggregating this probability over all nodes in S_{bc} derives the expectation $E(n_{\mathsf{bc}}(v)) = |S_{\mathsf{bc}}| \times \frac{|R(v)|}{n}$.

Order Generation. With the outputs of size estimation and partial betweenness estimation, we show that

$$\widetilde{\mathsf{kbc}}(v) = \frac{\kappa(v)}{n_{\mathsf{bc}}(v)} \cdot (n_{\mathsf{size}}(v) \cdot \frac{n}{|S_{\mathsf{size}}|}) \quad (5)$$

is an unbiased estimator of $\mathsf{kbc}(v)$.

Lemma 20 $E(\widetilde{\mathsf{kbc}}(v)) = \mathsf{kbc}(v)$, *for* $\forall v \in V$.

Proof Given a node $v \in V$, we define a random variable $X_s = \mathsf{kbc}_s(v) \cdot n_{\mathsf{size}}(v) \cdot \frac{n}{|S_{\mathsf{size}}|}$, for $\forall s \in \mathsf{R}(v)$. Then, $E(X_s) = \frac{1}{|\mathsf{R}(v)|} \cdot \sum_{s \in \mathsf{R}(v)} \mathsf{kbc}_s(v) \cdot E(n_{\mathsf{size}}(v) \cdot \frac{n}{|S_{\mathsf{size}}|}) = \sum_{s \in \mathsf{R}(v)} \mathsf{kbc}_s(v) = \mathsf{kbc}(v)$ (size estimation and betweenness estimation are independent). Suppose samples S_{bc} are used to estimate the betweenness, among which nodes $S_v \subseteq S_{\mathsf{bc}}$ are included in $\mathsf{R}(v)$. The size of S_v is $n_{\mathsf{bc}}(v)$. Then, $E(\widetilde{\mathsf{kbc}}(v)) = E(\sum_{s \in S_v} \mathsf{kbc}_s(v) \cdot \frac{n_{\mathsf{size}}(v)}{n_{\mathsf{bc}}(v)} \cdot \frac{n}{|S_{\mathsf{size}}|}) = E(\sum_{s \in S_v} X_s \cdot \frac{1}{n_{\mathsf{bc}}(v)}) = E(X_s) = \mathsf{kbc}(v)$.

Algorithm 5: Order Generation

```
   Input: Graph $G(V,E)$, hop $k$, time budget $T$, $\theta$
   Output: $r(v)$ for $\forall v \in V$
1  $S_{\mathsf{size}}, n_{\mathsf{size}}(v) \leftarrow$ Algorithm 3($G$, $k$, $\theta T$), for $\forall v \in V$;
2  $n_{\mathsf{bc}}(v), \kappa(v) \leftarrow$ Algorithm 4($G$, $k$, $(1-\theta)T$), for
     $\forall v \in V$;
3  $\widetilde{\mathsf{kbc}}(v) \leftarrow \frac{\kappa(v)}{n_{\mathsf{bc}}(v)} \cdot (n_{\mathsf{size}}(v) \cdot \frac{n}{|S_{\mathsf{size}}|})$, for $\forall v \in V$;
4  Generate $r(v)$ in non-increasing order of $\widetilde{\mathsf{kbc}}(v)$;
5  return $r(v)$ for $\forall v \in V$;
```

By applying Lemma 17, the accuracy is given below.

Lemma 21 *Suppose* $K = \max_{s \in S_v}(\mathsf{kbc}_s(v))$,

$$P(|\widetilde{\mathsf{kbc}}(v) - \mathsf{kbc}(v)| > \epsilon) \leq 2\exp\left(-2n_{\mathsf{bc}}(v) \cdot (\frac{\epsilon}{K \cdot |\mathsf{R}(v)|})^2\right).$$

With the estimation $\widetilde{\mathsf{kbc}}(v)$ of $\mathsf{kbc}(v)$ computed for each node $v \in V$, the betweenness-based node order r is set such that for any $u, v \in V$, $r(u) > r(v)$ if

- $\widetilde{\mathsf{kbc}}(u) > \widetilde{\mathsf{kbc}}(v)$;
- $\widetilde{\mathsf{kbc}}(u) = \widetilde{\mathsf{kbc}}(v)$, $ID(u) > ID(v)$.

Algorithm 5 shows the order generation algorithm. First, Algorithm 3 (with sampling time budget θT) and Algorithm 4 (with sampling time budget $(1-\theta)T$) are called to estimate size and partial betweenness of each node (Line 1-2). Then, $\widetilde{\mathsf{kbc}}(v)$ is computed based on Equation 5 (Line 3). Finally, $r(v)$ is determined by the above rule (Line 4). The parameter θ controls the time used in Algorithm 3 and Algorithm 4. In practice, we set the parameter θ as 0.2 since it leads to a good effect when k-betweenness is used for ordering nodes.

Lemma 22 *The time cost of Algorithm 5 is $O(|S|(n+m))$ where $|S|$ is the number of samples used in stage 1 and stage 2.*

Remarks. In Algorithm 5, instead of giving a predefined sample size, the sample size is controlled adaptively by the sampling time – the estimation accuracy will improve if more time is given.

5.3 Improved Betweenness-based Node Order

Recall that the index reduction techniques proposed in Section 4 remove the local minimum set $\mathsf{M(G)}$ which, in the current node order, is determined by the k-betweenness of nodes in V. The remaining nodes $V \setminus \mathsf{M(G)}$, however, may have different k-betweenness in the updated graph structure. Therefore, it is desirable to recompute k-betweenness for $V \setminus \mathsf{M(G)}$ once $\mathsf{M(G)}$ is eliminated for a better approximation.

Virtual Graph. To recompute k-betweenness, one challenge is that if two nodes $u, v \in V \setminus \mathsf{M(G)}$ are connected only by nodes in $\mathsf{M(G)}$, then u and v are disconnected in the updated graph. To this end, given a graph $G(V, E)$, we define a virtual graph $\overline{G}(\overline{V}, \overline{E})$ with $\overline{V} = V \setminus \mathsf{M(G)}$ as its node set. Recall Definition 13, we add two types of edges to $\overline{E}$ for each node $u \in \overline{V}$, (u, v) with $l(u, v) = 1$ for every $v \in N^1(v)$ and (u, v) with[5] $l(u, v) = 2$ for every $v \in N^2(v) \setminus N^1(v)$. The edge set is sufficient to keep the connectivity: due to the property of the local minimum reduction, if $u \in \mathsf{M(G)}$ for some node $u \in V$, then $N(v) \cap \mathsf{M(G)} = \emptyset$; therefore, to retain the connectivity, we only need to consider $u, v \in \overline{V}$ that are connected only by one node in $\mathsf{M(G)}$.

[5] For the convenience of presentation, we replace an edge (u, v) of length 2 with two unit-weighted edges $(u, w), (w, v)$ with a new node w interpolated in between.

Example 19 In Fig. 5, since $v_8 \in N^1(v_9)$, we have the edge (v_9, v_8) with weight 1 in $\overline{G}$; since $v_2 \in N^2(v_9) \setminus N^1(v_9)$, we have the edge (v_9, v_2) with weight 2 in $\overline{G}$.

Algorithm 6: Improved Order Generation

```
Input: Graph $G(V,E)$, hop $k$, sample time $T$, $\theta$
Output: $r(v)$, for $\forall v \in V$
1  $\widetilde{\mathsf{kbc}}(v), \forall v \in V \leftarrow$ Algorithm 5($G$, $k$, $(1-2\theta)T$);
2  $\mathsf{M(G)} \leftarrow \emptyset$;
3  for $v \in V$ do
4      for $w \in N(v)$ do
5          if $\widetilde{\mathsf{kbc}}(v) > \widetilde{\mathsf{kbc}}(w)$ then continue;
6      $\mathsf{M(G)} \leftarrow \mathsf{M(G)} \cup \{v\}$;
7  $\overline{V} \leftarrow V \setminus \mathsf{M(G)}$;
8  $\overline{E} \leftarrow \{(u,v) | u, v \in V \setminus \mathsf{M(G)}\}$;
9  for $v \in \overline{V}$ do
10     $N^1(v) \leftarrow \emptyset, N^2(v) \leftarrow \emptyset$;
11     for $w \in N(v) \cap \overline{V}$ do insert $w \rightarrow N^1(v)$;
12     for $w \in N(v) \cap \mathsf{M(G)}$ do
13         for $u \in N(w)$ do
14             if $u \neq v$ then
15                 Insert $u \rightarrow N^2(v)$;
16     for $w \in N^1(v)$ do
17         Insert edge $(v, w)$ with weight 1 in $\overline{E}$;
18     for $w \in N^2(v) \setminus N^1(v)$ do
19         Insert edge $(v, w)$ with weight 2 in $\overline{E}$;
20 For $\forall v \in \overline{V}$, $S_{\mathsf{size}}, n_{\mathsf{size}}(v) \leftarrow$ Algorithm 3($\overline{G}(\overline{V}, \overline{E})$, $k$,
       $\theta T$);
21 For $\forall v \in \overline{V}$, $\kappa(v), n_{\mathsf{bc}}(v) \leftarrow$ Algorithm 4($\overline{G}(\overline{V}, \overline{E})$, $k$,
       $\theta T$);
22 $\widetilde{\mathsf{kbc}}(v) \leftarrow \frac{\kappa(v)}{n_{\mathsf{bc}}(v)} \cdot (n_{\mathsf{size}}(v) \cdot \frac{n}{|S_{\mathsf{size}}|})$, for $\forall v \in \overline{V}$;
23 Sort $r(v)$ in non-increasing order of $\widetilde{\mathsf{kbc}}(v)$, for
       $\forall v \in \overline{V}$;
24 Set $r(v)$ as the minimum among its neighbors, for
       $\forall v \in \{V \setminus \overline{V}\}$ ;
25 return $r(v)$, for $\forall v \in V$;
```

Improved Order Generation. The sampling algorithm considering local minimum set $\mathsf{M(G)}$ elimination is in Algorithm 6. Similar to Algorithm 5, we set θ to 0.2 in practice. First, Algorithm 5 is invoked to compute $\widetilde{\mathsf{kbc}}(v), \forall v \in V$ (Line 1), and nodes v with the minimum $\widetilde{\mathsf{kbc}}(v)$ among $N(v)$ constitute $\mathsf{M(G)}$ (Line 2-6). Then, for each node $v \in \overline{V} = V \setminus \mathsf{M(G)}$, $N^1(v)$ and $N^2(v)$ are formed according to the Definition 13 (Line 8-15), where edges $\overline{E}$ are formed in Line 8 and Line 16-19.

Afterwards, Algorithm 3 approximates the size of $\mathsf{R}(v)$ in $\overline{G}$, for $\forall v \in \overline{V}$; Algorithm 4 obtains k-partial betweenness $\widetilde{\mathsf{kbc}}(v)$ in $\overline{G}$, for $\forall v \in \overline{V}$. The k-betweenness of $v \in \overline{V}$ in $\overline{G}$ is then computed by the outputs of the above two algorithms (Line 22). For nodes in $\overline{V}$, their orders are defined by $\widetilde{\mathsf{kbc}}(v)$ in $\overline{G}$ (Line 24), while we enforce nodes in $V \setminus \overline{V}$ to have the minimum orders — these nodes remains to be local minimum set after the re-computing (Line 25).

6 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 [31].

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 [5,20,22] or build a complete index to fully support the distance query [4,26]. 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 [20]. 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 [22] as index to guide the online searching to process the distance query.

In the second category, PLL [4] constructs the index by performing pruned BFS whose detail is given in Section 2.3. The hop doubling approach in [26] applies generation rules to join the short paths to long paths, until the whole paths are covered. Compared to PLL, the algorithm proposed in [26] 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 [2] constructs the index by eliminating the high ranking nodes and add it to the labels of its neighbors. The approach proposed by Wei [46] 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. [3] 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 [33] to answer 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 [47,31].

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 [35,15]. Online processing on landmarks is used to improve the precision [44,36]. However, on small-world networks, the relative error becomes significant since the distances are bounded by the small diameter.

Betweenness Computation. Betweenness was proposed by Freeman [19]. The best exact computation algorithm incurs $O(nm)$ [13], which has been confirmed to be almost optimal for both sparse [10] and dense graphs [1]. Due to the complexity, numerous approximation algorithms are given to trade accuracy for speed. Pioneering work was done in [24] by using a sampling-based approach, and subsequent studies aimed at reducing the sampling costs [37,38,11]. For example, Matteo et al. [37] applied VC-dimension theory to calculate the sample size required to achieve the desired approximation. The computed sample size is independent of the number of vertices but depends only on the graph diameter (i.e., the longest shortest path in the graph). To eliminate the dependence on the graph diameter and to further reduce the required sample size, Matteo et al. [38] used the concepts of Rademacher averages and pseudodimension to accelerate the betweenness approximation. An experimental comparison of approximate algorithms is presented in the literature [6] to validate the efficiency and accuracy of various methods. Another line of direction investigates variations of betweenness to reduce the computation costs [14,34, 18]. k-betweenness used in this paper belongs to this category [14], and we devise approximation algorithms to compute it fast.

k-betweenness is an approximate notion of betweenness: when k reaches the graph diameter (the length of the longest shortest path in the graph), k-betweenness becomes betweenness. We use k-betweenness to holistically optimize the node order given the time resource in computing the node order. An adequate k strikes a balance between i) the gap between the k-betweenness and betweenness and i) the gap introduced by the sampling-based estimation of the k-betweenness — a larger k reduces the first gap while increases the second gap. In this paper, k is carefully chosen to holistically optimize the node order. As a type of centrality measures, k-betweenness can be used to identify important nodes in networks, such as biological networks [25], virus propagation networks [32], terrorist networks [16], and transportation networks [21]. The approximation algorithms proposed in the paper can produce elegant estimation in a given sampling time budget and thus be beneficial for the tasks on the above networks.

Table 5 The Description of the Datasets

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

Extensions from [30]. This work is an extension of the conference version [30]. Compared to [30], we make the following novel contributions. (1) In Section 2.3, the benefits (index reduction) and challenges (quadratic computation) of using betweenness-based node order in distance labeling are discussed. (2) In Section 2.4, betweenness related concepts are introduced, including betweenness and its variation k-betweenness. (3) In Section 5, approximation algorithms for k-betweenness computation and how to use betweenness estimation in the index construction process are presented. (4) In Section 7, corresponding experiments are conducted to verify the effectiveness of distance labeling using betweenness-based node order.

7 Experimental Results

In this section, we first validate the effects of parallelism and compression techniques in Section 7.1, followed by the evaluation of k-betweenness as a node order in Section 7.2.

All algorithms used in the experiments 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.

7.1 Test of Parallelism and Compression

Algorithms. We compare our proposed algorithms against the state-of-the-art algorithm PLL [4]. 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.

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[6][39], Stanford Large Network Dataset Collection[7][28], Laboratory for Web Algorithms[8] [9,8], and the Koblenz Network Collection[9] [27].

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 [4] 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

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

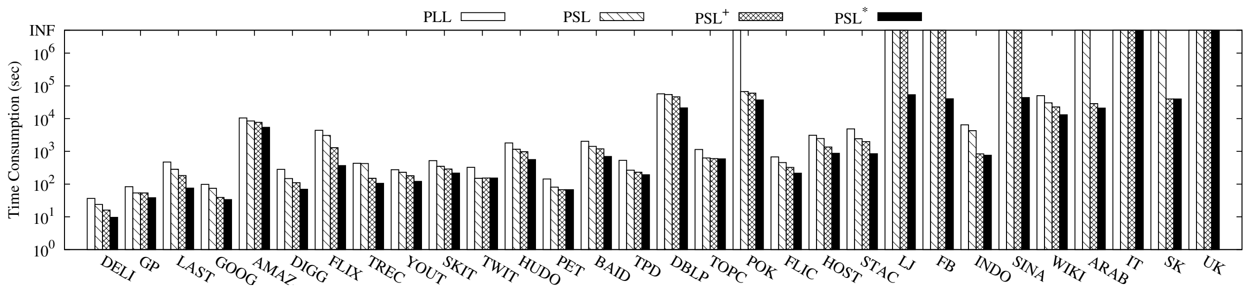


Fig. 6 The Comparison of the Index Time on One Core

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 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

$$\textit{speedup} = \frac{\text{the index time of the approach with 1 core}}{\text{the index time of the approach with } x \text{ cores}}.$$

According to the above equation, 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 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 scal-

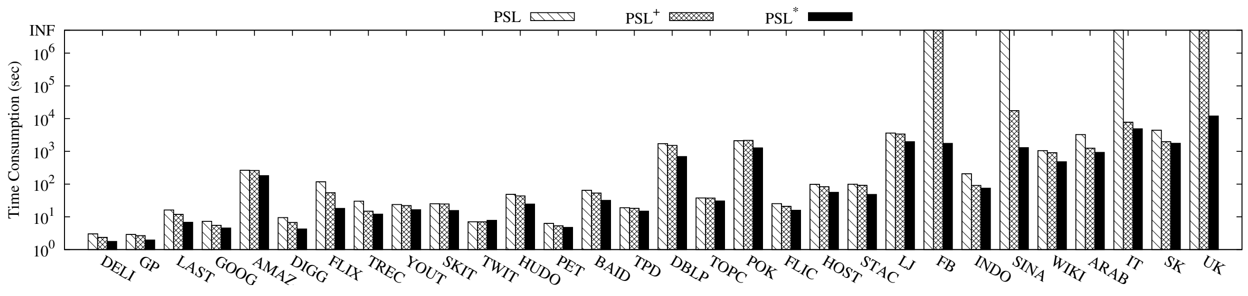


Fig. 7 The Comparison of the Index Time on 45 Cores

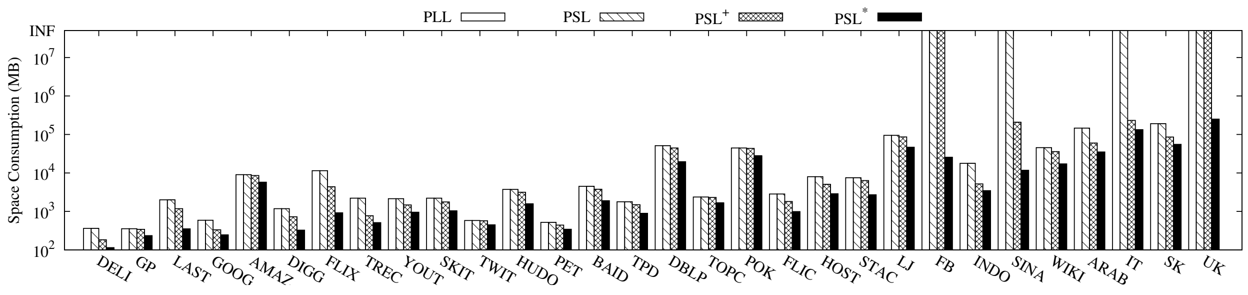


Fig. 8 The Comparison of the Index Size

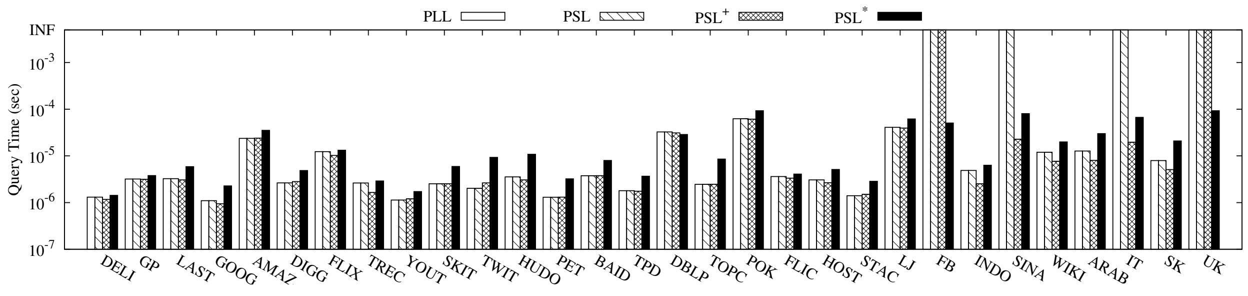


Fig. 9 The Comparison of the Query Time

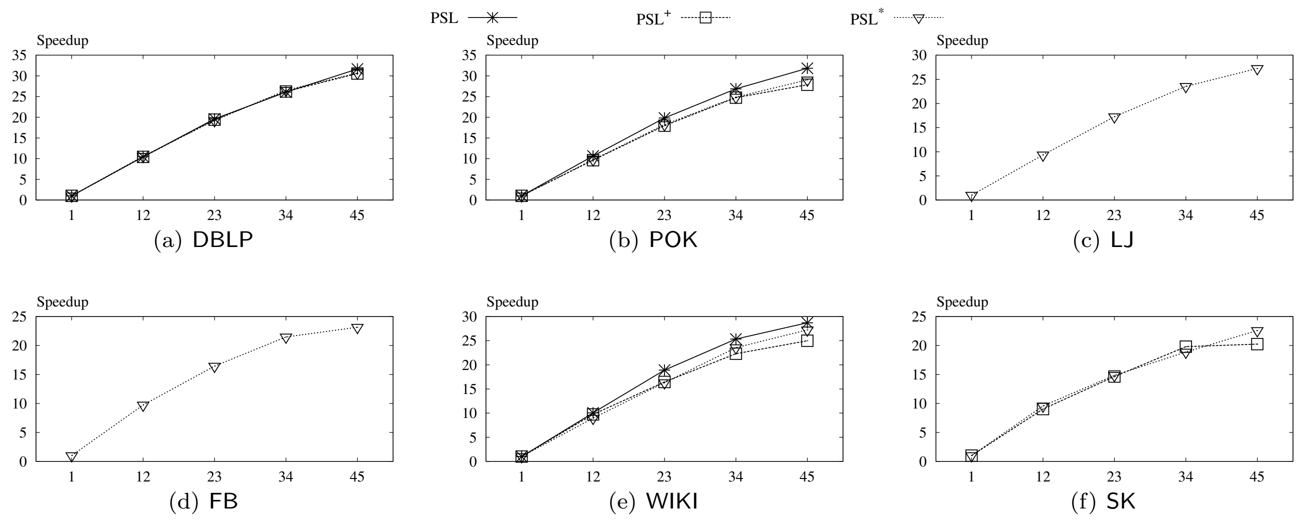


Fig. 10 The Effect of Core Number on the Index Time

ability test, we draw the conclusion that the proposed methods all show excellent scalability.

7.2 Test on Node Ordering

Algorithms. For PSL, we use $\mathsf{PSL_D}$ to denote PSL whose order is determined by degrees and $\mathsf{PSL_B}$ to de-

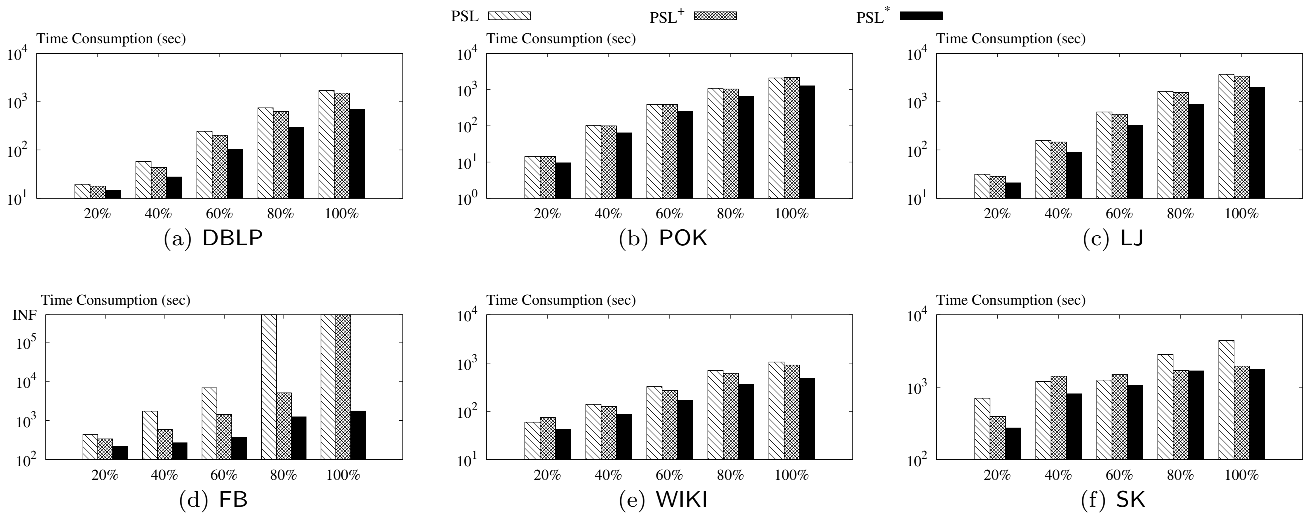


Fig. 11 The Test of Scalability for the Index Time

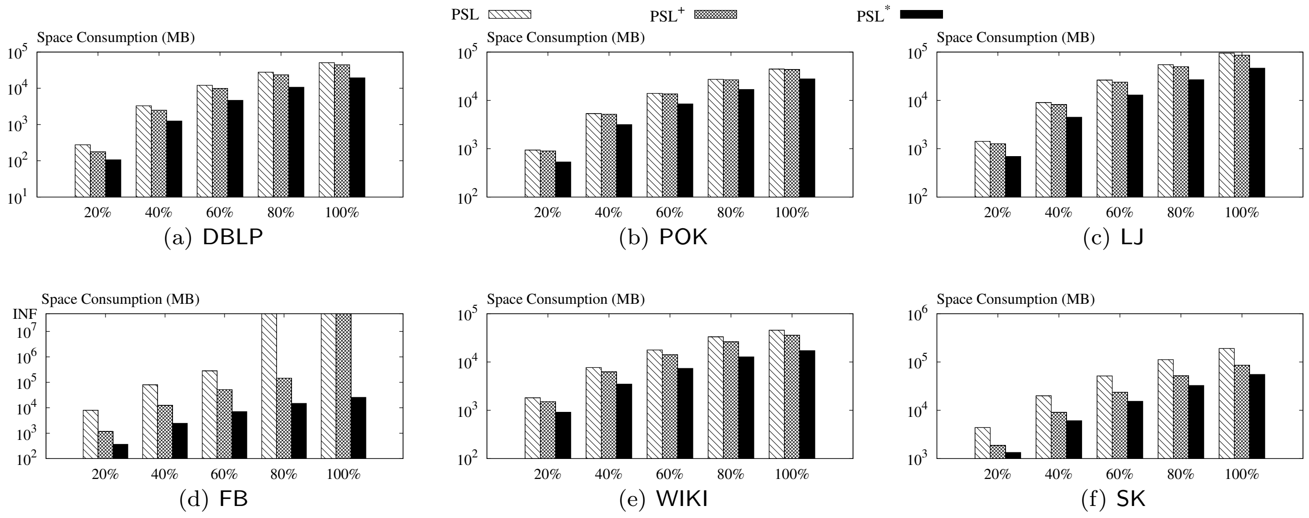


Fig. 12 The Test of Scalability for the Index Size

note PSL whose order is determined by k-betweenness. Furthermore, to test the effect of removing local minimum set on computing k-betweenness, we impose different node orders on PSL*, which includes the following three methods:

- PSL*$_D$: PSL* using degrees to determine node order.
- PSL*$_B$: PSL* using k-betweenness computed by the pool-based sampling method (Algorithm 5) for ordering.
- PSL*$_I$: PSL* using k-betweenness computed by the improved sampling method (Algorithm 6) for ordering.

Datasets. Experiments were performed on 30 real-world graphs in Table 5. Moreover, to further test the effect of different ordering methods, we provide two additional datasets, as shown in Table 6. The largest added graph has more than 5.5 billion edges.

Table 6 The Description of Added Datasets

Name	Dataset	n	m	Type
UK75	UK-2007-05[8]	105,896,555	3,738,733,648	Web Graph
UK07	UK-2007[8]	133,633,040	5,507,679,822	Web Graph

Exp 9: Degree-based and Betweenness-based Node Orders on PSL. We study the effect of node orders (using degree and betweenness) on PSL index sizes. Among them, we obtain the node orders determined by betweenness in two ways: PSL$_B$ whose order is determined by our proposed k-betweenness algorithm, and we set the parameter k to 4; and PSL$_C$ whose order is determined by classical betweenness. We use the

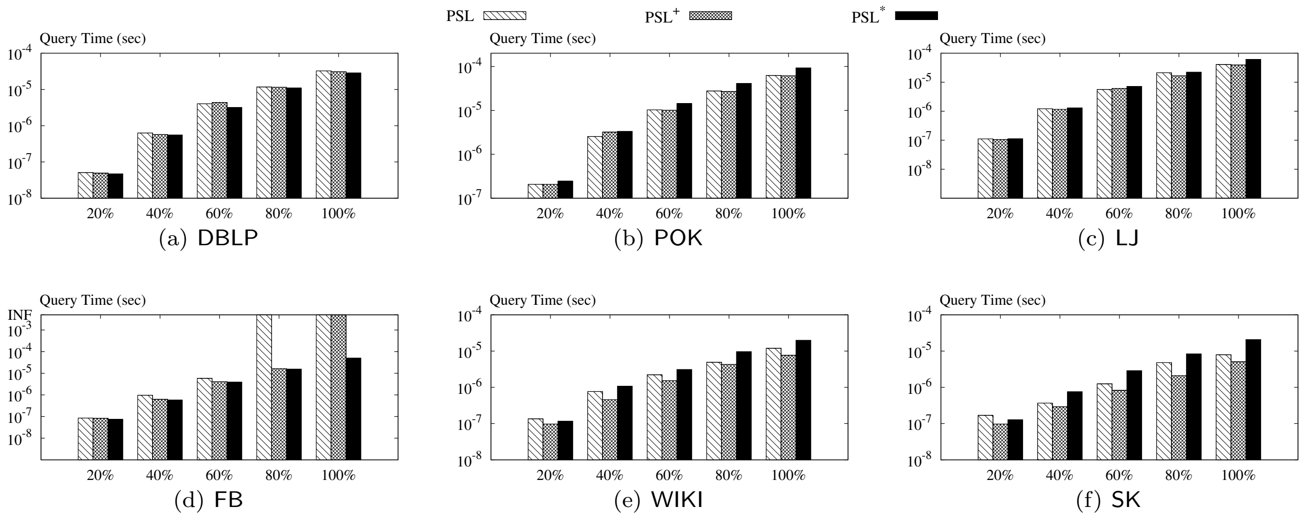


Fig. 13 The Test of Scalability for the Query Time

method ABRA[10] in [38] to estimate the classical betweenness values, and its parameters are set according to those in [38]. For $\mathsf{PSL_B}$ and $\mathsf{PSL_C}$, we stop sampling when the sampling time exceeds the same time threshold. We compared the index sizes of $\mathsf{PSL_D}$, $\mathsf{PSL_C}$ and $\mathsf{PSL_B}$ on all graphs where $\mathsf{PSL_D}$ can create indexes. The results are shown in Fig. 14.

First, we compare $\mathsf{PSL_B}$ with $\mathsf{PSL_D}$ to show that using betweenness is superior to using degree as the node order. As can be seen in Fig. 14, the index size of $\mathsf{PSL_B}$ is always smaller than that of $\mathsf{PSL_D}$, and the index size of $\mathsf{PSL_B}$ can be more than five times smaller than that of ARAB. These results show that setting betweenness to node order is useful for reducing the index size.

Then, we compare $\mathsf{PSL_B}$ with $\mathsf{PSL_C}$ to illustrate the necessity of the proposed k-betweenness approximation algorithm. In 19 out of 24 graphs, the index size of $\mathsf{PSL_B}$ is smaller than that of $\mathsf{PSL_C}$ (by a factor of up to 2.45 on WIKI); on other graphs, the index size of $\mathsf{PSL_B}$ is comparable to that of $\mathsf{PSL_C}$. This result illustrates why new betweenness approximation methods need to be designed for distance labeling: replacing classical betweenness with k-betweenness leads to a considerable reduction in the index size of $\mathsf{PSL_B}$ compared to $\mathsf{PSL_C}$, especially for large graphs.

Exp 10: Effect of Node Order on the Index Size of PSL*. The primary goal of determining the node order using k-betweenness is to reduce the index size — on a multiple core environment, the failure of labeling methods mainly results from the unaffordable index size. We compared PSL* using different node ordering methods, where the hop number k is set to 4, and the sampling time T is set to 3600 seconds. The effect of parameters T and k on the index size will be discussed later, and the results on all 32 graphs are given in Fig. 15.

Fig. 15 indicates that replacing degrees ($\mathsf{PSL^*_D}$) with k-betweenness ($\mathsf{PSL^*_B}$ and $\mathsf{PSL^*_I}$) enables the indexing on large graphs UK75 and UK07. This demonstrates the meaning of adopting k-betweenness as a node order. Moreover, on the 30 graphs where $\mathsf{PSL^*_D}$ finished labeling, the index size of $\mathsf{PSL^*_I}$ is, on average, 1.48 times smaller on average than that of $\mathsf{PSL^*_D}$, and the size of $\mathsf{PSL^*_D}$ is reduced by about 4 times at most.

We then verify that it is useful to consider the local minimum set ($\mathsf{M(G)}$) elimination in the computation of k-betweenness. As shown in Fig. 15, the index size of $\mathsf{PSL^*_B}$ can be sometimes larger than that of $\mathsf{PSL^*_D}$: $\mathsf{PSL^*_B}$'s index size is 1.22 times and 1.3 times that of $\mathsf{PSL^*_D}$ on FB and SK, respectively. In contrast, the index of $\mathsf{PSL^*_I}$ is always smaller than that of $\mathsf{PSL^*_D}$. Furthermore, the index size of $\mathsf{PSL^*_I}$ is, on average, 1.12 times smaller than that of $\mathsf{PSL^*_B}$, and the size of $\mathsf{PSL^*_B}$ is reduced by more than 1.67 times at most. These results are encouraging because it shows that taking $\mathsf{M(G)}$ into account can effectively reduce the index size under the same sampling time.

Exp 11: Effect of Node Order on Query Time of PSL*. Fig. 16 compares $\mathsf{PSL^*_D}$, $\mathsf{PSL^*_B}$, and $\mathsf{PSL^*_I}$ in query time. On average, $\mathsf{PSL^*_B}$ takes 0.91 times as long as $\mathsf{PSL^*_D}$, while $\mathsf{PSL^*_I}$ takes only 1.04 times as long as $\mathsf{PSL^*_D}$. This means that reducing the size does

[10] We chose ABRA for two reasons. First, as pointed out in [38], ABRA outperforms the method of [37]. Second, ABRA can be terminated at any time during execution, which leads to a fair comparison with our method. The source code of ABRA is also the code used in the literature [6], and has been implemented in parallel with OpenMP.

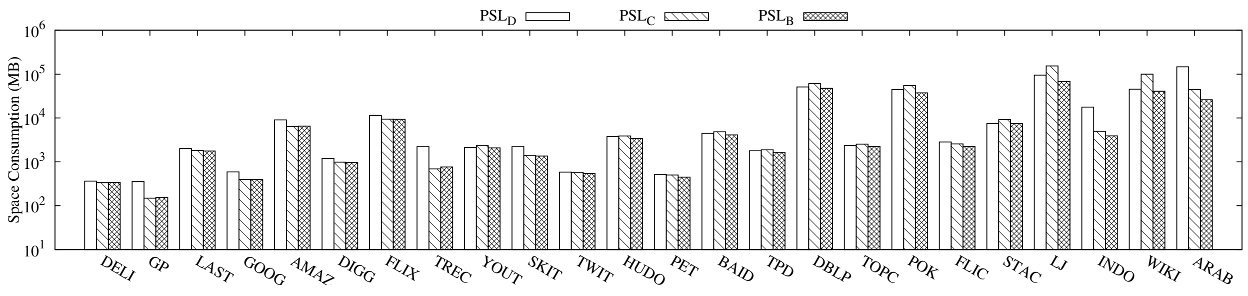


Fig. 14 The Comparison of Node Order Degree and Betweenness

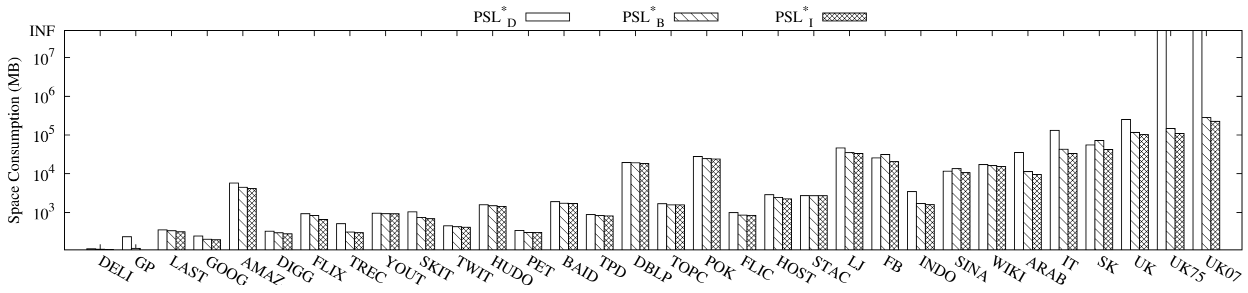


Fig. 15 The Effect of the Node Order on the Index size

not affect the query time — PSL$^*_B$ shortens the query time of PSL$^*_D$, and PSL$^*_I$'s query time is close to that of PSL$^*_D$.

Exp 12: Effect of Node Order on Index Time of PSL*. We show the index time (including one-hour sampling time for PSL$^*_B$ and PSL$^*_I$) for different ordering methods, and the results are given in Fig. 17. On all the graphs, the index time of PSL$^*_B$ does not exceed the index time of PSL$^*_D$ by more than 2 hours, while the index time of PSL$^*_I$ does not exceed the index time of PSL$^*_D$ by more than 1.5 hours. Note that the additional overhead in index time is acceptable: on the one hand we need time to estimate k-betweenness, on the other hand adopting k-betweenness as the node order does not significantly improve the index time.

It is also interesting to observe that on some graphs the index time is reduced when we replace PSL$^*_D$ by PSL$^*_I$: on UK, the index time of PSL$^*_D$ is 11986.17 seconds while the time is 9599.761 seconds for PSL$^*_I$. Furthermore, note that PSL$^*_D$ cannot index on large graphs such as UK75 and UK07 due to the exhaustive index size. These results support the idea of adopting k-betweenness as the node order, provided that index time can be dramatically reduced in a multi-core environment.

Exp 13: Effect of k on Index Size. We examine the effect of hop number k on the index size, where k is the parameter that defines k-betweenness. Since PSL$^*_I$ performs better than PSL$^*_B$ in reducing the index size, we only present the results of PSL$^*_I$. We varied the number k from 2, 3, 4, 5, to 6, and the results are shown in Fig. 18. Note that the red line in the figures are the index size when k is set as the diameter of the graph.

Fig. 18 shows that different graphs have different trends: as k increases, the index size first decreases and then increases on DBLP, POK, LJ, and WIKI; on FB, the index size decreases continuously; on SK, the index size first increases then decreases. The different trends suggest that we adopt k-betweenness rather than betweenness (when k is infinite) is desirable: given a limited sampling time, a larger k does not imply a smaller index. Furthermore, setting k to 4 allows a reasonably small index size on all graphs, and 4 is the default hop number for k-betweenness.

Exp 14: Effect of k on Query Time. We examine the effect of hop number k on the query time, where all experimental settings are the same as Exp 13. Fig. 19 shows that different graphs have different trends: as k increases, the query time first decreases and then increases on POK, LJ, and FB; the query time first increases then decreases on WIKI; the query time fluctuates on DBLP and SK. Also, by comparing with the query time obtained using betweenness (when k is set to infinity), we find that the query time obtained using k-betweenness are comparable. This shows that using k-betweenness as the node order can reduce the index size without sacrificing the query time.

Exp 15: Effect of Sampling Time T. PSL$^*_I$ adopts a sampling-based algorithm to approximate k-betweenness. Instead of giving the total sample size, PSL$^*_I$ provides the time limit T for the sampling process. To evaluate the effect of sampling time T on the

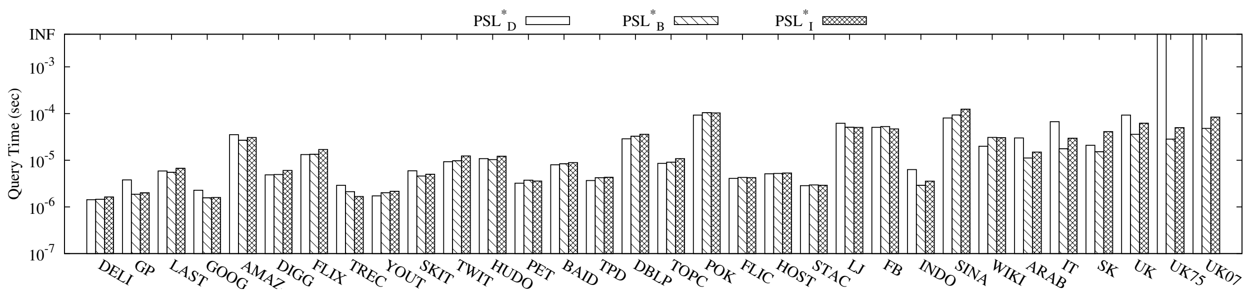


Fig. 16 The Effect of the Node Order on the Query Time

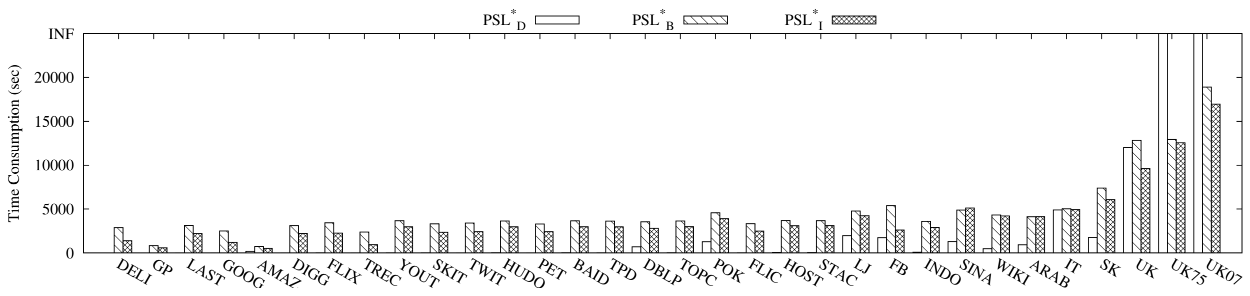


Fig. 17 The Effect of the Node Order on the Index Time

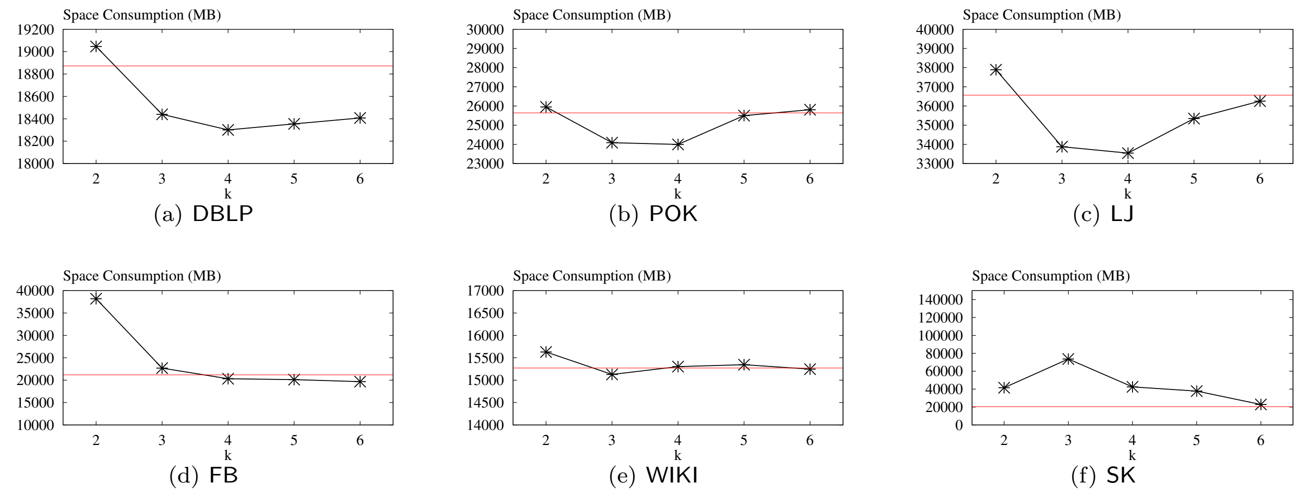


Fig. 18 The Effect of the Hop Number k on the Index Size (PSL*$_\mathsf{I}$)

index size of PSL*$_\mathsf{I}$, we changed T from 900, 1800, 3600, 5400, to 7200 seconds, and the results are given in Fig. 20.

On all the graphs, the index size does not increase as more sampling time is given. This is reasonable, as an increasing T leads to a more accurate estimation of k-betweenness. Furthermore, for some graphs, such as DBLP and FB, the index size reduce smoothly after one hour, which explains why 3600 seconds is the default sampling time for PSL*$_\mathsf{I}$. However, on large graphs such as LJ and SK, the index size keeps decreasing. This verifies the benefits of our method in handling large graphs when more sampling time is given.

Exp 16: Effect of Sampling Time T on Query Time. We examine the effect of sampling time T on the query time, where all experimental settings are the same as the Exp 15. Fig. 21 shows that the difference in query time across all graphs is insignificant when T is changed: despite the different trends in query time on various graphs, the ratio between the maximum and minimum query time on all graphs does not exceed 2.83. This result further highlights that k-betweenness as node order can guarantee good query time while reducing index size.

Exp 17: Overall Index Size Reduction Ratio. After the reduction of index time using multi-core par-

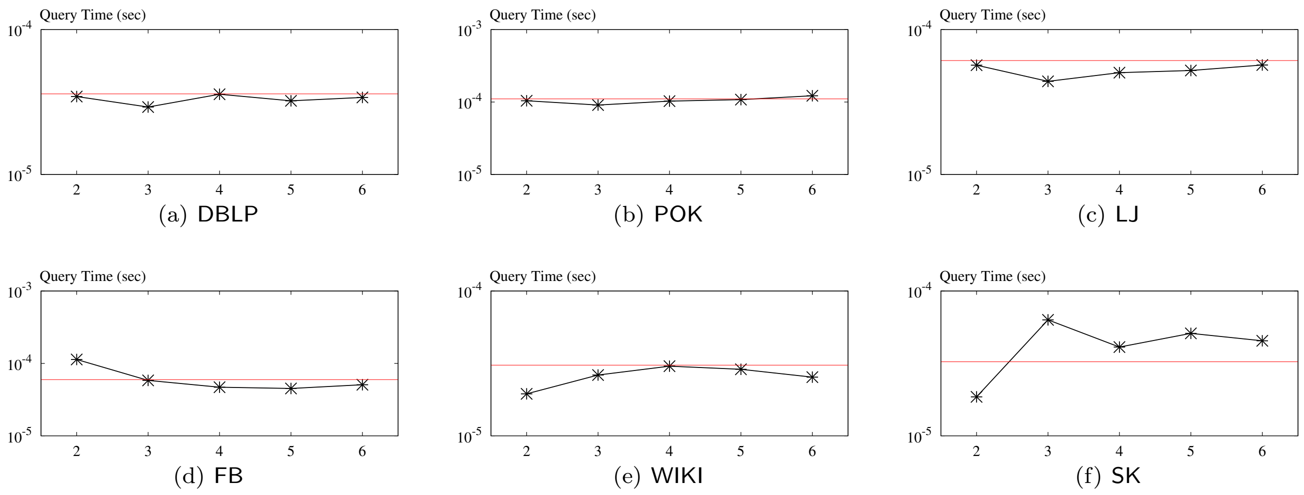


Fig. 19 The Effect of the Hop Number k on the Query Time

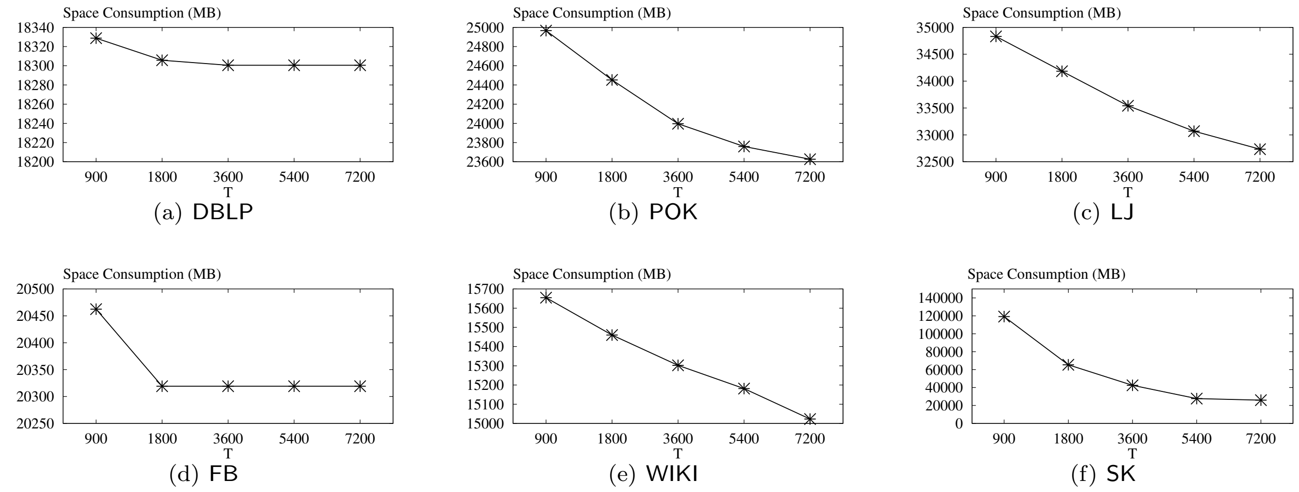


Fig. 20 The Effect of the Sampling Time T on the Index Size ($\mathsf{PSL^*_I}$)

Table 7 Overall Index Size Reduction Ratio

Name	$\mathsf{PSL_D}$ (MB)	$\mathsf{PSL^*_I}$ (MB)	*Ratio* %	Name	$\mathsf{PSL_D}$ (MB)	$\mathsf{PSL^*_I}$ (MB)	*Ratio* %
DELI	364.05	110.474	69.65	GP	355.24	107.495	69.74
LAST	1997.52	314.168	84.27	GOOG	589.11	196.834	66.59
AMAZ	9025.18	4150.954	54.01	DIGG	1178.41	278.791	76.34
FLIX	11444.23	657.548	94.25	TREC	2208.96	300.507	86.40
YOUT	2141.51	919.85	57.05	SKIT	2209.91	691.007	68.73
TWIT	582.58	414.14	28.91	HUDO	3738.98	1442.207	61.43
PET	519.32	305.785	41.12	BAID	4493.61	1717.496	61.78
TPD	1783.19	809.296	54.62	DBLP	50996.04	18300.504	64.11
TOPC	2365.64	1557.537	34.16	POK	44414.19	23996.3	45.97
FLIC	2839.96	845.43	70.23	HOST	8005.39	2230.982	72.13
STAC	7495.68	2686.053	64.17	LJ	94950.66	33542.286	64.67
INDO	17731.95	1581.035	91.08	WIKI	45447.10	15302.388	66.33
ARAB	146394.20	9587.213	93.45	SK	190216.16	42533.59	77.64

allelization, the study of index size reduction becomes important. This paper proposes two ways in reducing the index size of $\mathsf{PSL_D}$: i) index compression by removing redundant information (e.g., equivalent relationship reduction and local minimum set elimination); and ii) setting the node order using k-betweenness. The final $\mathsf{PSL^*_I}$ combines the above two reduction techniques. To further highlight the significance of size reduction, we compared the index size between $\mathsf{PSL_D}$ and $\mathsf{PSL^*_I}$. We use the metric *ratio* to show the percentage of index size reduction that $\mathsf{PSL^*_I}$ achieves compared to $\mathsf{PSL_D}$, where $ratio = 100\% - \frac{Index\ size\ of\ \mathsf{PSL^*_I}}{Index\ size\ of\ \mathsf{PSL_D}} \times 100\%$. Table 7 lists the *ratio* on all graphs that $\mathsf{PSL_D}$ can complete the labeling process.

Table 7 shows that $\mathsf{PSL^*_I}$ can compress $\mathsf{PSL_D}$'s index size by 94.25% on FLIX— $\mathsf{PSL_D}$'s index size is decreased by more than an order of magnitude. Furthermore, $\mathsf{PSL^*_I}$ can build the index on large graphs where $\mathsf{PSL_D}$ fails, demonstrating the necessity in using

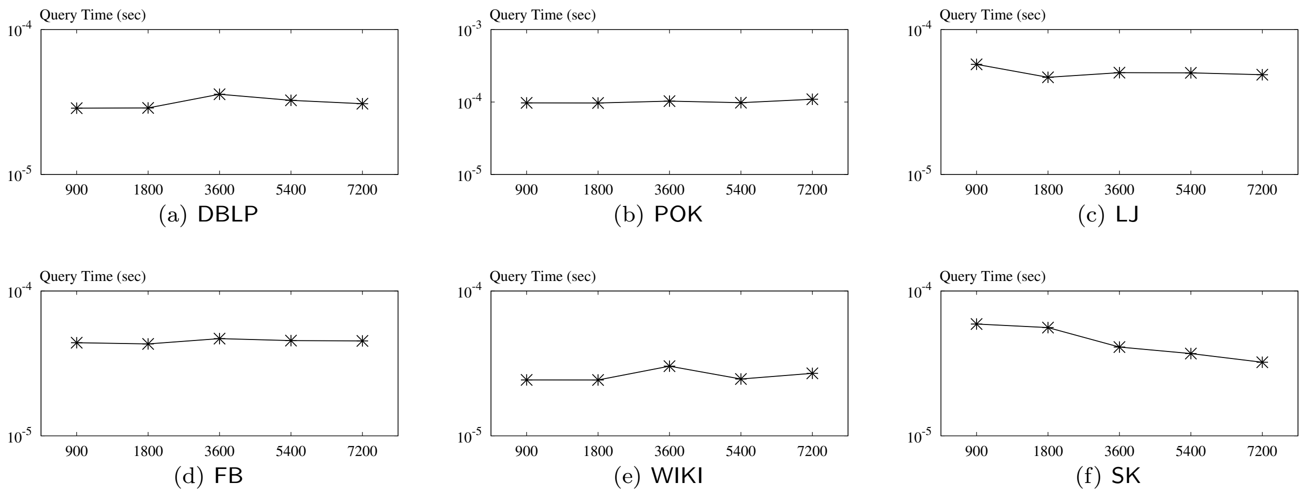


Fig. 21 The Effect of Sampling Time T on the Query Time

index reduction techniques for distance labeling even in a multi-core environment.

8 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 reduced by removing redundant nodes from the graph and removing labels of local minimum sets from the index. Scalable approximation algorithms for k-betweenness computation is proposed, so that k-betweenness can be used as a node order to further reduce the index size. 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.

Acknowledgements Miao Qiao is supported by Marsden Fund UOA1732, Royal Society of New Zealand and Catalyst: Strategic Fund 3721519 from Government Funding, Ministry of Business Innovation and Employment. Lu Qin is supported by ARC FT200100787. 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. Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1681–1697. SIAM, 2014.
2. Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hierarchical hub labelings for shortest paths. In *European Symposium on Algorithms*, pages 24–35. Springer, 2012.
3. Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. 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)*, pages 147–154. SIAM, 2014.
4. Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 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*, pages 349–360. ACM, 2013.
5. Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 144–155. ACM, 2012.
6. Ziyad AlGhamdi, Fuad Jamour, Spiros Skiadopoulos, and Panos Kalnis. A benchmark for betweenness centrality approximation algorithms on large graphs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2017.
7. David A Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 124–137. Springer, 2007.
8. Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.
9. Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference*

(WWW 2004), pages 595–601, Manhattan, USA, 2004. ACM Press.

10. Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science*, 322:51–67, 2016.
11. Michele Borassi and Emanuele Natale. Kadabra is an adaptive algorithm for betweenness via random approximation. *Journal of Experimental Algorithmics (JEA)*, 24(1):1–35, 2019.
12. Stephen P Borgatti and Martin G Everett. A graph-theoretic perspective on centrality. *Social networks*, 28(4):466–484, 2006.
13. Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.
14. Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2):136–145, 2008.
15. Wei Chen, Christian Sommer, Shang-Hua Teng, and Yajun Wang. A compact routing scheme and approximate distance oracle for power-law graphs. *ACM Transactions on Algorithms (TALG)*, 9(1):4, 2012.
16. Thayne Coffman, Seth Greenblatt, and Sherry Marcus. Graph-based technologies for intelligence analysis. *Communications of the ACM*, 47(3):45–47, 2004.
17. Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 937–946. Society for Industrial and Applied Mathematics, 2002.
18. Shlomi Dolev, Yuval Elovici, and Rami Puzis. Routing betweenness centrality. *Journal of the ACM (JACM)*, 57(4):1–27, 2010.
19. Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.
20. Ada Wai-Chee Fu, Huanhuan Wu, James Cheng, and Raymond Chi-Wing Wong. Is-label: an independent-set based labeling scheme for point-to-point distance querying. *Proceedings of the VLDB Endowment*, 6(6):457–468, 2013.
21. Roger Guimera, Stefano Mossa, Adrian Turtschi, and LA Nunes Amaral. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proceedings of the National Academy of Sciences*, 102(22):7794–7799, 2005.
22. Takanori Hayashi, Takuya Akiba, and Ken-ichi Kawarabayashi. Fully dynamic shortest-path distance query acceleration on massive networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1533–1542. ACM, 2016.
23. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.
24. Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. In *Network Analysis*, pages 62–82. Springer, 2005.
25. Hawoong Jeong, Sean P Mason, A-L Barabási, and Zoltan N Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.
26. Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *Proceedings of the VLDB Endowment*, 7(12):1203–1214, 2014.
27. Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.
28. Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.
29. Jianxin Li, Xinjue Wang, Ke Deng, Xiaochun Yang, Timos Sellis, and Jeffrey Xu Yu. Most influential community search over large social networks. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 871–882. IEEE, 2017.
30. Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. Scaling distance labeling on small-world networks. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1060–1077, 2019.
31. Ye Li, Man Lung Yiu, Ngai Meng Kou, et al. An experimental study on hub labeling based shortest path algorithms. *Proceedings of the VLDB Endowment*, 11(4):445–457, 2017.
32. Fredrik Liljeros, Christofer R Edling, Luis A Nunes Amaral, H Eugene Stanley, and Yvonne Åberg. The web of human sexual contacts. *Nature*, 411(6840):907–908, 2001.
33. Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *Proceedings of the 2018 International Conference on Management of Data*, pages 709–724. ACM, 2018.
34. Jürgen Pfeffer and Kathleen M Carley. k-centralities: Local approximations of global measures based on shortest paths. In *Proceedings of the 21st International Conference on World Wide Web*, pages 1043–1050, 2012.
35. Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 867–876. ACM, 2009.
36. Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):55–68, 2014.
37. Matteo Riondato and Evgenios M Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, 30(2):438–475, 2016.
38. Matteo Riondato and Eli Upfal. Abra: Approximating betweenness centrality in static and dynamic graphs with rademacher averages. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(5):1–38, 2018.
39. Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
40. Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1176–1185. ACM, 2014.
41. Chih-Ya Shen, Liang-Hao Huang, De-Nian Yang, Hong-Han Shuai, Wang-Chien Lee, and Ming-Syan Chen. On finding socially tenuous groups for online social networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 415–424. ACM, 2017.

42. Manuel Then, Moritz Kaufmann, Fernando Chirigati, Tuan-Anh Hoang-Vu, Kien Pham, Alfons Kemper, Thomas Neumann, and Huy T Vo. The more the merrier: Efficient multi-source graph traversal. *Proceedings of the VLDB Endowment*, 8(4):449–460, 2014.
43. Jeffrey Travers and Stanley Milgram. The small world problem. *Phychology Today*, 1(1):61–67, 1967.
44. Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. 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*, pages 1785–1794. ACM, 2011.
45. Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440, 1998.
46. Fang Wei. Tedi: efficient shortest path query answering on graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 99–110. ACM, 2010.
47. Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proceedings of the VLDB Endowment*, 5(5):406–417, 2012.

A Proof of Lemma 1

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

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