PSPC: Efficient Parallel Shortest Path Counting on Large-Scale Graphs

You Peng
The Chinese University of Hong Kong
Hong Kong, China
ypeng@se.cuhk.edu.hk

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hong Kong, China
yu@se.cuhk.edu.hk

Sibo Wang
The Chinese University of Hong Kong
Hong Kong, China
swang@se.cuhk.edu.hk

***Abstract*—In modern graph analytics, the shortest path is a fundamental concept. Numerous recent works concentrate mostly on the distance of these shortest paths. Nevertheless, in the era of betweenness analysis, the counting of the shortest path between s and t is equally crucial. It is also an important issue in the area of graph databases. In recent years, several studies have been conducted in an effort to tackle such issues. Nonetheless, the present technique faces a considerable barrier to parallel due to the dependencies in the index construction stage, hence limiting its application possibilities and wasting the potential hardware performance. To address this problem, we provide a parallel shortest path counting method that could avoid these dependencies and obtain approximately linear index time speedup as the number of threads increases. Our empirical evaluations verify the efficiency and effectiveness.**

***Index Terms*—Parallel, Shortest Path Counting, Graph Data Management**

I. Introduction

The shortest path problem [1]–[3] is a basic one in the field of graph analytics [4], [5], and numerous studies have been conducted on its related topics, e.g., shortest path distance [6]–[9], shortest path counting [10]–[13]. Given two vertices s and t, a shortest path between them is the one with the shortest length among all possible paths. A variety of applications, e.g., geographic navigation, Internet routing, socially tenuous group detecting [14], influential community searching [15], event detection [16], betweenness centrality [17], [18], and route planning [6], [19], make extensive use of it. In the aforementioned applications, distance between s and t is frequently taken into account when determining the vertices' importance and relevance, e.g., the nearest keyword search [20], and ranking search in social networks [21].

Beside distance, the shortest path counting (SPC) is a vital aspect of the shortest path related problems. This is due to the fact that basing relevance and importance merely on distance is uninformative [10]. In addition, many graphs in the real world have a limited diameter owing to the small-world phenomenon. Consequently, numerous pairs of vertices have the same distance between them. Several pairings of vertices will be considered equally relevant based on the distance information alone. This is really unrealistic. A typical example is given as follows: Consider graph H in Figure 1. Both t_1 and t_2 are located at a distance of 2 from s in H. Therefore, t_1 and t_2 are considered equally important to s based only on their distance. However, such a conclusion is irrational, since s and t_2 are connected by the shortest paths and hence are more relevant. In light of this, it is also essential to count the number of shortest paths between any two specified vertices.

Even with a not-so-"small" diameter (the longest shortest path), the distance is also uninformative. Consider a connected unweighted graph G with n vertices and a diameter of d. Given a vertex v, it could only differentiate d types of vertices if basing merely on distance. In the most of real networks, e.g., SNAP [22], $d \ll n$.

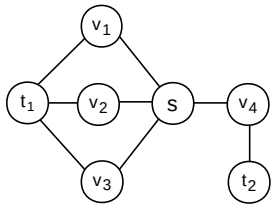


Fig. 1: Graph H

Application. Listed below are some vital applications for the SPC problem.

(1) *Group Betweenness*. The group betweenness exemplifies a typical application of the SPC problem. Puzis et al. [18] estimate the importance of a vertex set C to G based on the group betweenness. Let $P_{s,t}$ represent the set of shortest paths connecting vertices s and t, $spc(s,t)$ represent the number of shortest paths connecting vertices s and t, and $spc_C(s,t)$ indicate the # of the paths in $P_{s,t}$ via C. Group Betweenness of C, indicated by $\ddot{\mathrm{B}}(C)$, is defined as $\ddot{\mathrm{B}}(C) = \sum_{s,t} spc_C(s,t)/spc(s,t)$. As shown in [18], there is an algorithm GBC for progressively evaluating $\ddot{\mathrm{B}}(C)$. In particular, let $C = \{v_1, ..., v_{|C|}\}$ and let $C_i = \{v_1, ..., v_i\}$. GBC calculates $\ddot{\mathrm{B}}(C_i)$ in the i-th iteration by adding to $\ddot{\mathrm{B}}(C_{i-1})$ the total fraction of shortest paths that pass through v_i but not C_{i-1}. Therefore, after $|C|$ iterations, $\ddot{\mathrm{B}}(C)$ is found.

GBC receives as input three $|C| \times |C|$ matrices D, $\sum$, and $\widetilde{B}$, which store for $\forall x, y \in C$ the distance between x and y, $spc(x,y)$, and the path betweenness of (x,y), respectively. After computing $\ddot{\mathrm{B}}(C_i)$, GBC modifies $\widetilde{B}$ depending on D and $\sum$ such that $\widetilde{B}_{v_{i+1},v_{i+1}} = \ddot{\mathrm{B}}(C_{i+1}) - \ddot{\mathrm{B}}(C_i)$. This makes it simple to compute $\ddot{\mathrm{B}}(C_{i+1})$ in the subsequent iteration. Overall, GBC needs $O(T + |C|^3)$ time, where T is the construction time for D, $\sum$, and $\widetilde{B}$. In tasks such as estimating group betweenness distribution, the # of groups to evaluate is vast. To reduce the online time necessary to

generate D, $\sum$, and $\widetilde{B}$, [18] proposes to pre-compute and store the distance, the # of shortest paths, and the path betweenness for every pair of vertices, incurring prohibitive overhead. Although distance hub labeling and VC-dimension-based techniques [23] may minimize the cost associated with distance and path betweenness, the cost regarding SPC remains intractable.

(2) *Road Networks*. In real-world road network applications, the greater the number of shortest routes, the greater the number of traffic possibilities and the flexibility of route planning from the origin to the destination. For example, the top-k nearest neighbours search attempts to locate k objects adjacent to the query vertex inside a candidate set. It is a leading provider of taxi (e.g., Uber), restaurant (e.g., Tripadvisor), and hotel (e.g., Booking) recommendation services. A candidate item may be more desirable than others with the same or comparable distance if many shortest paths go to it, since we have more backup routing options and a greater chance of avoiding traffic congestion. In a movie ticket application, for instance, there are two theatres with the same shortest distance to the source location. Given the available traffic possibilities, we may choose the route with the most shortest paths. In addition to acting as a proximity measure, the shortest path count has been employed as a building component in the computation of betweenness centrality [18], [24].

Challenges. In the above application, the main obstacle is the increasing graph size of real applications. At the current stage, large-scale graphs are common for real applications. Nevertheless, the state-of-the-art algorithm for building an index based on node orders limits its scalability. To address this issue, we carefully designed a different scalable algorithm for this problem. Our experiments demonstrate that our algorithm could build an index for large-scale graphs. In addition, the query time is scalable.

Contributions. The primary contributions are listed below.

(1) Parallel Algorithm. This paper begins by analyzing the interdependence of the current hub labeling approach and explaining the challenge of parallel processing. It redesigns another propagation mechanism to avoid these dependencies by detecting the dependencies.

(2) Acceleration Optimizations. This paper investigates scheduling planning and landmark-based labeling to boost efficiency. Using these optimization techniques, our index creation phase might be accelerated significantly.

(3) Hybrid Vertex Ordering. Classical vertex orders for the SPC involve ordering by the degree and ordering by the significant path. In road networks, the significant-path-based ordering surpasses the degree-based ordering. Nevertheless, significant-path-based ordering must select the next vertex based on the shortest path trees built in the current vertex's construction, implying a dependency for each vertex and leading parallel processing challenging. This paper proposes a vertex-based ordering for the road network and a degree-based ordering for the social network to fill this gap. Then, it mixes them to provide a hybrid ordering for a scalable index construction process.

(4) Comprehensive Experimental Evaluation. In order to illustrate the effectiveness and efficiency of our algorithms, 10 datasets are employed in this study. The experimental results indicate that our method outperforms the baselines in terms of index building time and generates comparable index sizes. Furthermore, our method scales approximately linearly with the number of threads.

TABLE I: The summary of notations

Notation	Definition
$G = (V, E)$	a given undirected and unweighted graph
m, n	the number of edges(vertices) for G
$nbr(v)$	the set of neighbors of v
$deg(v)$	the degree of v
$len(p)$	the length of a path p $len(p) = l$ if $p = (v_0, v_1, ..., v_l)$
$rev(p)$	the reverse of a path p $rev(p) = (v_l, v_{l-1}, ..., v_0)$ if $p = (v_0, v_1, ..., v_l)$
$p_{s,t}$	a shortest path from s to t
$P_{s,t}$	the set of shortest paths from s to t
$SPC_{s,t}$	the shortest path counting from s to t
L^C, L^{NC}	the canonical and non-canonical labels
$dis(u, w)$	the distance from vertex u to w
$L(v)$	ESPC index for vertice v
$C_{v,w}$	the trough path counting from v to w
$rev(p)$	the reverse path of p

Roadmap. The rest of the paper is organized as follows. Section VI presents important related works. Section II introduces the preliminaries of the SPC problem. Section V experimentally evaluates our proposed approaches on real small-world networks and Section VII concludes the paper.

II. Preliminaries

Graphs. This paper concentrates on an unweighted and undirected graph denoted by $G = (V, E)$, where V and E represent the set of vertices and edges in G, respectively. Let $n = |V|$ and $m = |E|$ denote the number of vertices and the number of edges, respectively. For each vertex $v \in V$, let nbr(v) be the set of $v's$ neighbors and deg(v) be the degree of v. A path p from vertex s to vertex t is defined as a sequence of vertices $(s = v_0, v_1, ..., v_l = t)$ such that $(v_i, v_{i+1}) \in E$ for $0 \leq i < l$. The length of p, denoted by $len(p)$, is the number of edges included in p. In other words, $len(p) = l$. For simplicity, this work uses the notation $rev(p)$ to denote the reverse of a path. Specifically, $rev(p) = (v_l, v_{l-1}, ..., v_0)$. A path from s to t is shortest if its length is no larger than any other path from s to t.

The notations are summarized in Table I.

A. 2-Hop Labeling for Shortest Path Counting

To efficiently process point-to-point SPC queries, the 2-hop labeling technique [10] precomputes the SPC information from each node to pre-selected hub nodes and utilizes the 2-hop via hubs to respond to a query. It presents the hub labeling method for the shortest path counting between vertices s and t, SPC(s, t). The shortest path counting between vertices s and t in a directed graph seeks to determine the total number of all the shortest paths from s to t. [10] proposed a 2-hop labeling scheme and an index construction algorithm to build the index efficiently and enable real-time shortest path counting queries. The hub labeling scheme supports the cover constraint, Exact

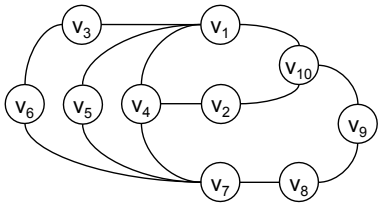


Fig. 2: An Undirected Graph. The total order $\leq$ is $v_1 \leq v_7 \leq v_4 \leq v_{10} \leq v_3 \leq v_5 \leq v_6 \leq v_2 \leq v_8 \leq v_9$.

TABLE II: Shortest Path Counting Labels of Fig. 2

Vertex	$L(\cdot)$
v_1	$(v_1, 0, 1)$
v_2	$(v_1, 2, 2)$ $(v_7, 2, 1)$ $(v_4, 1, 1)$ $(v_{10}, 1, 1)$ $(v_2, 0, 1)$
v_3	$(v_1, 1, 1)$ $(v_7, 2, 1)$ $(v_3, 0, 1)$
v_4	$(v_1, 1, 1)$ $(v_7, 1, 1)$ $(v_4, 0, 1)$
v_5	$(v_1, 1, 1)$ $(v_7, 1, 1)$ $(v_5, 0, 1)$
v_6	$(v_1, 2, 1)$ $(v_7, 1, 1)$ $(v_3, 1, 1)$ $(v_6, 0, 1)$
v_7	$(v_1, 2, 2)$ $(v_7, 0, 1)$
v_8	$(v_1, 3, 3)$ $(v_7, 1, 1)$ $(v_{10}, 2, 1)$ $(v_8, 0, 1)$
v_9	$(v_1, 2, 1)$ $(v_7, 2, 1)$ $(v_4, 3, 1)$ $(v_{10}, 1, 1)$ $(v_8, 1, 1)$ $(v_9, 0, 1)$
v_{10}	$(v_1, 1, 1)$ $(v_7, 3, 2)$ $(v_4, 2, 1)$ $(v_{10}, 0, 1)$

Shortest Path Covering (ESPC), which implies that it not only encodes the shortest distance between two vertices but also ensures that such shortest paths are correctly counted. HP-SPC is the algorithm developed for constructing the SPC label index that satisfies ESPC.

Formally, given a directed graph G, HP-SPC assigns each vertex $v \in G$ an in-label $L_{in}(v)$ and an out-label $L_{out}(v)$, consisting of entries of the form $(w, sd(v,w), \theta_{v,w})$. The shortest distance between v and w is denoted by $sd(v,w)$, and the number of shortest paths between v and w is denoted by $\theta_{v,w}$. If $w \in L_{in}(v)$ or $w \in L_{out}(v)$, then w is considered a hub of v.

In essence, the in-label $L_{in}(v)$ keeps track of the distance and counting information from its hubs to itself, whereas the out-label $L_{out}(v)$ records distance and counting information from v to its hubs. HP-SPC adheres to the cover constraint, which states that for each given starting vertex s and ending vertex t, there exists a vertex $w \in L_{out}(s) \cap L_{in}(t)$ that lies on the shortest path from s to t.

In addition, HP-SPC guarantees the correctness of shortest path counting by including the shortest path from s to t via a hub vertex once during the label construction. SPC(s,t) is evaluated by scanning the $L_{out}(s)$ and $L_{in}(t)$ for the shortest distance via common hubs and adding the multiplication of the corresponding count. Equation (1) identifies all common hubs (on the shortest paths) from $L_{out}(s)$ and $L_{in}(t)$. Equation (2) determines the result of SPC(s,t).

$$H = \{h | \underset{h \in L_{out}(s) \cap L_{in}(t)}{\arg\min} \{sd(s,h) + sd(h,t)\}\} \quad (1)$$

$$SPC(s,t) = \sum_{h \in H} \theta(s,h) \cdot \theta(h,t) \quad (2)$$

Example 1. *Figure 2 depicts an undirected graph with* 10 *vertices, and Table II provides its hub labeling index for SPC queries. SPC(v_{10}, v_7) is used as an example to determine the shortest paths counting from v_{10} to v_7. By scanning $L(v_{10})$ and $L(v_7)$, two common hubs $\{v_1, v_7\}$ are found. The shortest distance through v_1 is $2+2=4$, whereas the counting is $2 \cdot 1 = 2$; The shortest distance via v_7 is $3+0=3$, and the counting is $1 \cdot 2 = 2$. Therefore, the number of shortest paths from v_{10} to v_7 is $2+2=4$ with a length of* 4.

III. PSPC ALGORITHM DESCRIPTION FROM THE COVERING SIDE

This section presents our parallel method for the SPC problem. First, the *Shortest Path Covering* is defined.

Definition 1. *(Shortest Path Covering). $T(v)$ represents a set of entries of the form $(w, C_{v,w})$, where $C_{v,w} \subset P_{v,w}$ represents a subset of the shortest paths from v to w. For each pair of vertices u and v, the shortest paths are covered by $T(u)$ and $T(v)$ as a multiset as follows:*

$$\begin{aligned} cover(T(u), T(v)) = \\ \{p_1 \odot rev(p_2) | (w, C_{u,w}) \in T(u), (w, C_{v,w}) \in T(v), \\ p_1 \in C_{u,w}, p_2 \in C_{v,w}, \\ sd(u,w) + sd(v,w) = sd(u,v)\} \end{aligned} \quad (3)$$

It is noted that w is the common vertex in both $T(u)$ and $T(v)$.

This is followed by the definition of *Exact Shortest Path Covering*.

Definition 2. *(Exact Shortest Path Covering). $T(\cdot)$ is an exact shortest path covering (ESPC for short), which means that for any two vertices u and v, the multiset cover $(T(u), T(v))$ must be identical to $P_{u,v}$, i.e., the set of shortest paths between u and v.*

The Construction of an ESPC Index. The construction process of an ESPC is explained. Let $\leq$ be a total order over V. A trough path [25] is a path whose endpoint is ranked higher than all the other vertices. A trough shortest path is a path that is both trough and shortest. For a shortest path $p \in SP_{u,v}$, there exists a vertex w with the highest rank. Thus, p could be divided into two trough shortest paths $p_{u,w}$ and $p_{w,v}$. By doing this, we could find the exact shortest path covering by using the trough shortest paths of u and v.

Consider, for instance, the graph G' in Figure 2, and a total order $\leq$ where $v_1 \leq v_7 \leq v_4 \leq v_{10} \leq v_3 \leq v_5 \leq v_6 \leq v_2 \leq v_8 \leq v_9$. The path (v_3, v_1, v_{10}) is not a trough path since v_1 has a higher rank than both endpoints v_3 and v_{10}. The path (v_6, v_3, v_1) is the trough shortest path because one endpoint (i.e. v_1) has the highest rank and the path is the shortest. Given a total order $\leq$ over the vertices, an ESPC can be constructed as follows. $T(v)$ is initially empty for each vertex v. Then, for any two (potentially identical) vertices v and w with $w \leq v$,

an entry $(w, C_{v,w})$ is added to $T(v)$, where $C_{v,w}$ is the set of all trough shortest paths from v to w. This $C_{v,w}$ is not empty. Note that for every such label entry, since $w \leq v$, w has the highest rank in p for each path $p \in C_{v,w}$. $T_{\leq}(\cdot)$ and $L_{\leq}(\cdot)$ denote the $T(\cdot)$ constructed this way and the corresponding $L(\cdot)$, respectively. The ESPC result in Figure 2 is shown in Table II.

Then, this work investigates why the state-of-the-art algorithm [10] cannot be parallelized. In the state-of-the-art algorithm, the label entries are separated into two types: canonical labels (L^c) and non-canonical labels (L^{nc}).

$p_{s,t}$ denotes a shortest path in G from s to t. The shortest distance between s and t in G, denoted by $sd_G(s,t)$ is defined as the length of the shortest path between s and t in G. The set of all shortest paths from s to t is denoted by $P_{s,t}$, and the set of the vertices involved in $P_{s,t}$ is denoted by $Q_{s,t}$. $spc_G(s,t)$ denotes the number of shortest paths from s to t in G. When the context is clear, $sd(s,t)$ and $spc(s,t)$ are used instead of $sd_G(s,t)$ and $spc_G(s,t)$ for simplicity. Let $\leq$ be a total order over V. For two distinct vertices w and v, if $w \leq v$, then w has a higher rank than v. The total order of vertices is an order that the ESPC index constructs for each vertex. It could be obtained by ranking all the vertices in any order, e.g., sorting the vertices by the degree order.

[10] defined the Canonical Hubs as follows: given a total order $\leq$ over the vertices, is one that comprises the following hubs: For two vertices v and w, $w \in L(v)$ if and only if w is the highest-ranked vertex in $Q_{v,w}$.

Definition 3. *(Canonical Hubs). A canonical hub labeling, given a total order $\leq$ over the vertices, is one that comprises the following hubs: For two vertices v and w, $w \in L(v)$ if and only if w is the highest-ranked vertex in $Q_{v,w}$.*

Thus, non-canonical hubs are those hubs that do not comply with the definition of Canonical hubs and also in ESPC.

A. Trough Path Property

The labels of L^c demonstrate an essential node-order characteristic.

Theorem 1. *For each pair of vertices $\forall u, v \in V$, v is a hub of u in L^c, i.e., $(v, dis(v,u), c(v,u)) \in L^c(u)$, if and only if paths $SP(u,...,v)$ are the trough paths between u and v, i.e., v is the highest-ranked node along all the shortest paths from u to v.*

Proof. This property is proved by contradiction. $SP(u,..,v)$ represents the set of all shortest paths from u to v. Consider a node u_h that is the highest-ranked node in $SP(u,...,v)$. Assume that there exists a node u_c in $SP(u,...,v)$ such that u_c does not have a hub of u_h in the label set. $L^c_{<u_h}$ denotes the label index when finishing the pruned BFS for all vertices whose rank is higher than u_h. Consider the construction of ESPC, and the iteration when the pruned BFS sourced u_h is performing. When there is no hub of u_h for u_c in this iteration, then either

- Query$(u_c, u_h, L^c_{<u_h}) = (d_1, c_1)$ and $d_1 < dis(u_c, u_h)$, or
- u_c is not explored in the Pruned BFS sourced from u_h, indicating that there is a vertex u'_c on the shortest paths from u_c to u_h and could be pruned with Query$(u'_c, u_h, L^c_{<u_h} = (d_2, c_2)$) with $d_2 < dis(u_h, u'_c)$.

In either scenario, it needs a common hub between u_h and u_c to produce the query result. Otherwise, some shortest paths would be omitted from in our index, which would violate the definition of ESPC. Nevertheless, such a common hub cannot exist since i) $u_c, u'_c \in SP(u,...,v)$ and ii) u_h is the highest-ranked vertex in $SP(u,...,v)$. This results in a contradiction.

Since all nodes in $SP(u,...,v)$ have u_h as their hubs, the theorem can be proved in the two cases: i) $u_h = v$, i.e., v is the highest-ranked vertex in $SP(u,...,v)$, the v is a hub of u and ii) if $r(u_h) > r(v)$, when before the pruned BFS sourced from v is performed, u_h is already a common hub of u and v. Since u_h is on the shortest path between u and v, the label with hub v on u is inserted into the cL or pruned. □

B. Order Property

The labels are partitioned in the index according to their hub nodes to see the dependency among the labels. Let $v_1 \leq v_2 \leq ... \leq v_n$ represent the node order under which label set the index was constructed.

Two distinct sets are defined. Recall that the HP-SPC$_s$ index consists of n iterations where the i-th iteration executes a pruned BFS sourced from v_i. $L^{SPC}_{<i}(u)$ is the snapshot of $L^{SPC}(u)$ at the beginning of the i-th iteration, and by $L^{SPC}_i(u)$ the incremental label of u built in the i-th iteration. It is notable that $L^{SPC}_{<i}(u) = L^c_{<i}(u) \cup L^{nc}_{<i}(u)$

Definition 4. *(Order Specific Label Set). $L^{SPC}(u) = (v_i, dis(v_i,u), c) \in L^{SPC}$, for $\forall i \in [1,n]$, $u \in V$. Let $L^{SPC}_i = \bigcup_{u \in V} L^{SPC}_i(u)$.*

Definition 5. *(Order Partial Label Set). $L^{SPC}_{<i}(u) = (v_j, dis(v_j,u), c) \in L^{SPC} | j < i$ for $\forall i \in [1, n+1]$, $u \in V$. Let $L^{SPC}_{<i} = \bigcup_{u \in V} L^{SPC}_{<i}$. $L^{SPC}_{<n+1} = L^{SPC}$.*

The following lemma demonstrates that the pruning condition in HP-SPC$_s$ results in an order dependency among labels.

Lemma 1 (Order Dependency)**.** *The L^{SPC}_i would depend on $L^{SPC}_{<i}(u)$. Specifically, $L^{SPC}_i(u)$ would be updated as follows:*

- *If Query$(v_i, u, L^{SPC}_{<i}) = (d_1, c_1)$, and $d_1 > dis(v_i,u)$, then $L^c_i(u) = (v_i, dis(v_i,u), count(v_i,u))$.*
- *If Query$(v_i, u, L^{SPC}_{<i}) = (d_1, c_1)$, and $d_1 = dis(v_i,u)$, then $L^{nc}_i(u) = (v_i, dis(v_i,u), count(v_i,u) + c_1)$.*
- *Otherwise, $L^{SPC}_i(u) = \emptyset$.*

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 in L^c and ii) for $\forall v \in S$ v_i, v is not a hub of v_i in L^{SPC}, and hence, if Query$(v_i, u, L^{SPC}_{<i} = (d_1, c_1))$, then $d_1 > dis(v_i,u)$. If $r(v_i) < r(w)$, then v_i is not a hub of u in L^c and label $(w, dis(w,v_i), d_{w1}), (w, dis(w,u), d_{w2}) \in L^{SPC}_{<i}$ and thus Query$(v_i, u, L^{SPC}_{<i}) = (d_2, c_2)$, then $d_2 = dis(v_i,u)$. It is only necessary to update $L^{nc}_{\leq i}$ in the i-th iteration. □

Lemma 1 demonstrates that $L_i^{SPC}(u)$ depends on $L_{<i}^{SPC}$ while $L_{<i}^{SPC}(u)$ depends on L_{i-1}^{SPC}. Such a convolved dependency is difficult to remove so long as the labels are constructed in the node order.

C. Distance Dependency

To break the order dependency in the label construction, we consider the pruning condition where Query$(v_i, u, L_{<i}^{SPC}) = (dis(u, v_i), count(u, v_i))$. It prunes a node label on u, and there must be two labels on u and v_i to a common hub w such that $dis(u, w) + dis(w, v_i) < dis(u, v_i)$. Consequently, $dis(u, w)$ and $dis(w, v_i)$ must be smaller than $dis(u, v_i)$. In other words, none of the labels with distances greater than $dis(u, v_i)$ influence the query result of Query$(v_i, u, L_{<i}^{SPC})$ and the corresponding pruning outcomes.

Based on the above intuition, the label entries in L^{SPC} are categorized by their label distances. The reorganized label sets will pave the way to our Parallel Shortest Path Counting Labeling approach and are hence referred to as PSPC label sets. Let D be the diameter of graph G.

Definition 6. *(Distance Specific Label Set).* $L_d^{PSPC}(u) = (u, dis(v, u), c) \in L^{SPC}(v) | dis(v, u) = d$, *for* $\forall u \in V, d \in [1, D]$. *Let* $L_d^{PSPC} = L_d^{SPC}(u) | u \in V$.

Similarly, the partial label of a node then becomes the set of label entries with a distance no larger than a certain distance and is defined in Definition 7.

Definition 7. *(Distance Partial Label Set).* $L_{\leq d}^{PSPC}(u) = (v, dis(v, u), c) \in L^{SPC}(u) | dis(v, u) \leq d$, *for* $\forall u \in V$, $d \in [1, D + 1]$. *Let* $L^{PSPC} = \bigcup_{u \in V} L^{PSPC}(u)$. *In particular,* $L^{PSPC}(u) = L_{\leq D+1}^{PSPC}(u)$.

Theorem 2 describes the equivalence between the index L^{SPC} and the new index L^{PSPC}.

Theorem 2. $L^{SPC} = L^{PSPC}$.

Proof. Since each label $(v, dis(u, v), c)$ in L^{SPC} has $dis(v, u) \leq D$, L^{PSPC} contains all labels in L^{SPC} and, by definition, no additional labels. □

Distance Dependency. Definitions 6 and 7 provide us the option to eliminate the order dependency in the label construction process.

Theorem 3. $L_d^{PSPC}(u)$ *relies on* $L_{\leq d}^{SPC}$. *Specifically, given a node* u, *for a node* $v \in V$ *with* $r(v) > r(u)$ *and* $dis(u, v) = d$, $(v, d, count(v, u)) \in L_d^{PSPC}(u) \in L_d^{SPC}(u)$ *if and only if* Query$(u, v, L_{\leq d}^{PSPC}) = (d_0, c_0)$ *and* $d_0 > d$

Proof. Consider a node v with $dis(u, v) = d$. S denotes 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, there are two cases that are exclusive:

1) $w = v$ iff v is the hub of i in L^c.
2) $w \neq v$ indicates that
   a) w is the hub of both u and v, and
   b) $dis(u, w), dis(w, v) \leq d$ and hence, Query$(u, v, L_{\leq d}^{PSPC}) = (d_0, c_0)$ and $d_0 = d$.

Therefore, if $(v, dis(v, u), c) \notin L_d^{PSCP}(u)$, namely, v is not a hub of u, then $w \neq v$, and then $Query(u, v, L_{\leq d}^{PSCP}) = (d_0, c_0)$ and $d_0 = d$. Besides, if $(v, dis(v, u), c_0) \in L_d^{PSPC}(u)$, 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, $Query(u, v, L_{\leq d}^{PSPC}) = (d_0, c_0)$ and $d_0 > d$. □

By transforming the order dependency to distance dependency, the index may be constructed in D iterations, where D denotes the diameter of the graph.

D. The Parallelized Labeling Method

To apply Theorem 3 to construct $L_d^{PSPC}(u)$, it is costly to examine all the node pairs with a distance equal to d. This section provides a practical algorithm, Parallel Shortest Path Counting (PSPC), to construct the index L^{PSPC} in label propagation.

Propagation-Based Label Construction. This subsection provides a positive answer to the following question: can $L_d^{PSPC}(u)$ be built by gathering the labels of its neighbors, namely, $L_{d-1}^{PSPC}(v)$ is sufficient to create $L_d^{PSPC}(u)$ in Lemma 2.

Lemma 2. *All the hub nodes of labels in* $L_d^{PSPC}(u)$ *appear in labels* $\bigcup_{v \in N(u)} L_{d-1}^{PSPC}(v)$ *as hub nodes.*

Proof. It is shown that if a node is not a hub of any node $v \in N(u)$ in $L_{d-1}^{PSPC}(v)$, then it is not a hub of u in $L_d^{PSPC}(u)$. Let $w \neq u$ be a hub of u in $L_d^{PSPC}(u)$ but is not a hub of any node $v \in N(u)$ in $L_{d-1}^{PSPC}(v)$. Note that the SPC was built in a BFS search. Consider the iteration 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 $(w, dis(w, s), c_{w,s}) \in L^{SPC}$. Since $dis(w, s) = d - 1$, w is a hub of $L_{d-1}^{PSCP}(s)$, this leads to the contradiction. □

Pruning Conditions. According to Lemma 2, $L^{PSPC}(u)$ may be constructed iteratively, with the initial condition being the insertion of u into the label $L_0^{SPC}(u)$ as its own hub. Nevertheless, pouring all nodes in $\bigcup_{v \in N(u)} L_{d-1}^{SPC}(v)$ directly into $L_d^{SPC}(u)$ produces a large set of candidate labels. Consequently, two rules are proposed to prune superfluous label entries.

Lemma 3. *A hub* w *in the label set* $\bigcup_{v \in N(u)} L_{d-1}^{PSPC}(v)$ *is not a hub of* u *if* $r(w) < r(u)$.

Lemma 4. *A hub* w *in the label set* $\bigcup_{v \in N(u)} L_{d-1}^{PSPC}(v)$ *is not a hub of* u *in* $L_d^{PSPC}(v)$ *if* $Query(w, u, L_{\leq d}^{PSPC}) = (d_0, c_0)$ *and* $d_0 < d$.

Proof. If $Query(w, u, L_{\leq d}^{PSPC}) = (d_0, c_0)$, and $d_0 < d$, then $dis(w, u) \neq d$, w is not a hub of u with distance $dis(w, u) = d$. If $Query(w, u, L_{\leq d}^{PSPC}) = d$, two situations are discussed:

1) $dis(w, u) < d$, w is not a hub of u with distance d.
2) $dis(w, u) = d$, 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 a hub of u in L^c. Then, these newly found shortest paths could be added into L^{nc}.

Therefore, w is not a hub of u if $Query(w, u, L_{\leq d}^{PSPC}) = (d_0, c_0)$ and $d_0 < d$. □

Based on the above pruning rules, the label propagation function is proposed to find the exact $L_d^{PSPC}(u), \forall u \in V$.

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

Definition 8 (Label Propagation Function)**.** $L_d^{PSPC}(u) = \bigcup_{w \in C_{d-1}(v), for \forall v \in N(u)} L_d^{PSPC}(u, w)$ *where* $L_d^{PSPC}(u) =$

$$\begin{cases} \emptyset \ \ if\ r(w) < r(u)\ or \\ Query(w, u, L_{\leq d}^{PSPC}) = (d_0, c_0),\ and\ d_0 < d; \\ (w, dis(w, u)) \quad otherwise \end{cases} \tag{4}$$

Proof. L' denote the label set computed from Equation 4. It is shown that $L' = L_d^{PSPC}(u)$ in two directions. Due to the correctness of Lemma 2, and the pruning conditions, the label set $L_d^{PSPC}(u) \subset L'$. The following parts demonstrate $L' \subset L_d^{PSPC}(u)$. Let $(w, dis(w, u)$ be a label in L'. Equation 4 shows that $r(w) > r(u)$ and $Query(w, u, L_{\leq d}^{PSPC}) \geq d$.

If in L^c, w is not a hub of u, then according to Theorem 1, there exists 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, $dis(w, s), dis(s, u) < d$ and $dis(w, u) \leq d$, and consequently, $Query(w, u, L_{\leq d}^{PSPC}) < d$, this leads to the contradiction.

Therefore, w is a hub of u in L^{SPC}. Additionally, if $dis(w, u) < d$, $Query(w, u, L_{\leq d}^{PSPC}) = (d_0, c_0)$ and $d_0 < d$, there is a contradiction. Thus, $dis(w, u) = d$. Now, it has been proved that w is a hub of u in L^{SPC} with $dis(w, u) = d$, i.e., w is a hub of u in $L_d^{PSPC}(u)$ which completes the proof. □

E. Propagation Paradigms

There are two paradigms for label propagation. The first one is *Push-Based* paradigm, whereas the second one is *Pull-Based* paradigm. Then, the benefits and drawbacks are discussed.

Definition 9 (Push-Based Label Propagation)**.** *In i^{th} iteration, vertex v propagates its label entries to all of its out-neighbors.*

The Algorithm is illustrated in the following manner: Line 1 tackles all the vertices in parallel. For each vertex $v \in V$, it pushes its in-neighbors in Line 2. Followed, Line 3 inserts all the candidates' hubs to $\mathcal{C}(u)$. Line 5 eliminates the duplicate candidates. Line 6 traverses each element in the $\mathcal{C}(u)$ recursively. Then, two pruning conditions are validated in Lines 7 and 9, respectively. If not pruned, this index entry would be inserted into $L_d^{SPC}(u)$ in Line 11.

Definition 10 (Pull-Based Label Propagation)**.** *In i^{th} iteration, vertex v receives all the label entries from all of its in-neighbors.*

The details of Algorithm 2 are illustrated as follows: Line 1 tackles all the vertices in parallel. For each vertex $v \in V$, it pulls its in-neighbors in Line 2. Followed, Line 3 inserts all the candidates' hubs to $\mathcal{C}(u)$. Line 4 removes the duplicate candidates. Line 5 traverses each element in the $\mathcal{C}(u)$ recursively. Then, in Lines 6 and 8, two pruning conditions are validated, respectively. This index entry would be inserted into $L_d^{SPC}(u)$ in Line 10 if not pruned.

Algorithm 1: PUSH(G, L_{d-1}^{SPC})

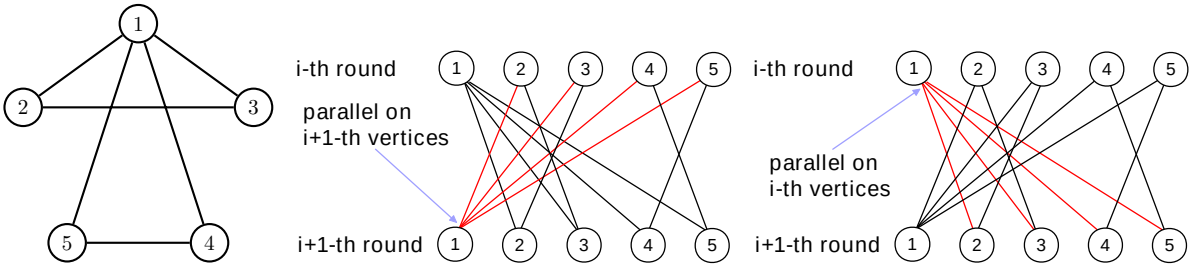
```
1  for each u ∈ V in parallel do
2      for each v_k ∈ G_out(u) do
3          C(v_k) ← hubs ∈ L_{d-1}^{SPC}[u] ;
4  for each u ∈ V in parallel do
5      Remove the duplicates in C ;
6      for each c ∈ C(u) do
7          if r(c.v) < r(u) then
8              continue;
9          if Query(c.v, u, L_d^{SPC}) < d then
10             continue;
11         Insert(c) into L_d^{SPC}(u);
```

Algorithm 2: PULL(G, L_{d-1}^{SPC})

```
1  for each u ∈ V in parallel do
2      for each v_k ∈ G_in(v_i) do
3          C(u) ← hubs ∈ L_{d-1}^{SPC}[v_k] ;
4      Remove the duplicates in C ;
5      for each c ∈ C(u) do
6          if r(c.v) < r(u) then
7              continue;
8          if Query(c.v, u, L_d^{SPC}) < d then
9              continue;
10         Insert(c) into L_d^{SPC}(u);
```

Example 2. *Figure 3 depicts an example of the difference between PULL-based and PUSH-based propagation paradigms. In the original graph G in Figure 3(a), there are 5 vertices and 6 edges. In the $(i+1)$-th iteration of the index construction, Figure 3(b) indicates how pull-based method works. For instance, vertex 1 received the i-th iteration's index entry from all of its neighbors (in-neighbors in directed graphs), i.e., vertices 2, 3, 4, and 5. If available, each vertex of the $(i+1)$-th iteration in graph 3(b) may be assigned with one thread. It is not necessary to allocate each edge to a single thread since there are several erroneous index entries that may be merged or removed to accelerate the process. Figure 3(c) demonstrates how the push-based method runs. In contrast to the pull-based, the i-th iteration's vertex processing can be parallelize at the vertex level.*

Candidates Elimination. In this part, the duplicate removal method is briefly discussed. The reason is that there are numerous duplicate candidates for one vertex. In shortest path counting, the counting number could be fairly large. Then, one vertex v would receive all of its neighbors' label entries in a single iteration. In such a scenario, there could be multiple



(a) Original Graph G (b) PULL (c) PUSH

Fig. 3: Pull-based and Push-based Paradigms.

duplicate candidates. If they were not merged, the overall computation cost would be prohibitively expensive. Thus, this part investigates the method for eliminating duplicate candidates.

The PULL-based paradigm is employed to simplify. To reduce the potential label entries for each label entry (v, d, c) received by vertex u, there are primarily two types of operations, named *Label Elimination* and *Label Merging*.

Two pruning rules are proposed to speed up this index construction process.

- **Label Elimination.** For two label entries L_1 (v_1, d_1, c_1) and L_2 (v_2, d_2, c_2) for vertex u, (v_1, d_1, c_1), if $v_1 = v_2 \wedge d_1 < d_2$, then L_2 is eliminated by L_1.
- **Label Merging.** For two label entries L_1 (v_1, d_1, c_1) and L_2 (v_2, d_2, c_2) for vertex u, (v_1, d_1, c_1), if $v_1 = v_2 \wedge d_1 = d_2$, then L_1 and L_2 could be merged into a L_3 $(v_1, d_1, c_1 + c_2)$.

Lemma 5. *These two pruning rules Label Elimination and Label Merging do not affect the correctness of the parallel shortest path counting algorithm.*

Proof. We simply prove its correctness by contradiction. Due to the space limit, we only prove the correctness of *Label Elimination*, while the proof for *Label Merging* is similar. Assume Label Elimination missed a label entry L_o (v_o, d_o, c_o), which is caused by the elimination of L_2 (v_2, d_2, c_2). Thus, there must exist a label entry L'_o $(v_o, d_o - d_2 + d_1, c'_o)$ by replacing the L_2 parts with L_1. This contradicts the fact that L_o (v_o, d_o, c_o) is necessary. □

Time and Space Complexity. Since these two types of operations are based on $v_1 = v_2$, (v, d, c) can be stored into map by using v as the key. For the label entries with the same key, it is necessary to sort them and then conduct *Label Elimination* and *Label Merging*. Assume there are η candidates, the worst-case time complexity would be $O(\eta \times log\eta)$. The worst case of η is the number of vertices whose rank is lower than the current vertex.

F. Schedule Plan

A basic objective of parallel index construction is to balance workloads. This subsection investigates how to allocate tasks in the most equitable manner feasible.

Node-Order-Based Schedule. Assume there are n vertices and t threads, $order[i]$ indicates the vertex id which ranks i. In our index construction algorithm, tasks are allocated according to the node order. In each iteration for distance d, the $t_i \in [0, t-1]$ thread would cope with all the tasks for vertex whose order ranges in $[t_i \times \lfloor n/t \rfloor, (t_i + 1) \times \lfloor n/t \rfloor)$.

Fig. 4: Node-order based schedule.

Example 3. *Figure 4 illustrates an example of how the node-order based schedule works. There are* 20 *vertices and* 7 *threads. Consequently, threads* 0 *would cope with vertices with order values in* $(0, 1, 2)$*. Such a schedule plan would result in an imbalance between vertices. For instance, in Pull-Based Paradigm, vertices with order* 0 *would receive no candidates according to Lemma 3.*

Cost-Function-Based Dynamic Schedule. Motivated by Example 3, such a schedule could cause an imbalance of tasks for threads. Therefore, this part proposes a dynamic cost-function-based schedule for improved scalability.

Definition 11. $cost(v_i) = \sum_{v_j \in Nbr(v_i)} \{|u| | u \in L_{v_j}$ *and* $r(v(u)) < r(v_i)\}$

Since the precise cost function is expensive to calculate, an approximation-based approach is proposed for measuring the cost and the schedule. In addition, rather than statically assigning each thread with an equal amount of tasks, it dynamically allocates tasks when tasks on it finish.

G. Vertex Ordering Strategies

The vertex order is crucial for both HP-SPC$_s$ and PSPC since it has a significant impact on indexing time, index size, and the query time. An efficient ordering scheme should rank vertices that cover more shortest paths higher s.t. later searches in HP-SPC$_s$ can be reduced as early as possible, thereby reducing the number of label entries generated. In the literature, several heuristics for obtaining such orderings have been investigated. For the sake of completeness, two state-of-the-art schemes are reviewed, namely, degree-based and significant-path-based, below.

Degree-Based Scheme. Vertices are arranged in ascending degree order. This technique is based on the premise that vertices with a higher degree have stronger connections to many other vertices, and as a result, many shortest paths will pass through them.

Significant-Path-Based Scheme. The significant-path-based method is more adaptable than the degree-based scheme, which utilizes only local information. Let w_1, w_2, ..., w_n be the ordering generated under this scheme, where w_i is the i-th hub to be pushed. Given w_i, the scheme determines w_{i+1} as follows. When pushing hub w_i in SPC, a partial shortest-path tree T_{w_i} rooted at w_i will be produced. For each vertex v in T_{w_i}, let $des(v)$ be the number of descendants and par(v) be the parent of v. Beginning with w_i, the scheme computes a significant path P_{sig} to a leaf by iteratively selecting a child v with the largest des(v). p_{sig} is intuitively a path that many shortest paths intersect. Then, among all vertices on p_{sig} other than w_i, vertex v with the largest deg(v) $\cdot$ $(des(par(v)) - des(v))$ is empirically selected as w_{i+1}. w_1 is initially configured to have the highest degree in G. The Significant-Path-Based Scheme is the most efficient order in [10]. Nevertheless, as stated before, this ordering needs to compute the next vertex based on the current vertex's shortest path tree, which naturally includes a dependency on the index construction process and hence is not suited for parallel processing.

Halin [26], and Robertson et al. [27] introduced tree decomposition, which is a technique for mapping a graph to a tree in order to expedite the resolution of certain computational problems in graphs. Bodlaender's introductory survey can be found in [28]. Vertices are naturally arranged in a hierarchy through tree decomposition. Therefore, this paper utilizes tree decomposition to create the vertex hierarchy, and demonstrate that the effectiveness of this hierarchy in answering shortest path counting queries in a road network. Given a graph $G(V, E)$, a tree decomposition of it is defined as follows [28]:

Definition 12 (Tree Decomposition)**.** *A tree decomposition of a graph $G(V, E)$, denoted as T_G, is a rooted tree in which each node $X \in V(T_G)$ is a subset of $V(G)$ (i.e., $X \subset V(G)$) such that the following three conditions hold:*

- $\bigcup_{X \in V(T_G)} X = V$;
- *For every $(u, v) \in E(G)$, there exists $X \in V(T_G)$ s.t. $u \in X$ and $v \in X$.*
- *For every $v \in V(G)$ the set $\{X | v \in X\}$ forms a connected subtree of T_G.*

Road Network Order. In the road networks, the degree order is inefficient since there are two many low-degree vertices with the same degree. Thus, a Tree Decomposition method is proposed in [29] to address the shortest distance queries. For instance, there are $v_1, ..., v_n$ vertices and a vertex order is demanded. This technique also has a natural dependency structure. The main steps of obtaining this order could be listed as follows:

- Set $Q = \emptyset$ as a queue. In this first iteration, vertex u_0 with the lowest degree would be inserted into Q. Then, u_0 is removed from this graph, for vertex $u \in Neighbor(u_0)$, this step connects them and update their degree as $deg(u) + deg(u_0) - 1$.
- Likewise, in the i-th iteration, the u_i (the lowest degree vertex in the current graph) are pushed into the Q. Then, u_i is removed from this graph, for vertex $u \in Neighbor(u_i)$, this step connects them and update their degree as $deg(u) + deg(u_i) - 1$.
- This step could produce a resultant vertex order by append vertices in Q into the R from the back of the queue to the front.

Then, the original graph could be divided into core- and fringe-parts, and employ a hybrid vertex ordering.

Hybrid Vertex Ordering. Therefore, a hybrid vertex ordering is proposed, which compromises between the computational efficiency of degree vertex order and the index size effectiveness of the road network order. All vertices are divided into two categories, core-part, and fringe-part. To achieve this, a threshold δ is set for the node degree. If a vertex v, has a degree larger than δ, then v would be divided into core-part. Otherwise, it would be divided into the fringe-part. Each part would utilize the corresponding order.

H. Landmark-Based Filtering

This subsection introduces landmark pruning. Landmark-based techniques are proved to be efficient in the Label Constrained Reachability problem [30]. The overall idea is to choose small groupings of vertices as landmarks and construct some small indexes on them to accelerate the whole index construction process. This work follows the strategies in [30] to select landmarks. To begin, the definition of landmarks is introduced.

Definition 13. *(Landmark). Given a threshold θ, a vertex v is a landmark if and only if $degree(v) \geq \theta$.*

In the index construction process, the distances from landmarks are frequently used. Based on this observation, this work proposes a landmark-based filtering to further expedite the index construction process.

In the classical shortest path counting with a 2-hop labeling index, vertices are explored individually. It processes vertex serially. For a vertex u, once it is explored, there is no need to utilize its distance information due to the pruning rule 1. Thus, there is no need to construct such a landmark index.

Nevertheless, the facts alter for our parallel algorithm. Regarding the parallel algorithm, the label propagation is divided by distance iteration. Since the degree of the landmarks is quite high, the labels from the landmarks would be the majority in each iteration. Consequently, if the distance information is stored from these landmarks, these queries could be instantly answered if landmarks are involved.

Furthermore, since all the distances are in increasing order. One bit is needed to store the "True" or "False" information, and it is sufficient to answer queries for pruning.

IV. Index Size Reduction

Two index reduction techniques proposed in [10] are analyzed in the context of parallel. They are 1-shell reduction and neighborhood-equivalence reduction to reduce the index size.

A. Reduction by 1-Shell

Shortest path counting problem is simple in the tree: there is only one shortest path between any two vertices in it. Despite the fact that graphs are often far more complex than trees, trees are more common. Graph G can always be transformed into a core-fringe structure. If it is not empty, it consists of trees, as will be shown following. Additionally, each of these trees connects to the remaining of G with no more than one edge. Consequently, it is safe to trim $G's$ fringe without affecting the shortest paths inside the core. In this situation, the 1-shell of G is characterized by the fringe. Specifically, the 1-shell of G is defined as the maximum subgraph within which each vertex is incident to at least k edges. The k-core of G is defined as the largest subgraph within which each vertex is incident to at least k edges.

During the index construction process of our parallel algorithm, the graph could be divided into a core-fringe structure. Regarding the fringe structure consisting of trees, they can be deleted from the original graph and utilize them during the query procedure. Therefore, the Reduction by 1-Shell technique does not influence our parallel paradigm.

Query Evaluation in Parallel. A query(s, t) is processed in the following manner. If $shr(s) = shr(t)$, 1 is directly returned; otherwise, a query$(shr(s), shr(t))$ is issued on $G(s)$ and its result is returned.

B. Reduction by Equivalence Relation

Given an undirected graph G and two vertices u and v, it is straightforward to demonstrate that $spc_G(u, w) = spc_G(v, w)$ for $\forall w \neq u, v$ if u and v share the same neighborhoods. In fact, a shortest path from u to w can be modified with a shortest path from v to w by substituting u for v, and vice versa. In this situation, the label of v alone would serve to answer any query involving $\{u, v\}$, and other vertices. In other words, it is permissible to eliminate the label of u, hence reducing the size of the index. u is neighborhood equivalent to v, indicated by $u \equiv v$, if $nbr(u)\ \{v\} = nbr(v)\ \{u\}$. In other words, if u and v are not adjacent, they must have the same set of neighbors; otherwise, the neighbors of u and v must be identical after eliminating u and v. It is feasible to demonstrate that is an equivalence relation. This equivalence has been implemented in graph reduction tasks such as subgraph isomorphism [31] and distance hub labeling [32]. As will be shown, the relation may also be utilized to reduce the graph size. However, straight application without adjustment might result in findings that are grossly underestimated.

During the index creation phase of our parallel approach, vertices with an Equivalence Relation are eliminated, leaving a single vertex to represent them. A weight is assigned to it depending on the quantity of equivalents.

Query Evaluation. Next, two query schemes are discussed for handling a query (s, t). Assume w.l.o.g. that $s, t \in I$ and $s \neq t$. In this case, $Rs = nbr(s)$ and $Rt = nbr(t)$.

As for the query process, it could also speed up with a parallel framework. It could be parallel twofold: i) Since each query is independent of the other, it is natural to dynamically assign the query to the available thread. ii) Since the query is a set intersection, it could parallelize at the label entry granularity. Then, all the counting of the shortest path is summed up.

V. Experimental Results

This section evaluates the effectiveness and efficiency of the proposed techniques on comprehensive experiments.

A. Experimental Settings

Algorithms. We compared the proposed algorithms with baseline solutions.

- HP-SPC$_s$. The state-of-the-art shortest path counting algorithm proposed in [10].
- PSPC. Our parallel shortest path counting algorithm in a single thread.
- PSCP$^+$. Our parallel shortest path counting algorithm in 20 threads.

Datasets Table III shows the important statistics of real graphs used in the experiments. 10 publicly available datasets are used. The largest dataset Indochina is a web graph from LAW[1]. The remaining 9 graphs, downloaded from KONECT [33][2] and SNAP [34][3], include an interaction network (WikiConflict), a coauthorship network (DBLP), a location-based social network (Gowalla), 2 web networks (Berkstan and Google), and 4 social networks (Facebook, Youtube, Petster, and Flickr). All of the graphs are unweighted. Directed graphs were converted to undirected ones in our testings. For query performance evaluation, 10,000 random queries were employed, and the average time is reported.

Settings In experiments, all programs were implemented in standard c++11 and compiled with g++4.8.5.
All experiments were performed on a machine with 20X Intel Xeon 2.3GHz and 385GB main memory running Linux(Red Hat Linux 7.3 64-bit). The number of landmarks is set to 100 by default.

TABLE III: Statistics of Datasets.

Name	Dataset	$\|V\|$	$\|E\|$	d_{avg}
FB	Facebook	63,731	817,035	25.6
GW	Gowalla	196,591	950,327	9.7
WI	WikiConflict	118,100	2,027,871	34.3
GO	Google	875,713	4,322,051	9.9
DB	DBLP	1,314,050	5,326,414	8.1
BE	Berkstan	685,230	6,649,470	19.4
YT	Youtube	3,223,589	9,375,374	5.8
PE	Pester	623,766	15,695,166	50.3
FL	Flickr	2,302,925	22,838,276	19.8
IN	Indochina	7,414,866	150,984,819	40.7

Exp 1: Indexing Time. This experiment evaluates the indexing time for three algorithms, HP-SPC$_s$, PSPC, and PSCP$^+$. The ordering time is also counted in the indexing time. What stands out in Figure 5 is that PSPC could beat HP-SPC$_s$ in the

[1]http://law.di.unimi.it
[2]http://konect.uni-koblenz.de
[3]https://snap.stanford.edu

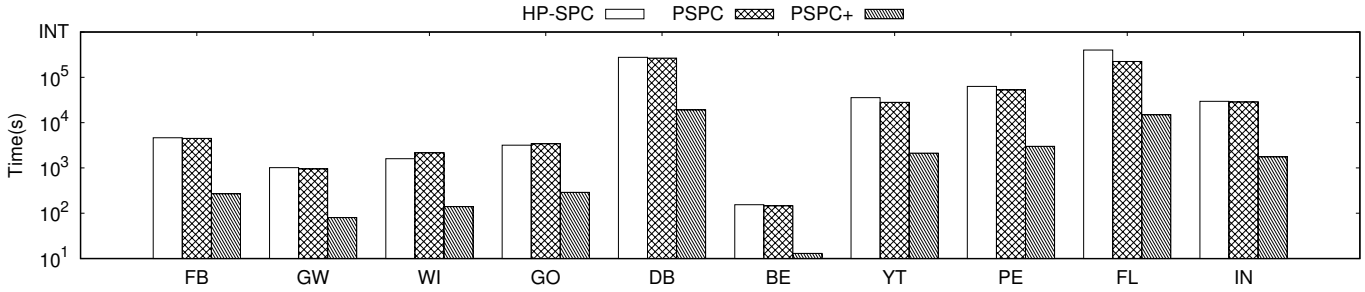


Fig. 5: Indexing Time (s) for HP-SPC$_s$, PSPC, and PSCP$^+$.

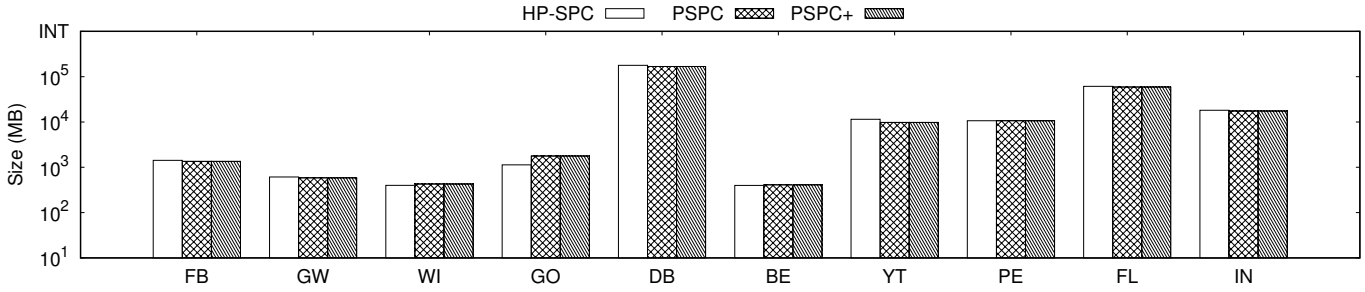


Fig. 6: Indexing Size (MB) for HP-SPC$_s$, PSPC, and PSCP$^+$.

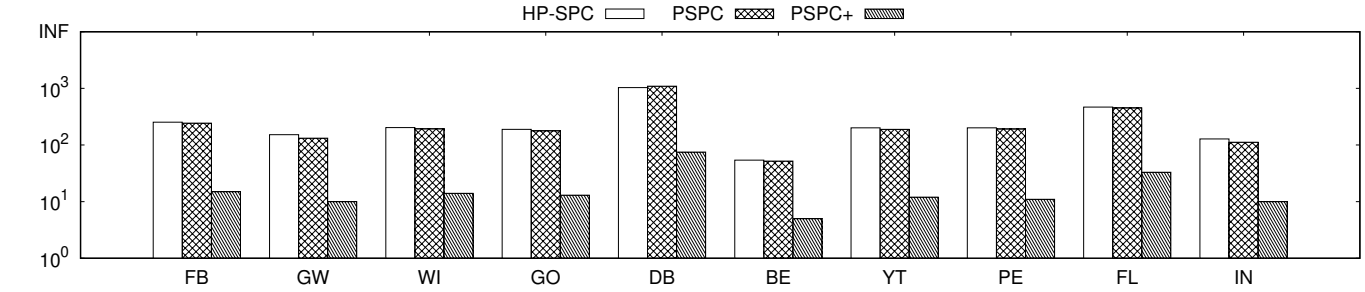


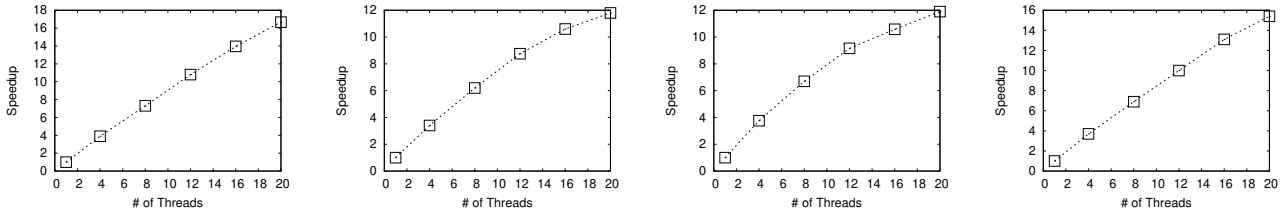
Fig. 7: Query Time (microsecond) for HP-SPC$_s$, PSPC, and PSCP$^+$.

7 of 10 datasets for single-core, except GW, GO, and BE. The PSPC could construct an index about 27% faster than HP-SPC$_s$ in YT, and about 18% faster on average. Moreover, PSPC could naturally be paralleled since the label entries in each round are divided into independent sets, while HP-SPC$_s$ need to obey label dependency due to the node order rank. As for the multiple cores, the speedup of our PSCP$^+$ could achieve nearly linear speedup with the growth of the number of cores. It is also apparent from this table that only PSCP$^+$ could construct the index for billion-scale, which is urgently demanded in real-world applications. In the tested 10 datasets, PSCP$^+$could achieve at least 12 speedups when using 20 threads compared with the single thread.

Exp 2: Index Size. Figure 6 shows the results on index size. What is striking about the table is that PSPC and PSCP$^+$ return the same index size. The reason behind this phenomenon is that the dependencies are eliminated between each thread. Thus, the final index would be the same with any number of threads. As for the HP-SPC$_s$, the index size is similar to PSPC and PSCP$^+$. The reason behind this result is that the parallel paradigm does not affect index size. What is interesting about the data in this figure is that PSPC and PSCP$^+$could achieve the same index size, which indicates that there is no dependency on the index construction in each iteration. Specifically, in each iteration, the execution order of each vertex in Pull-Based Paradigmor Push-Based Paradigmdoes not affect the final result of our index. Figure 6 illustrates that PSPC and PSCP$^+$ could construct a smaller index size in the datasets of FB, GW, YT, PE, and FL.

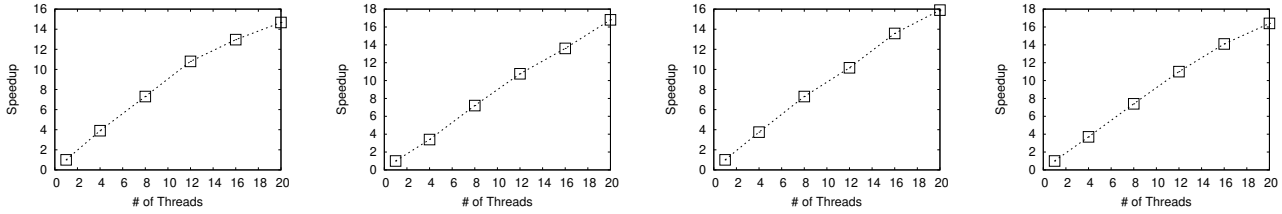
From the shortest path cover perspective, one of the most important factors for the index size is the node order of the original graph. Thus, it would compare different node orders in the following experiments.

Exp 3: Query Time. Exp 3 evaluates the average query time



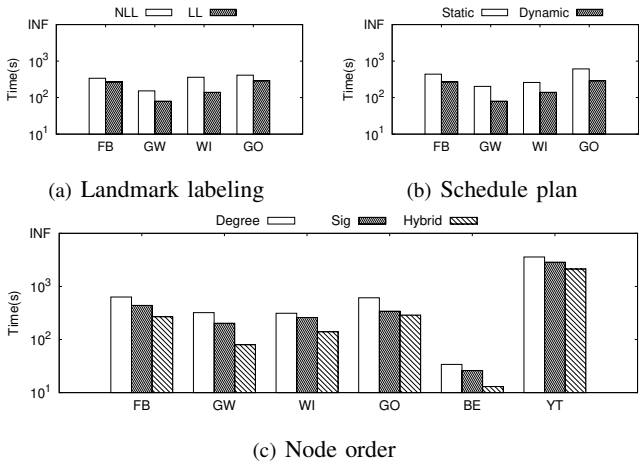
(a) FB (b) GO (c) GW (d) WI

Fig. 8: Speedup of indexing time when tuning the # of threads.



(a) FB (b) GO (c) GW (d) WI

Fig. 9: Speedup of query time when tuning the # of threads.



(a) Landmark labeling (b) Schedule plan

(c) Node order

Fig. 10: Ablation analysis of different techniques with 20 threads.

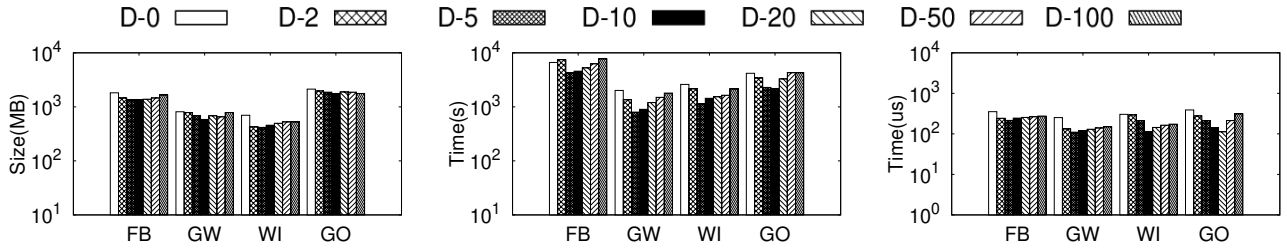
taken by HP-SPC$_s$, PSPC, and PSCP$^+$ with 100,000 random queries for each dataset. Figure 7 illustrates the query time of HP-SPC$_s$, PSPC, and PSCP$^+$. What is striking about the figures is that the query time of HP-SPC$_s$ and PSPC is similar. Both of them could answer queries in about 100 microseconds. Nevertheless, with the parallel techniques in PSCP$^+$, PSCP$^+$ could achieve a nearly linear speedup when compared with HP-SPC$_s$ and PSPC. The main reason is that the query is very efficient and there is no significant bottleneck in the query process. Thus, a divide and conquer strategy on the query workload could achieve a linear speedup.

Exp 4: Indexing Speedup on Multi-Cores. The speedup of the index time of an approach on x cores is calculated by using the index time of the approach with 1 core dividing that of x cores.

Thus, 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. This experiment evaluates the scalability of PSCP$^+$ by varying the number of threads on four datasets, i.e., FB, GO, GW, and WI. It is illustrated in Figure 8 that PSCP$^+$ could achieve nearly linear scalability with the growth of threads. When the number of threads is 20, PSCP$^+$ could achieve 16.7, 11.8, 11.9, and 15.4 speedups for FB, GO, GW, and WI, respectively. It is shown that the scalability of PSCP$^+$ is better in FB and WI than that of GO and GW. According to the statistics in III, FB and GW have a high average degree, i.e., 25.6 and 34.3 respectively, while GW and WI have a lower average degree, i.e., 9.7 and 9.9, respectively. What stands out in Figure 9 is that the speedup of query time could achieve nearly linear scalability with the growth of threads. A similar trend could be observed in terms of query time's scalability.

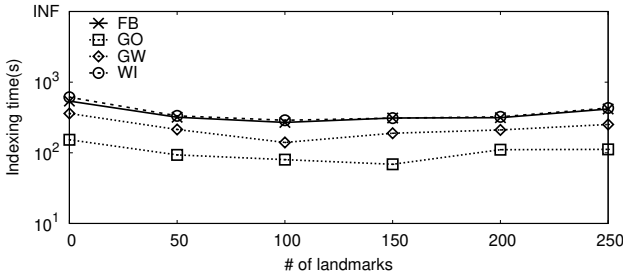
Exp 5: Ablation Analysis. Exp 5 analyzes the influence of separate techniques for the shortest path counting problem in terms of scalability. It evaluates the proposed techniques, i.e., landmark labeling, schedule plan, and node order, under 20 threads. Since these techniques do not have many effects on the indexing size and query time, it only compares their indexing time. Figure 10 includes three sub-figures, i.e., Figure 10(a), 10(b), and 10(c). In Figure 10(a), *LL* denotes landmark labeling, while *NLL* indicates the index construction without landmark labeling. What stands out in this figure is that landmark labeling could achieve about a little faster than that without landmark labeling. Figure 10(b) illustrates that our cost function-based schedule plan could achieve about somewhat faster indexing time than that of the static schedule plan. What is striking in Figure 10(c) indicates the hybrid node order could be the fastest among these three node orders.

Exp 6: The effect of δ. Figure 11 shows the effect of δ in terms of index time, index size, and query time. What stands out in this figure is that when the δ increases, the index time, index size, and query time decrease first and then increase. The reason would be that the tree decomposition order would



(a) Index Size(MB) (b) Index Time(s) (c) Query Time(us)

Fig. 11: The effect of threshold δ.



(a) Node order

Fig. 12: The effect of # of landmarks.

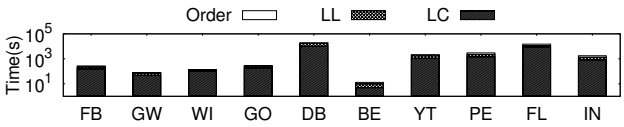


Fig. 13: The time cost of different part during index construction. Order indicates the time cost for node ordering, while LL and LC denote the Landmark labeling and Label Construction, respectively.

be suitable for the vertices with small order. Thus, the δ is set to 5 from the empirical study.

Exp 7: The effect of # of landmarks. Figure 12 illustrates the effect of # of landmarks. Since the landmarks do not affect the index size and query time, we only compare the index time. What is striking in this figure is that when the number of landmarks increases, the index time decreases first and then increases. The reason would be that there is an extra cost if landmark-based filtering returns a $false$ result. When the number of landmarks increases, the possibility of returning $false$ may increase.

Exp 8: Break Down the Indexing Time. Figure 13 illustrates the separate time cost for the node ordering, landmark labeling (LL), and label construction (LC). What stands out in the figure is that the LC dominates all the other two phases, which is the most time-consuming part. Although the LL and Order do not cost too much time, their results have an important impact on the LC phase.

VI. Related Works

In this section, some important related works are surveyed as follows.

Counting. Counting the occurrences of certain structs is also fundamental in graph analytics [35]–[38]. [39] present randomized algorithms with provable guarantee to count cliques and 5-vertex subgraphs in a graph, respectively. An algorithm that counts triangles in $O(m^{1.41})$ time is shown in [40]. There are also many works on counting paths and cycles in the literature.

The problem of exactly counting paths and cycles of length l, parameterized by l, is #W[1]-complete under parameterized Turing reductions [41]. In addition, given vertices s and t, the problem of counting the # of simple paths between s and t is #P-complete [42], [43]. [44] and [12] also study the problem of counting shortest paths for two vertices, but unlike this work, they focus on planar graphs and probabilistic networks, respectively.

Dynamic Maintenance for 2-hop Labeling. To adapt to the dynamic update of the network, some works [45]–[47] proposed dynamic algorithms to deal with edge insertion and deletion. For the edge insertion, a partial BFS for each affected hub is started from one of the inserted-edge endpoints and creates a label when finding the tentative distance is shorter than the query answer from the previous index. Also, hub labeling is used in the related constrained path-based problems, e.g., label constrained [48]–[50].

VII. Future Work and Conclusion

Future Work. Experiments demonstrate that for graphs such as DB and FL, our techniques require much more indexing time and index space than for other graphs of comparable size. Unfortunately, it is currently unclear what properties of these graphs contribute to this inefficiency. It would be interesting to investigate it in our future work.

Conclusion. We study the problem of counting the # of shortest paths between two vertices s and t in the context of parallel. To address the scalability issue of existing work, we propose a parallel algorithm to speedup this problem. We also investigate the proper vertex ordering for the parallel index construction. Our comprehensive experimental study verifies the effectiveness and efficiency of our algorithms.

Acknowledgment

This work is supported by Hong Kong RGC GRF grant (No. 14203618, No. 14202919, No. 14205520, and No. 14217322), RGC CRF grant (No. C4158-20G), Hong Kong ITC ITF grant (No. MRP/071/20X), and NSFC grant (No. U1936205).

REFERENCES

[1] A. D. Zhu, X. Xiao, S. Wang, and W. Lin, "Efficient single-source shortest path and distance queries on large graphs," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 998–1006, 2013.

[2] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 967–982, 2015.

[3] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," *Proceedings of the VLDB Endowment*, vol. 10, no. 2, pp. 61–72, 2016.

[4] Z. Yang, L. Lai, X. Lin, K. Hao, and W. Zhang, "Huge: An efficient and scalable subgraph enumeration system," in *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, (New York, NY, USA), p. 2049–2062, Association for Computing Machinery, 2021.

[5] Q. Shi, Y. Wang, P. Yao, and C. Zhang, "Indexing the extended dyck-cfl reachability for context-sensitive program analysis," *Proceedings of the ACM on Programming Languages*, vol. 6, no. OOPSLA2, pp. 1438–1468, 2022.

[6] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," in *International Symposium on Experimental Algorithms*, pp. 230–241, Springer, 2011.

[7] T. Akiba, C. Sommer, and K.-i. 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*, pp. 144–155, 2012.

[8] T. Akiba, Y. Iwata, and Y. Yoshida, "Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling," in *Proceedings of the 23rd international conference on World wide web*, pp. 237–248, 2014.

[9] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling distance labeling on small-world networks," in *Proceedings of the 2019 International Conference on Management of Data*, pp. 1060–1077, 2019.

[10] Y. Zhang and J. X. Yu, "Hub labeling for shortest path counting," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1813–1828, 2020.

[11] T. Oyama and H. Morohosi, "Applying the shortest-path-counting problem to evaluate the importance of city road segments and the connectedness of the network-structured system," *International Transactions in Operational Research*, vol. 11, no. 5, pp. 555–573, 2004.

[12] Y. Ren, A. Ay, and T. Kahveci, "Shortest path counting in probabilistic biological networks," *BMC bioinformatics*, vol. 19, no. 1, pp. 1–19, 2018.

[13] A. Botea, M. Mattetti, A. Kishimoto, R. Marinescu, and E. Daly, "Counting vertex-disjoint shortest paths in graphs," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, pp. 28–36, 2021.

[14] C.-Y. Shen, L.-H. Huang, D.-N. Yang, H.-H. Shuai, W.-C. Lee, and M.-S. 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*, pp. 415–424, 2017.

[15] J. Li, X. Wang, K. Deng, X. Yang, T. Sellis, and J. X. Yu, "Most influential community search over large social networks," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 871–882, IEEE, 2017.

[16] P. Rozenshtein, A. Anagnostopoulos, A. Gionis, and N. Tatti, "Event detection in activity networks," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1176–1185, 2014.

[17] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[18] R. Puzis, Y. Elovici, and S. Dolev, "Fast algorithm for successive computation of group betweenness centrality," *Physical Review E*, vol. 76, no. 5, p. 056709, 2007.

[19] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *European Symposium on Algorithms*, pp. 24–35, Springer, 2012.

[20] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong, "Exact top-k nearest keyword search in large networks," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 393–404, 2015.

[21] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto, "Efficient search ranking in social networks," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pp. 563–572, 2007.

[22] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data, June 2014.

[23] M. Riondato and E. M. Kornaropoulos, "Fast approximation of betweenness centrality through sampling," *Data Mining and Knowledge Discovery*, vol. 30, no. 2, pp. 438–475, 2016.

[24] M. Pontecorvi and V. Ramachandran, "A faster algorithm for fully dynamic betweenness centrality," *arXiv preprint arXiv:1506.05783*, 2015.

[25] M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu, "Hop doubling label indexing for point-to-point distance querying on scale-free networks," *arXiv preprint arXiv:1403.0779*, 2014.

[26] R. Halin, "S-functions for graphs," *Journal of geometry*, vol. 8, no. 1, pp. 171–186, 1976.

[27] N. Robertson and P. D. Seymour, "Graph minors. iii. planar tree-width," *Journal of Combinatorial Theory, Series B*, vol. 36, no. 1, pp. 49–64, 1984.

[28] H. L. Bodlaender, "A tourist guide through treewidth," *Acta cybernetica*, vol. 11, no. 1-2, p. 1, 1994.

[29] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. 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*, pp. 709–724, 2018.

[30] L. D. Valstar, G. H. Fletcher, and Y. Yoshida, "Landmark indexing for evaluation of label-constrained reachability queries," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 345–358, 2017.

[31] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, "Robust distance queries on massive networks," in *European Symposium on Algorithms*, pp. 321–333, Springer, 2014.

[32] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, pp. 157–168, 2012.

[33] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd international conference on World Wide Web*, pp. 1343–1350, 2013.

[34] J. Leskovec, A. Krevl, and S. Datasets, "Stanford large network dataset collection," 2011.

[35] S. Jain and C. Seshadhri, "A fast and provable method for estimating clique counts using turán's theorem," in *Proceedings of the 26th international conference on world wide web*, pp. 441–449, 2017.

[36] Q. Feng, Y. Peng, W. Zhang, Y. Zhang, and X. Lin, "Towards real-time counting shortest cycles on dynamic graphs: A hub labeling approach," in *ICDE*, IEEE, 2022.

[37] X. Qiu, W. Cen, Z. Qian, Y. Peng, Y. Zhang, X. Lin, and J. Zhou, "Real-time constrained cycle detection in large dynamic graphs," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1876–1888, 2018.

[38] Z. Yuan, Y. Peng, P. Cheng, L. Han, X. Lin, L. Chen, and W. Zhang, "Efficient k-clique listing with set intersection speedup," in *ICDE*, IEEE, 2022.

[39] A. Pinar, C. Seshadhri, and V. Vishal, "Escape: Efficiently counting all 5-vertex subgraphs," in *Proceedings of the 26th international conference on world wide web*, pp. 1431–1440, 2017.

[40] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997.

[41] J. Flum and M. Grohe, "The parameterized complexity of counting problems," *SIAM Journal on Computing*, vol. 33, no. 4, pp. 892–922, 2004.

[42] B. Roberts and D. P. Kroese, "Estimating the number of st paths in a graph.," *J. Graph Algorithms Appl.*, vol. 11, no. 1, pp. 195–214, 2007.

[43] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[44] I. Bezáková and A. Searns, "On counting oracles for path problems," in *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[45] T. Akiba, Y. Iwata, and Y. Yoshida, "Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling," in *Proceedings of the 23rd international conference on World wide web*, pp. 237–248, 2014.

[46] G. D'angelo, M. D'emidio, and D. Frigioni, "Fully dynamic 2-hop cover labeling," *Journal of Experimental Algorithmics (JEA)*, vol. 24, pp. 1–36, 2019.

[47] Y. Qin, Q. Z. Sheng, N. J. Falkner, L. Yao, and S. Parkinson, "Efficient computation of distance labeling for decremental updates in large dynamic graphs," *World Wide Web*, vol. 20, no. 5, pp. 915–937, 2017.

[48] Y. Peng, Y. Zhang, X. Lin, L. Qin, and W. Zhang, "Answering billion-scale label-constrained reachability queries within microsecond," *Proceedings of the VLDB Endowment*, vol. 13, no. 6, pp. 812–825, 2020.
[49] Y. Peng, X. Lin, Y. Zhang, W. Zhang, and L. Qin, "Answering reachability and k-reach queries on large graphs with label-constraints," *The VLDB Journal*, pp. 1–25, 2021.
[50] X. Chen, Y. Peng, S. Wang, and J. X. Yu, "Dlcr : Efficient indexing for label-constrained reachability queries on large dynamic graphs," *Proceedings of the VLDB Endowment*, 2022.