Hub Labeling for Shortest Path Counting

Yikai Zhang
Chinese University of Hong Kong
ykzhang@se.cuhk.edu.hk

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

ABSTRACT

The notion of shortest path is fundamental in graph analytics. While many works have devoted to devising efficient distance oracles to compute the shortest distance between any vertices s and t, we study the problem of efficiently counting the number of shortest paths between s and t in light of its applications in tasks such as betweenness-related analysis. Specifically, we propose a hub labeling scheme based on hub pushing and discuss several graph reduction techniques to reduce the index size. Furthermore, we prove several theoretical results on the performance of the scheme for some special graph classes. Our empirical study verifies the efficiency and effectiveness of the algorithms. In particular, a query evaluation takes only hundreds of microseconds in average for graphs with up to hundreds of millions of edges. We report our findings in this paper.

KEYWORDS

Shortest Path; Counting; Hub Labeling; Algorithms

ACM Reference Format:
Yikai Zhang and Jeffrey Xu Yu. 2020. Hub Labeling for Shortest Path Counting. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3389737

1 INTRODUCTION

The notion of shortest path is fundamental in graph analytics. Specifically, a path between two vertices s and t is shortest if its length is the minimum among all paths between s and t. Due to such an optimality, shortest paths have been employed in a large number of important problems, including keyword search [27, 31, 52], betweenness centrality [15, 44] and route planning [1, 2]. Given a graph G, the length of the shortest paths between s and t (*a.k.a.* the distance of s and t) is considered to indicate the relevance of the vertices. For example, (1) in nearest keyword search, among all the vertices with the specified keyword, the ones that are closer to the query source are preferred [31]; and (2) in social networks, distances are used in search ranking to help a user identify the most relevant results [54].

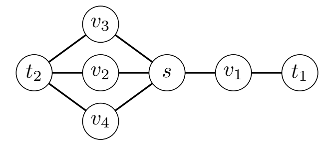


Figure 1: Graph H

It can be uninformative to base relevance solely on distance. Specifically, due to the small-world phenomenon, the diameter of many graphs in real world is typically small. As a result, many pairs of vertices are of the same distance. In such a case, based on the distance information alone, many pairs of vertices will be considered as equally relevant. This is by no means realistic. For example, consider the graph H in Figure 1. In H, both t_1 and t_2 are at distance 2 from s. Therefore, based on their distances alone, t_1 and t_2 are considered equally relevant to s. However, such a conclusion is counter-intuitive, since as can be observed, s and t_2 are connected by more shortest paths, and thus are more relevant. In light of this, we study in this paper the problem of counting the # of shortest paths between any two given vertices.

As another application, consider the problem of group betweenness evaluation [44], which is to measure the importance of a vertex set C to G. Let $P_{s,t}$ be the set of shortest paths between vertices s and t, $\mathtt{spc}(s,t)$ be the # of shortest paths between s and t, and $\mathtt{spc}_C(s,t)$ be the # of the paths in $P_{s,t}$ that pass through C. The group betweenness of C, denoted $\ddot{B}(C)$, is defined to be $\ddot{B}(C) = \sum_{s,t} \mathtt{spc}_C(s,t)/\mathtt{spc}(s,t)$. As shown in [44], there is an algorithm GBC to evaluate $\ddot{B}(C)$ incrementally. Specifically, suppose $C = \{v_1, \ldots, v_{|C|}\}$ and let $C_i = \{v_1, \ldots, v_i\}$. In the i-th iteration, GBC computes $\ddot{B}(C_i)$ by adding to $\ddot{B}(C_{i-1})$ the total fraction of shortest paths that pass through v_i but not C_{i-1}. As a result, after $|C|$ iterations, $\ddot{B}(C)$ is obtained. To make it efficient, GBC takes as input three $|C| \times |C|$ matrices D, Σ and $\tilde{B}$, which store for $\forall x, y \in C$ the distance between x and y, $\mathtt{spc}(x,y)$, and the path betweenness of (x,y), respectively. After $\ddot{B}(C_i)$ is computed, GBC updates $\tilde{B}$ based on D and Σ such that $\tilde{B}_{v_{i+1},v_{i+1}} =$

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

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389737

$\ddot{B}(C_{i+1}) - \ddot{B}(C_i)$. In this way, $\ddot{B}(C_{i+1})$ would be trivial to compute in the next iteration. Overall, GBC uses $O(T + |C|^3)$ time, where T is the time to construct D, Σ and $\tilde{B}$. In tasks such as estimating group betweenness distribution, the # of groups to evaluate is enormous. For this reason, to reduce the online time to construct D, Σ and $\tilde{B}$, [44] proposes to precompute and store the distance, the # of shortest paths, and the path betweenness for every pair of vertices, incurring unaffordable overhead. While the cost related to distance and path betweenness can be reduced by using distance hub labeling and VC-dimension-based techniques [48], respectively, the cost regarding shortest path counting remains intractable.

In light of these, we propose to devise a hub labeling for efficient shortest path counting. Formally, a hub labeling for shortest path counting is a pair $(L(\cdot), f(\cdot,\cdot))$ such that for any two vertices s and t, $\mathsf{spc}(s,t) = f(L(s), L(t))$ holds. That is, $\mathsf{spc}(s,t)$ can be obtained by inspecting only $L(s)$ and $L(t)$, without searching G. Here, $L(v)$ is called the label of v and contains a small number of vertices known as hubs, together with some auxiliary information. We stress that the cover constraint, which is the basis of distance hub labeling [17], is no longer sufficient under the case of shortest path counting. Indeed, for any vertices s and t, the cover constraint only requires that one, rather than all, of the shortest paths between s and t be covered by the hubs in $L(s) \cap L(t)$, thereby possibly resulting in underestimated results. We also stress that it is important not to cover a shortest path more than once such that we do not obtain overestimated results.

Contributions. Our main contributions are as follows.

Hub Labeling for Shortest Path Counting. Unlike prior work [12], we focus on non-planar graphs. We first propose the notion of *exact shortest path cover* (ESPC) which is guaranteed to cover each shortest path exactly once. An ESPC naturally induces a hub labeling. We then propose an approach to constructing an ESPC, and accordingly devise an algorithm based on hub pushing to construct a hub labeling.

Index Reduction. We further discuss 3 index reduction techniques, namely reduction by 1-shell, reduction by neighborhood equivalence, and reduction by independent set, to reduce the resulting index size. Specifically, (1) the 1-shell reduction exploits the core-fringe structure of a graph; (2) the neighborhood-equivalence reduction exploits the symmetry between vertices; and (3) the independent-set reduction exploits the fact the a query between two vertices can be converted to a query between the neighbors of the two vertices.

Theoretical Results. We show that our proposed algorithm is able to exploit the intrinsic dimension of a graph. In particular, our results show that if certain condition is satisfied, the resulting hub labeling can be used to answer a query in $O(\sqrt{n})$ time for a planar graph, in $O(\omega \log n)$ time for a graph with treewidth ω, and in $O(h \log D)$ time for a graph with highway dimension h and diameter D.

Experimental Evaluation. We conducted experiments to verify the efficiency and effectiveness of our algorithms on 10 graphs. Our experiments show that (1) the algorithms can finish indexing within 3 hours for all graphs except `DBLP` and `Flickr`; for the largest graph `Indochina` with 150 millions of edges, indexing takes only 0.8 hours; (2) the 3 index reduction techniques together can reduce the index size by from 38% to 79% across the graphs tested; (3) a query evaluation takes only hundreds of microseconds in average; and (4) compared with [12], our algorithm is able to provide comparable indexing time, much smaller index size and much better query time on an artificial planar graph `Delaunay`.

Related Work. We classify the related work as follows.

Graph Search for Distance Queries. Both breadth-first search and Dijkstra's algorithm are classic algorithms for shortest path problems. Instead of Dijkstra's algorithm, the ALT algorithm [24] employs A* search with a *landmark*-based heuristic to speed up query processing. The notion of *vertex reach* is proposed to reduce the search space of Dijkstra's algorithm in [26]. In the approaches that are based on *arc-flag* [28], a graph is partitioned into k regions and each arc (u, v) is associated with a k-bit flag of which the i-th bit indicates if there is a shortest path from u to the i-th region via (u, v). Based on the arc-flags, the search space of Dijkstra's algorithm can also be greatly reduced. The notion of *highway hierarchy* (HH) [50] is designed to capture the natural hierarchy of road networks so that queries can be answered by searching the sparse high levels of HH, reducing the search space. Geisberger et. al. [23] introduced *contraction hierarchy* (CH), in which, different from HH, each level consists of only one vertex. Its efficiency relies heavily on the notion of *shortcut*, which is to preserve the distance between vertices after less important vertices are removed. In transit node routing [10], a set T of *transit nodes* is selected and each vertex v is associated with a small subset $A(v)$ of T via which v can reach any distant destination. If a query (s, t) is distant, the distance can be answered by inspecting the distances from s (resp. t) to $A(s)$ (resp. $A(t)$) and the distances between vertices in $A(s)$ and $A(t)$; otherwise, the query is answered by CH. The techniques mentioned above can be combined, leading to more efficient algorithms. For example, the REAL algorithm [25] is obtained by combining *reach* and ALT, and the SHARC algorithm [11] is based on *shortcut* and *arc-flag*.

Hub Labeling for Distance Queries. Another important class of algorithms for distance evaluation is hub labeling [17]. In this class, a label $L(v)$ is computed for each vertex v such that the distance between two vertices s and t can be obtained by inspecting $L(s)$ and $L(t)$ only, without searching the graph. In general, it is NP-hard to construct a labeling with the minimum size [17]. In [1, 2], efficient hub labelings for road networks are discussed. A labeling scheme that instead uses paths as hubs is presented in [5]. In [42], under the assumption of small treewidth and bounded tree height, a scheme combining both hub labeling and hierarchy is proposed for road networks. For real graphs that are scale-free, pruned landmark labeling (PLL) [6] is the state-of-the-art and its many extensions have been devised. For example, an external algorithm generating the same set of labels is proposed in [32]; a parallel algorithm is devised in [38]; and [7] shows an algorithm to update the labels when new edges are inserted into the graph. In [39], an experimental study on hub labeling for distance queries is presented.

Counting. Counting the occurrences of certain structs is also fundamental in graph analytics. [30] and [43] 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 [9]. 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 [21]. In addition, given vertices s and t, the problem of counting the # of simple paths between s and t is #P-complete [49, 53]. [12] and [47] also study the problem of counting shortest paths for two vertices, but unlike this work, they focus on planar graphs and probabilistic networks, respectively.

Organization. The rest of this paper is organized as follows. Section 2 introduces the preliminaries. Section 3 presents the main algorithm. The 3 index reduction techniques are discussed in Section 4. Section 5 presents our theoretical findings. Section 6 shows the experimental results. We discuss the extension and the limitations of our work in Sections 7 and 8, respectively. Section 9 concludes the paper.

2 PRELIMINARIES

Graphs. We focus on an unweighted and undirected graph $G = (V, E)$, where V and E denote 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 $\mathtt{nbr}(v)$ be the set of v's neighbors and $\mathtt{deg}(v)$ be the degree of v. A path p from vertex s to vertex t is defined as a sequence of vertices ($s = v_0, v_1, \ldots, v_\ell = t$) such that $(v_i, v_{i+1}) \in E$ for $0 \le i < \ell$. The length of p, denoted by $\mathtt{len}(p)$, is the number of edges included in p. In other words, $\mathtt{len}(p) = \ell$. For convenience, we use the notation $\mathtt{rev}(p)$ to denote the reverse of a path. Specifically, $\mathtt{rev}(p) = (v_\ell, v_{\ell-1}, \ldots, v_0)$. A path from s to t is shortest if its length is no larger than any other path from s to t. We

Notation	Description
$G = (V, E)$	an unweighted and undirected graph
n, m	n is the # of vertices, and m is the # of edges
$\mathtt{nbr}(v)$	the set of neighbors of v
$\mathtt{deg}(v)$	the degree of v
$\mathtt{len}(p)$	the length of a path p $\mathtt{len}(p) = \ell$ if $p = (v_0, v_1, \ldots, v_\ell)$
$\mathtt{rev}(p)$	the reverse of a path p $\mathtt{rev}(p) = (v_\ell, v_{\ell-1}, \ldots, v_0)$ if $p = (v_0, v_1, \ldots, v_\ell)$
$p_{s,t}$	a shortest path from s to t
$P_{s,t}$	the set of shortest paths from s to t
$Q_{s,t}$	the set of the vertices involved in $P_{s,t}$
$\mathtt{sd}_G(s,t)$	the shortest distance between s and t in G; G is omitted when the context is clear
$\mathtt{spc}_G(s,t)$	the # of shortest paths between s and t in G; G is omitted when the context is clear
$\le$	a total order over V
$w \le v$	w has a higher rank than v with respect to $\le$

Table 1: Notations

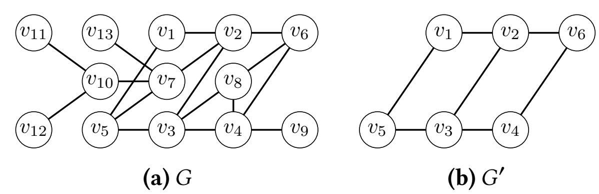


Figure 2: Two Graphs

denote a shortest path in G from s to t by $p_{s,t}$. The shortest distance between s and t in G, denoted by $\mathtt{sd}_G(s,t)$, is defined as the length of the shortest paths 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 $Q_{s,t}$. We use $\mathtt{spc}_G(s,t)$ to denote the number of shortest paths from s to t in G. When the context is clear, we use $\mathtt{sd}(s,t)$ and $\mathtt{spc}(s,t)$ instead of $\mathtt{sd}_G(s,t)$ and $\mathtt{spc}_G(s,t)$ for simplicity. Let $\le$ be a total order over V. For two distinct vertices w and v, if $w \le v$, then we say w has a higher rank than v. We summarize the notations in Table 1.

Example 2.1. Figure 2a shows a graph G, where $\mathtt{nbr}(v_7) = \{v_2, v_5, v_{10}, v_{13}\}$ and $\mathtt{deg}(v_7) = 4$. There are several paths from v_3 to v_6, such as $p_1 = (v_3, v_4, v_6)$, $p_2 = (v_3, v_8, v_6)$, $p_3 = (v_3, v_2, v_6)$ and $p_4 = (v_3, v_4, v_8, v_6)$. Among them, only p_1, p_2 and p_3 are shortest. Hence, $\mathtt{sd}(v_3, v_6) = 2$, $P_{v_3,v_6} = \{p_1, p_2, p_3\}$, $Q_{v_3,v_6} = \{v_2, v_3, v_4, v_6, v_8\}$ and $\mathtt{spc}(v_3, v_6) = 3$. The reverse of p_4 is $\mathtt{rev}(p_4) = (v_6, v_8, v_4, v_3)$.

Hub Labeling for Shortest Distance Queries. Given two query vertices s and t, a shortest distance query asks for the shortest distance $\mathtt{sd}(s,t)$ between s and t. To efficiently deal with such queries, *hub labeling* has been widely employed.

Formally, given an undirected graph G, a hub labeling is to assign to each vertex $v \in V$ a label $L(v)$, which comprises entries of the form $(w, \mathsf{sd}(v, w))$. We say w is a *hub* of v if $(w, \mathsf{sd}(v, w)) \in L(v)$. When the context is clear, we also use $L(v)$ to denote the set of v's hubs. A hub labeling needs to satisfy the cover constraint. That is, for any two vertices s and t, there exists a vertex $w \in L(s) \cap L(t)$ that lies on a shortest path between s and t. The size of a hub labeling is defined to be $\sum_v |L(v)|$, *i.e.*, the total number of label entries.

Given a labeling $L(\cdot)$ of G, for any vertices s and t, their shortest distance can be evaluated in linear time as follows:

$$\mathsf{sd}(s,t) = \min_{w \in L(s) \cap L(t)} \mathsf{sd}(s,w) + \mathsf{sd}(t,w) \tag{1}$$

Given a total order $\leq$ over the vertices, a *canonical* hub labeling is one that contains only the following hubs. For two vertices v and w, $w \in L(v)$ if and only if w is the highest ranked vertex in $Q_{v,w}$. Take v_2 and v_4 in Figure 2a as an example. Assume that $v_i \leq v_j$ if and only if $i \leq j$. It is easy to conclude $v_2 \in L(v_4)$ since v_2 has a higher rank than the other vertices in $Q_{v_4,v_2} = \{v_2, v_3, v_4, v_6\}$. To see that a canonical labeling indeed obeys the cover property for any two vertices s and t, it suffices to observe that the highest ranked vertex in $Q_{s,t}$ belongs to both $L(s)$ and $L(t)$.

3 HUB LABELING FOR SHORTEST PATH COUNTING

In the following we first introduce our hub labeling scheme. We then describe how to compute the labeling.

3.1 Exact Shortest Path Covering

The cover constraint is not enough for correct shortest path counting. Indeed, it only ensures that one, rather than all, of the shortest paths between s and t is covered, and thus a labeling that obeys only the cover property may lead to underestimated results. Also, it is necessary to avoid double covering a shortest path. Otherwise, overestimated results can be resulted. Hence, we need to design our covering scheme carefully so that each shortest path is covered exactly once.

To this end, we focus on the following covering scheme. Specifically, the underlying idea is to find for each vertex v a collection $T(v)$ of entries of the form $(w, C_{v,w})$, where $C_{v,w} \subseteq P_{v,w}$ is a subset of the shortest paths from v to w. For any two vertices u and v, we define the shortest paths covered by $T(u)$ and $T(v)$ as a *multiset* as follows.

$$\begin{aligned}
&\mathsf{cover}(T(u), T(v)) \\
&= \{p_1 \odot \mathsf{rev}(p_2) \mid (w, C_{u,w}) \in T(u), (w, C_{v,w}) \in T(v), \\
&\qquad p_1 \in C_{u,w}, p_2 \in C_{v,w}, \\
&\qquad \mathsf{sd}(u,w) + \mathsf{sd}(v,w) = \mathsf{sd}(u,v)\}
\end{aligned}$$

Here, $\odot$ denotes the concatenation of two paths. Intuitively, for each hub $w \in T(u) \cap T(v)$ that is on some shortest path between u and v, the paths in $C_{u,w}$ are concatenated with those in $C_{v,w}$ to obtain shortest paths between u and v. We would like to stress two points in the following. First, $\mathsf{cover}(T(u), T(v))$ contains only the shortest paths between u and v. By definition, each such shortest path consists of two parts, where the front part is from some entry of $T(u)$ and the back part is from some entry of $T(v)$. Second, a shortest path may be present multiple times in the multiset $\mathsf{cover}(T(u), T(v))$, which is undesirable.

Example 3.1. Consider graph G' in Figure 2b. Suppose $T(v_5)$ contains 2 entries, namely, (v_1, P_{v_5,v_1}) and (v_2, P_{v_5,v_2}). Also, suppose $T(v_6)$ contains 2 entries, which are (v_1, P_{v_6,v_1}) and (v_2, P_{v_6,v_2}). Since $\mathsf{sd}(v_5, v_6) = \mathsf{sd}(v_5, v_1) + \mathsf{sd}(v_1, v_6) = \mathsf{sd}(v_5, v_2) + \mathsf{sd}(v_2, v_6)$, $\mathsf{cover}(T(v_5), T(v_6))$ contains the following 3 shortest paths: $p_1 = (v_5, v_1) \odot (v_1, v_2, v_6)$, $p_2 = (v_5, v_1, v_2) \odot (v_2, v_6)$ and $p_3 = (v_5, v_3, v_2) \odot (v_2, v_6)$. Note that (v_5, v_1, v_2, v_6) is covered twice by p_1 and p_2. In this example, the $C_{v,w}$ of each entry happens to be exactly $P_{v,w}$. We emphasize that this is not always the case in general.

$T(\cdot)$ is desired to be an *exact shortest path covering* (ESPC for short). That is, for any two vertices u and v, the *multiset* $\mathsf{cover}(T(u), T(v))$ is required to be exactly the same as $P_{u,v}$, *i.e.*, the set of shortest paths between u and v.

Example 3.2. The second column of Table 2 shows an exact shortest path covering $T(\cdot)$ for graph G' (Figure 2b). To verify that $T(\cdot)$ is indeed an ESPC, we take $T(v_5)$ and $T(v_6)$ as an example. Hubs v_5 and v_6 are only present in one of $T(v_5)$ and $T(v_6)$, thus are omitted. Hubs v_2 and v_3 both lie on some shortest paths between v_5 and v_6. For hub v_2, concatenating the paths in C_{v_5,v_2} and C_{v_6,v_2} results in 2 paths, namely, $p_1 = (v_5, v_1, v_2) \odot (v_2, v_6)$ and $p_2 = (v_5, v_3, v_2) \odot (v_2, v_6)$. Similarly, for hub v_3, we can obtain $p_3 = (v_5, v_3) \odot \mathsf{rev}((v_6, v_4, v_3)) = (v_5, v_3) \odot (v_3, v_4, v_6)$. These 3 shortest paths make up P_{v_5,v_6}.

An ESPC naturally induces a hub labeling for shortest path counting. In detail, for each vertex v, we construct its label $L(v)$ as follows. Initially, $L(v)$ is empty. Then, for each entry $(w, C_{v,w})$ in $T(v)$, we add to $L(v)$ accordingly a triplet $(w, \mathsf{sd}(v, w), \sigma_{v,w})$, where $\sigma_{v,w} = |C_{v,w}|$ is the # of shortest paths included in $C_{v,w}$. Given $L(\cdot)$ constructed this way, it is not hard to verify the following equation.

$$|\mathsf{cover}(T(u), T(v))| = \sum_{\substack{w \in L(u),\, w \in L(v), \\ \mathsf{sd}(u,w) + \mathsf{sd}(v,w) = \mathsf{sd}(u,v)}} \sigma_{u,w} \cdot \sigma_{v,w} \tag{2}$$

Provided that $T(\cdot)$ is an ESPC, then the result of Equation (2) is exactly $|P_{u,v}| = \mathsf{spc}(u, v)$. Moreover, the resulting $L(\cdot)$ obeys the cover property. Hence, we can capitalize on Equation (1) to compute the shortest distance $\mathsf{sd}(u, v)$, which is necessary to determine the eligible w's in Equation (2).

	$T(\cdot)$	$L(\cdot)$
v_1	$(v_2, P_{v_1,v_2}), (v_3, (v_1, v_5, v_3)), (v_5, P_{v_1,v_5}), (v_1, P_{v_1,v_1})$	$(v_2, 1, 1), (v_3, 2, 1), (v_5, 1, 1), (v_1, 0, 1)$
v_2	(v_2, P_{v_2,v_2})	$(v_2, 0, 1)$
v_3	$(v_2, P_{v_3,v_2}), (v_3, P_{v_3,v_3})$	$(v_2, 1, 1), (v_3, 0, 1)$
v_4	$(v_2, P_{v_4,v_2}), (v_3, P_{v_4,v_3}), (v_6, P_{v_4,v_6}), (v_4, P_{v_4,v_4})$	$(v_2, 2, 2), (v_3, 1, 1), (v_6, 1, 1), (v_4, 0, 1)$
v_5	$(v_2, P_{v_5,v_2}), (v_3, P_{v_5,v_3}), (v_5, P_{v_5,v_5})$	$(v_2, 2, 2), (v_3, 1, 1), (v_5, 0, 1)$
v_6	$(v_2, P_{v_6,v_2}), (v_3, (v_6, v_4, v_3)), (v_6, P_{v_6,v_6})$	$(v_2, 1, 1), (v_3, 2, 1), (v_6, 0, 1)$

Table 2: An Exact Shortest Path Covering of G' and the Corresponding Hub Labeling

Example 3.3. The third column of Table 2 shows the corresponding $L(\cdot)$ for the ESPC $T(\cdot)$ in the second column. Take v_6 as an example. Since $\mathsf{sd}(v_6, v_2) = 1$ and $|P_{v_6,v_2}| = 1$, for the entry (v_2, P_{v_6,v_2}) in $T(v_6)$, an entry $(v_2, 1, 1)$ is added to $L(v_6)$. Similarly, $(v_3, (v_6, v_4, v_3))$ and (v_6, P_{v_6,v_6}) result in $(v_3, 2, 1)$ and $(v_6, 0, 1)$, respectively. We have $\mathsf{sd}(v_5, v_6) = \min\{\mathsf{sd}(v_5, v_2) + \mathsf{sd}(v_6, v_2), \mathsf{sd}(v_5, v_3) + \mathsf{sd}(v_6, v_3)\} = 3$ and $\mathsf{spc}(v_5, v_6) = \sigma_{v_5,v_2} \cdot \sigma_{v_6,v_2} + \sigma_{v_5,v_3} \cdot \sigma_{v_6,v_3} = 2 \cdot 1 + 1 \cdot 1 = 3$.

Constructing an ESPC. We next describe how to construct an ESPC. Let $\leq$ be a total order over V. A trough path [32] is a path in which one of its endpoints has a higher rank than all the remaining vertices. A trough shortest path is a trough path that is also a shortest path. For example, consider graph G' in Figure 2b and a total order $\leq$ where $v_2 \leq v_3 \leq v_5 \leq v_6 \leq v_1 \leq v_4$. The path (v_1, v_2, v_6) is not a trough path since v_2 has a higher rank than both endpoints v_1 and v_6. The path (v_6, v_4, v_3) is a trough shortest path because one endpoint (*i.e.* v_3) has the highest rank and the path is shortest.

Given a total order $\leq$ over the vertices, we can construct an ESPC as follows. Initially, $T(v)$ is empty for each vertex v. Then, for any two (possibly identical) vertices v and w with $w \leq v$, we add an entry $(w, C_{v,w})$ to $T(v)$, where $C_{v,w}$ is the set of *all* trough shortest paths from v to w, provided that $C_{v,w}$ is not empty. Note that for each such entry, since $w \leq v$, w has the highest rank in p for each path $p \in C_{v,w}$. For convenience, we denote the $T(\cdot)$ constructed this way and the corresponding $L(\cdot)$ by $T_{\leq}(\cdot)$ and $L_{\leq}(\cdot)$, respectively.

Example 3.4. Consider the total order $v_2 \leq v_3 \leq v_5 \leq v_6 \leq v_1 \leq v_4$ for G' (Figure 2b). The $T(\cdot)$ shown in Table 2 is constructed with respect to $\leq$. Take $T(v_6)$ for instance. Since v_2 has a higher rank than all other vertices, all paths in P_{v_6,v_2} are trough shortest paths as a matter of course. There are 2 shortest paths from v_6 to v_3, *i.e.*, (v_6, v_2, v_3) and (v_6, v_4, v_3), but the former is not a trough shortest path due to the presence of v_2. In light of this, $(v_3, (v_6, v_4, v_3))$ is added to $T(v_6)$.

THEOREM 3.5. *$T_{\leq}(\cdot)$ is an exact shortest path covering.*

PROOF. Consider two vertices u and v. We prove $P_{u,v} \subseteq \mathsf{cover}(T_{\leq}(u), T_{\leq}(v))$ and $\mathsf{cover}(T_{\leq}(u), T_{\leq}(v)) \subseteq P_{u,v}$ below.

Let $p_{u,v}$ be any shortest path from u to v and let w be the vertex with the highest rank in $p_{u,v}$. Observe that $p_{u,v}$ can be rewritten as $p_{u,w} \odot p_{w,v}$ where $p_{u,w}$ is the subpath from u to w and $p_{w,v}$ is the subpath from w to v in p. Since w has the highest rank, both $p_{u,w}$ and $\mathsf{rev}(p_{w,v})$ are trough shortest paths. Hence, there exist entries $(w, C_{u,w})$ and $(w, C_{v,w})$ respectively in $T_{\leq}(u)$ and $T_{\leq}(v)$, where $p_{u,w} \in C_{u,w}$ and $\mathsf{rev}(p_{w,v}) \in C_{v,w}$. As a result, $p_{u,v} \in \mathsf{cover}(T_{\leq}(u), T_{\leq}(v))$, implying $P_{u,v} \subseteq \mathsf{cover}(T_{\leq}(u), T_{\leq}(v))$.

In order to prove $\mathsf{cover}(T_{\leq}(u), T_{\leq}(v)) \subseteq P_{u,v}$, it suffices to prove $\mathsf{cover}(T_{\leq}(u), T_{\leq}(v))$ contains no duplicates, since by definition $\mathsf{cover}(T_{\leq}(u), T_{\leq}(v))$ contains only shortest paths. Assume the opposite; that is, there is a shortest path $p_{u,v}$ appearing more than once in $\mathsf{cover}(T_{\leq}(u), T_{\leq}(v))$. Then, there must exist two distinct hubs w_1 and w_2 with $w_1 \leq w_2$ in $T_{\leq}(u) \cap T_{\leq}(v)$, each of which results in a copy of $p_{u,v}$. Therefore, w_1 lies on some trough path in $C_{u,w_2} \cup C_{v,w_2}$, implying that w_2 has a higher rank than w_1. A contradiction. □

We emphasize that $T_{\leq}(\cdot)$ is minimal in that after removing any entry from any $T_{\leq}(v)$, $T_{\leq}(\cdot)$ is no longer an ESPC.

3.2 A Hub Pushing Algorithm

Given a total order $\leq$, we aim to construct $L_{\leq}(\cdot)$ without materializing $T_{\leq}(\cdot)$. To this end, we propose a hub pushing algorithm, in which for each vertex w, vertices that have w as a hub are identified and label entries are generated accordingly on the fly. For each vertex v, we classify the entries in $T_{\leq}(v)$ into two types. In particular, an entry $(w, C_{v,w}) \in T_{\leq}(v)$ is called *canonical* if $C_{v,w}$ is exactly $P_{v,w}$, and w is said to be a canonical hub of v in such a case. Otherwise, the entry is *non-canonical* and w is a non-canonical hub of v. Intuitively, a hub w of v is canonical if and only if all shortest paths between w and v are trough paths. We use $T^{c}_{\leq}(v)$ and $T^{nc}_{\leq}(v)$ to denote the set of canonical entries and the set of non-canonical entries in $T_{\leq}(v)$, respectively. $L^{c}_{\leq}(v)$ and $L^{nc}_{\leq}(v)$ are defined correspondingly. Recall that in a canonical hub labeling $L_{\mathsf{sd}}(\cdot)$ for shortest distance queries, w is a hub of v if and only if w has the highest rank in $Q_{v,w}$. Therefore, under the same $\leq$, $L^{c}_{\leq}(\cdot)$ contains the same hubs as $L_{\mathsf{sd}}(\cdot)$. That is to say, $L^{c}_{\leq}(\cdot)$ alone is sufficient for shortest distance queries, but for correct shortest path counting, we still need to supplement $L^c_{\leq}(\cdot)$ with $L^{nc}_{\leq}(\cdot)$.

Our hub pushing algorithm named HP-SPC is shown in Algorithm 1. For each vertex w in descending order of rank, it conducts a breadth-first search to seek those vertices v that have w as a hub in $L_{\leq}(v)$. For ease of explanation, we denote by H_w the vertices with higher ranks than w and by G_w the resulting graph after removing H_w from G. Note that all trough paths in G that start from w and have w as the highest ranked vertex are preserved in G_w. For a vertex v, Algorithm 1 uses $D[v]$ and $C[v]$ to keep track of the shortest distance and the number of shortest paths between v and w when searching G_w, respectively. For correctness, the algorithm maintains the following loop invariant throughout:

Loop Invariant: At the start of each iteration of the `for` loop (lines 3-20), all label entries $(w', \cdot, \cdot)$ with $w' \in H_w$ have been correctly added to $L^c_{\leq}(\cdot)$ and $L^{nc}_{\leq}(\cdot)$.

With this invariant in hand, we are ready to explain lines 8-13. Among all paths between v and w that pass through H_w, let $P^{H_w}_{v,w}$ be the ones of the smallest length and let $Q^{H_w}_{v,w}$ be the vertices involved in $P^{H_w}_{v,w}$. Let w^* be the highest ranked vertex in $Q^{H_w}_{v,w}$. By construction, w^* belongs to H_w, and moreover, the shortest paths in $P_{w,w^*} \cup P_{v,w^*}$ are all trough paths. Therefore, by the loop invariant above, w^* has been added to both $L^c_{\leq}(v)$ and $L^c_{\leq}(w)$ when the algorithm enters the iteration for w (lines 4-20). It thus can be concluded that the result d in line 8 is essentially the smallest length of the paths between v and w that pass through H_w. For each vertex v in G_w, if the set $C_{v,w}$ of trough shortest paths between v and w is empty (*i.e.*, w is not a hub of v), then either v is not visited in the BFS from w or $D[v] > d$. In either case, Algorithm 1 does not add w to $L_{\leq}(v)$. In contrast, if $C_{v,w}$ is non-empty, by induction on $\mathsf{sd}_{G_w}(v, w)$, we can prove that $D[v] = \mathsf{sd}_{G_w}(v, w)$ and $C[v] = |C_{v,w}|$. If $d = D[v]$ (line 10), the shortest paths between v and w in G_w are trough shortest paths in G. But since w^* belongs to some shortest path between v and w, w is a non-canonical hub of v. If $d > D[v]$ (line 12), all shortest paths between v and w do not pass through H_w. Hence, w is a canonical hub in such a case. Note that since we correctly add w to the related labels, the loop invariant is maintained. When the algorithm terminates, the loop invariant ensures the correctness of the algorithm.

Example 3.6. Consider the graph G in Figure 2a and the order $v_2 \leq v_3 \leq v_7 \leq v_8 \leq \cdots$. At first, HP-SPC pushes v_2 (Figure 3a). Since v_2 has the highest rank, HP-SPC visits all vertices. Consequently, all vertices have v_2 as a hub. The case of v_3 is similar since there exist trough shortest paths between v_3 and any other vertex except v_2. When pushing v_7, only the vertices in the left part are visited (Figure 3b). For the hub v_8, the right part of G is visited (Figure 3c).

Algorithm 1: HP-SPC $(G, \leq)$

1 **for each** $v \in V$ **do**
2 $\quad L^c_{\leq}(v) \leftarrow \emptyset$; $L^{nc}_{\leq}(v) \leftarrow \emptyset$; $D[v] \leftarrow \infty$; $C[v] \leftarrow 0$;
3 **for each** $w \in V$ *in descending order of rank* **do**
4 $\quad Q \leftarrow$ an empty queue; Q.enqueue(w);
5 $\quad D[w] \leftarrow 0$; $C[w] \leftarrow 1$; $R \leftarrow \{w\}$;
6 $\quad$ **while** *Q is not empty* **do**
7 $\quad\quad v \leftarrow Q$.dequeue();
8 $\quad\quad d \leftarrow \min_{w' \in L^c_{\leq}(w) \cap L^c_{\leq}(v)} \mathsf{sd}(w, w') + \mathsf{sd}(v, w')$;
9 $\quad\quad$ **if** $d < D[v]$ **then continue**;
10 $\quad\quad$ **if** $d = D[v]$ **then**
11 $\quad\quad\quad$ append $(w, D[v], C[v])$ to $L^{nc}_{\leq}(v)$;
12 $\quad\quad$ **else**
13 $\quad\quad\quad$ append $(w, D[v], C[v])$ to $L^c_{\leq}(v)$;
14 $\quad\quad$ **for each** *neighbor v' of v* **do**
15 $\quad\quad\quad$ **if** $D[v'] = \infty \wedge w \leq v'$ **then**
16 $\quad\quad\quad\quad D[v'] \leftarrow D[v] + 1$; $C[v'] \leftarrow C[v]$;
17 $\quad\quad\quad\quad Q$.enqueue(v'); $R \leftarrow R \cup \{v'\}$;
18 $\quad\quad\quad$ **else if** $D[v'] = D[v] + 1$ **then**
19 $\quad\quad\quad\quad C[v'] \leftarrow C[v'] + C[v]$;

$\quad$ /* reset $D[\cdot]$ and $C[\cdot]$ */
20 $\quad$ **for each** $v \in R$ **do** $D[v] \leftarrow \infty$; $C[v] \leftarrow 0$;
21 **for each** $v \in V$ **do** $L_{\leq}(v) \leftarrow L^c_{\leq}(v) \cup L^{nc}_{\leq}(v)$;

Algorithm 2: Count (s, t)

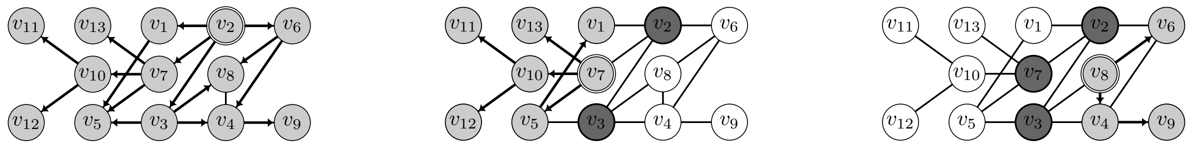
$\delta \leftarrow \infty$; $\sigma \leftarrow 0$;
for each $w \in L_{\leq}(s) \cap L_{\leq}(t)$ **do**
$\quad$ **if** $\mathsf{sd}(s, w) + \mathsf{sd}(t, w) < \delta$ **then**
$\quad\quad \delta \leftarrow \mathsf{sd}(s, w) + \mathsf{sd}(t, w)$;
$\quad\quad \sigma \leftarrow \sigma_{s,w} \cdot \sigma_{t,w}$;
$\quad$ **else if** $\mathsf{sd}(s, w) + \mathsf{sd}(t, w) = \delta$ **then**
$\quad\quad \sigma \leftarrow \sigma + \sigma_{s,w} \cdot \sigma_{t,w}$;
return σ;

3.3 Query Evaluation

According to Equation (1) and Equation (2), we can use Count (Algorithm 2) to compute the # of shortest paths between vertices s and t. Assuming that the hubs in each $L_{\leq}(v)$ are arranged in descending order of rank, Count can be implemented to run in $O(|L_{\leq}(s)| + |L_{\leq}(t)|)$ time.

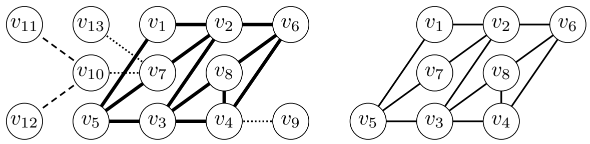
3.4 Vertex Ordering

The order $\leq$ is crucial for HP-SPC in that it remarkably affects the indexing time, the index size and the query time of the resulting labeling. Intuitively, a good ordering scheme should rank vertices that cover more shortest paths higher so that later searches in HP-SPC can be pruned as early as possible, thereby reducing the number of label entries generated. Several heuristics [2, 6, 39] have been studied in the



(a) Pushing v_2 **(b) Pushing v_7** **(c) Pushing v_8**

Figure 3: Illustration of HP-SPC



(a) **(b)**

Figure 4: 1-Shell Reduction

literature to obtain such orderings. For completeness, we review two state-of-the-art schemes, namely, degree-based [6, 32] and significant-path-based [5, 39], below.

Degree-Based Scheme. Intuitively, a vertex of higher degree is likely to cover more shortest paths. In view of this, in degree-based ordering, vertices are sorted in non-ascending order of degree. This scheme leads to the state-of-the-art canonical hub labeling for shortest distance queries [6].

Significant-Path-Based Scheme. Unlike the degree-based scheme, which uses only local information, the significant-path-based scheme is more adaptive. Let $w_1, w_2, \ldots, w_n$ 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 HP-SPC, a partial shortest path tree T_{w_i} rooted at w_i will be resulted. For each vertex v in T_{w_i}, let $\mathtt{des}(v)$ be the number of descendants and $\mathtt{par}(v)$ be the parent of v. Starting from w_i, the scheme computes a significant path $p_{\mathtt{sig}}$ to a leaf by iteratively selecting a child v with the largest $\mathtt{des}(v)$. Intuitively, $p_{\mathtt{sig}}$ is a path that many shortest paths cross. Then, among all vertices on $p_{\mathtt{sig}}$ other than w_i, the vertex v with the largest $\mathtt{deg}(v)\cdot(\mathtt{des}(\mathtt{par}(v))-\mathtt{des}(v))$ is empirically selected as w_{i+1}. Initially, w_1 is set as the one with the largest degree in G.

4 INDEX SIZE REDUCTION

We next discuss 3 index reduction techniques, namely 1-shell reduction (Section 4.1), neighborhood-equivalence reduction (Section 4.2) and independent-set reduction (Section 4.3), to reduce the index size.

4.1 Reduction by 1-Shell

In trees, the number of shortest paths between any two vertices is exactly one. Hence, the problem of shortest path counting is trivial for trees. Although graphs are generally much more complicated than trees, it is always possible to decompose a graph G into a core-fringe structure where the fringe, if not empty, consists of trees, as will be shown in the following. Better still, each such tree connects to the rest of G via at most one edge. As a result, it is safe to cut the fringe from G without breaking the shortest paths within the core. Here, the fringe is defined to be the 1-shell of G. Specifically, the 1-shell of G is defined as the subgraph induced by the vertices that belong to the 1-core but not to the 2-core, where the k-core of G is defined as the maximal subgraph within which each vertex is incident to at least k edges.

Example 4.1. Figure 4a illustrates the cores and the 1-shell of G (Figure 2a). Specifically, the 1-core is G itself, and the 2-core is the subgraph induced by $\{v_1, \ldots, v_8\}$. Thus, $V_1 = \{v_9, \ldots, v_{13}\}$ are those that reside in the 1-core but not in the 2-core. The 1-shell of G is induced by V_1 and consists of 3 connected components, *i.e.*, $\{v_{10}, v_{11}, v_{12}\}$, $\{v_9\}$ and $\{v_{13}\}$.

It can be verified that each connected component $\mathtt{cc}$ in the 1-shell is a tree. Moreover, $\mathtt{cc}$ either is attached to the 2-core of G via an edge or is isolated with the rest of G. For example, in Figure 4a, the 3 components in the 1-shell connect to the 2-core via the 3 dotted edges separately. If $\mathtt{cc}$ is connected to the 2-core, let $a(\mathtt{cc})$ be the corresponding access vertex within the 2-core; that is, some vertex a' in $\mathtt{cc}$ is adjacent to $a(\mathtt{cc})$ and the edge $(a(\mathtt{cc}), a')$ is the only edge that connects $\mathtt{cc}$ and the 2-core. In the rare case that $\mathtt{cc}$ is isolated, let $a(\mathtt{cc})$ be any vertex in $\mathtt{cc}$. For instance, in Figure 4a, $a(\{v_{10}, v_{11}, v_{12}\}) = a(\{v_{13}\}) = v_7$ and $a(\{v_9\}) = v_4$.

For each vertex v in G, we define its 1-shell-based representative $\mathtt{shr}(v)$ as follows. If v is not a vertex in the 1-shell, $\mathtt{shr}(v)$ is set to v itself. Otherwise, $\mathtt{shr}(v)$ is set to $a(\mathtt{cc})$, where *cc* here is the connected component that contains v in the 1-shell. For instance, in Figure 4a, $\mathtt{shr}(v_i) = v_i$ for $1 \le i \le 8$, $\mathtt{shr}(v_i) = v_7$ for $10 \le i \le 13$, and $\mathtt{shr}(v_9) = v_4$.

Graph Reduction. We compress G by removing from G all the vertices v with $\mathtt{shr}(v) \neq v$ and their incident edges. The resulting graph is denoted by G_s. It can be verified that only the vertices in the 1-shell is eligible to be eliminated and $\mathtt{shr}(v)$ is not removed for any vertex v. Figure 4b shows the resulting graph G_s after compressing G (Figure 4a).

Lemma 4.2. *For any vertices s and t, the number of shortest paths between s and t in G is the same as that between* $\mathsf{shr}(s)$ *and* $\mathsf{shr}(t)$ *in G_s, i.e.,* $\mathsf{spc}_G(s,t) = \mathsf{spc}_{G_s}(\mathsf{shr}(s), \mathsf{shr}(t))$.

Proof. We assume s and t are connected, since the other case is trivial. If s and t belong to the same component cc in the 1-shell of G, $\mathsf{spc}_G(s,t) = 1$ because cc is a tree. Also, $\mathsf{spc}_{G_s}(\mathsf{shr}(s), \mathsf{shr}(t)) = 1$ because $\mathsf{shr}(s) = \mathsf{shr}(t)$. Therefore, the lemma holds. In the other case where s and t are not in the same component, a shortest path $p_{s,t}$ between s and t in G must be of the form $(s, \ldots, \mathsf{shr}(s), \ldots, \mathsf{shr}(t), \ldots, t)$. Due to its optimality, the subpath p' connecting $\mathsf{shr}(s)$ and $\mathsf{shr}(t)$ in $p_{s,t}$ contains no vertices from the 1-shell and thus is a shortest path in G_s. Since there is exactly one path between s (resp. t) and $\mathsf{shr}(s)$ (resp. $\mathsf{shr}(t)$), a one-to-one correspondence exists between $p_{s,t}$ and p'. Hence, $\mathsf{spc}_G(s,t) = \mathsf{spc}_{G_s}(\mathsf{shr}(s), \mathsf{shr}(t))$. The lemma thus is proved. □

Query Evaluation. We process a query (s, t) as follows. If $\mathsf{shr}(s) = \mathsf{shr}(t)$, 1 is directly returned; otherwise, a query $(\mathsf{shr}(s), \mathsf{shr}(t))$ is issued on G_s and its result is returned.

4.2 Reduction by Equivalence Relation

Given an undirected graph G and two vertices u and v, if u and v share the same neighborhood, it is not hard to verify that $\mathsf{spc}_G(u, w) = \mathsf{spc}_G(v, w)$ for $\forall w \neq u, v$. Indeed, we can convert a shortest path from u to w to one from v to w by replacing u with v, and vice versa. Therefore, in this case, the label of v alone would suffice to answer any query between $\{u, v\}$ and other vertices. In other words, it is sound to disregard the label of u, reducing the index size. Formally, we say u is neighborhood equivalent to v, denoted by $u \equiv v$, if $\mathsf{nbr}(u)\backslash\{v\} = \mathsf{nbr}(v)\backslash\{u\}$. That is, if u and v are not adjacent, they are required to possess the same set of neighbors; otherwise, the neighbors of u and v should be the same after excluding u and v. For example, in Figure 4b, (i) $v_1 \equiv v_7$ since $\mathsf{nbr}(v_1)\backslash\{v_7\} = \mathsf{nbr}(v_7)\backslash\{v_1\} = \{v_2, v_5\}$, and (ii) $v_4 \equiv v_8$ since $\mathsf{nbr}(v_4)\backslash\{v_8\} = \mathsf{nbr}(v_8)\backslash\{v_4\} = \{v_3, v_6\}$. It can be verified that $\equiv$ is an equivalence relation. We remark that this equivalence has been employed in problems such as subgraph isomorphism [46] and distance hub labeling [38] for graph reduction. As will be shown, the relation can also be applied to our problem to reduce graph, thereby reducing index size. However, direct application of it without modification can lead to excessively underestimated results.

The equivalence relation $\equiv$ partitions V into equivalence classes. In particular, each non-singleton equivalence class ec either is an independent set or induces a clique. For example, in Figure 4b, $\equiv$ partitions V into 6 equivalence classes, namely $\{v_1, v_7\}$, $\{v_4, v_8\}$, $\{v_2\}$, $\{v_3\}$, $\{v_5\}$ and $\{v_6\}$, among which $\{v_1, v_7\}$ is an independent set and $\{v_4, v_8\}$ induces a clique. Within each equivalence class, we select the vertex with the minimum ID as the representative of the class. For notational convenience, for each vertex $v \in V$, we denote by $\mathsf{eqc}(v)$ the equivalence class containing v and by $\mathsf{eqr}(v)$ the representative of $\mathsf{eqc}(v)$. As an example, in Figure 4b, we have (i) $\mathsf{eqc}(v_i) = \{v_1, v_7\}$, $\mathsf{eqr}(v_i) = v_1$ for $i \in \{1, 7\}$, (ii) $\mathsf{eqc}(v_i) = \{v_4, v_8\}$, $\mathsf{eqr}(v_i) = v_4$ for $i \in \{4, 8\}$, and (iii) $\mathsf{eqc}(v_i) = \{v_i\}$, $\mathsf{eqr}(v_i) = v_i$ for $i \in \{2, 3, 5, 6\}$.

Lemma 4.3. *For two vertices s and t with $s \neq t$, if* $\mathsf{eqr}(s) \neq \mathsf{eqr}(t)$, *then* $\mathsf{spc}_G(s,t) = \mathsf{spc}_G(\mathsf{eqr}(s), \mathsf{eqr}(t))$; *otherwise,*

$$\mathsf{spc}_G(s,t) = \begin{cases} 1 & \textit{if } \mathsf{eqc}(s) \textit{ induces a clique} \\ \mathsf{deg}(s) & \textit{if } \mathsf{eqc}(s) \textit{ is an independent set} \end{cases}$$

By Lemma 4.3, it suffices to focus on the representatives.

Graph Reduction. With $\mathsf{eqr}(\cdot)$ at hand, we compress G by eliminating from G all the vertices v with $\mathsf{eqr}(v) \neq v$ and their incident edges. That is, only the representatives of all the equivalence classes are reserved. The resulting graph is denoted by G_e. Figure 2b shows the resulting graph after compressing G_s (Figure 4b) as above. It is worth emphasizing that while such a reduction is able to preserve the shortest distance between vertices [38], it fails to preserve the # of shortest paths. To see this, consider Figures 4b and 2b as an example. There are 3 shortest paths between v_2 and v_5 (Figure 4b), but only two of them are preserved after reduction (Figure 2b). To resolve this issue, we propose to associate each shortest path p in G_e with a weight $\lambda(p)$ such that

$$\mathsf{spc}_G(s', t') = \textstyle\sum_{p \in P_{s',t'} \text{ in } G_e} \lambda(p) \tag{3}$$

for any representatives s' and t'. That is, the # of shortest paths between s' and t' in G can be obtained by summing the weights of the shortest paths between s' and t' in G_e. To this end, we charge each shortest path between s' and t' in G to one between s' and t' in G_e. Specifically, for a shortest path $p' = (s' = v_0, v_1, \ldots, v_\ell = t')$ in G, we charge it to $p = (s', \mathsf{eqr}(v_1), \ldots, t')$. By definition, if there is an edge between v_i and v_{i+1}, there is also an edge between $\mathsf{eqr}(v_i)$ and $\mathsf{eqr}(v_{i+1})$. Therefore, p is a path in G_e with the same endpoints as p'. Moreover, due to the optimality of p', p is also a shortest path in G_e. For example, we charge both (v_2, v_1, v_5) and (v_2, v_7, v_5) in Figure 4b to (v_2, v_1, v_5) in Figure 2b.

For a shortest path $p = (s', v_1', v_2' \ldots, v_{\ell-1}', t')$ in G_e, it can be verified that there are $c_1 \times c_2 \times \cdots \times c_{\ell-1}$ shortest paths in G, including itself, charged to it, where $c_i = |\mathsf{eqc}(v_i')|$ is the # of vertices in the equivalence class containing v_i'. Accordingly, we let $\lambda(p) = c_1 \times c_2 \times \cdots \times c_{\ell-1}$. Specially, if $\mathtt{len}(p) \leq 1$, $\lambda(p) = 1$. For instance, we have $\lambda((v_2, v_1, v_5)) = 2$ and $\lambda((v_2, v_3, v_5)) = 1$ in Figure 2b. Since each shortest path between s' and t' in G is charged to exactly one shortest path with the same endpoints in G_e, Equation (3) holds.

Suppose that v' is a vertex on some shortest path between s' and t' in G_e. Let P_1 and P_2 be collections of shortest paths from s' to v' and from v' to t' in G_e, respectively. Let $P = \{p_1 \odot p_2 \mid p_1 \in P_1 \wedge p_2 \in P_2\}$ be the shortest paths obtained by concatenating the paths in P_1 with those in P_2. We have:

Lemma 4.4. *If $v' \neq s'$ and $v' \neq t'$, then*

$$\sum_{p \in P} \lambda(p) = \left(\sum_{p_1 \in P_1} \lambda(p_1)\right) \cdot \left(\sum_{p_2 \in P_2} \lambda(p_2)\right) \cdot |\mathsf{eqc}(v')|$$

Proof. For $\forall p_1 \in P_1$ and $\forall p_2 \in P_2$, it can be verified that $\lambda(p_1 \odot p_2) = \lambda(p_1) \cdot \lambda(p_2) \cdot |\mathsf{eqc}(v')|$. Therefore,

$$\begin{aligned}\sum_{p \in P} \lambda(p) &= \sum_{p_1 \in P_1} \sum_{p_2 \in P_2} \lambda(p_1 \odot p_2) \\ &= \sum_{p_1 \in P_1} \sum_{p_2 \in P_2} \lambda(p_1) \cdot \lambda(p_2) \cdot |\mathsf{eqc}(v')| \\ &= \left(\sum_{p_1 \in P_1} \lambda(p_1)\right) \cdot \left(\sum_{p_2 \in P_2} \lambda(p_2)\right) \cdot |\mathsf{eqc}(v')|. \ \square\end{aligned}$$

Adapting HP-SPC. We adapt HP-SPC based on Lemma 4.4 as follows. For each $(w, C_{v,w})$ where $C_{v,w}$ is the set of trough shortest paths from v to w, a label entry $(w, \mathsf{sd}(v, w), \sigma_{v,w})$ is generated with $\sigma_{v,w}$ set to $\sum_{p \in C_{v,w}} \lambda(p)$ rather than $|C_{v,w}|$. Accordingly, we also need to adapt the query evaluation algorithm. According to Lemma 4.3, we process a query (s, t) with $s \neq t$ as follows. If $\mathsf{eqr}(s) = \mathsf{eqr}(t)$, $\mathsf{deg}(s)$ is returned if $\mathsf{eqc}(s)$ is an independent set; otherwise, 1 is returned. If $\mathsf{eqr}(s) \neq \mathsf{eqr}(t)$, we capitalize on the labeling $L_{\leq}(\cdot)$ constructed on G_e. Specifically, we initialize the final result σ to 0. Then, based on Equation (3) and Lemma 4.4, for each common hub w that is on some shortest path between $\mathsf{eqr}(s)$ and $\mathsf{eqr}(t)$, if $w \notin \{\mathsf{eqr}(s), \mathsf{eqr}(t)\}$, we add $\sigma_{\mathsf{eqr}(s),w} \cdot \sigma_{\mathsf{eqr}(t),w} \cdot |\mathsf{eqc}(w)|$ to σ; otherwise, we add $\sigma_{\mathsf{eqr}(s),w} \cdot \sigma_{\mathsf{eqr}(t),w}$ instead.

4.3 Reduction by Independent Set

Let I be an independent set of G. For any vertex v, let $R_v = \mathsf{nbr}(v)$ if $v \in I$ and $R_v = \{v\}$ otherwise. For two vertex sets $S_1, S_2 \subseteq V$, we define their distance as $\mathsf{sd}(S_1, S_2) = \min_{v_1 \in S_1, v_2 \in S_2} \mathsf{sd}(v_1, v_2)$. Additionally, we define the number of shortest paths between S_1 and S_2, denoted $\mathsf{spc}(S_1, S_2)$, to be the # of paths of length $\mathsf{sd}(S_1, S_2)$ between the vertices of S_1 and the vertices of S_2. For any query (s, t), $\mathsf{spc}(s, t) = \mathsf{spc}(R_s, R_t)$. Indeed, for each shortest path p_{v_1,v_2} with length $\mathsf{sd}(R_s, R_t)$ between $v_1 \in R_s$ and $v_2 \in R_t$, $(s, v_1) \odot p_{v_1,v_2} \odot (v_2, t)$ is a shortest path between s and t, and vice versa. Therefore, for a vertex $v \in I$, we can drop its label and resort to the labels of its neighbors for query evaluation.

Intuitively, vertices with lower ranks are likely to have larger labels in $L_{\leq}(\cdot)$. Therefore, following [38], we select $I = \{v \in V \mid \forall u \in \mathsf{nbr}(v), u \leq v\}$. For $\forall v \in I$ selected this way, v is not a hub of any vertex but itself. Below, we describe how HP-SPC is adapted when independent-set-based reduction with I selected as above is applied. (1) Under degree-based ordering, for $\forall v \in I$, we do not generate entries of $L_{\leq}(v)$; that is, $L_{\leq}(v)$ remains empty in the course of computation. Such a modification does not affect the correctness of the algorithm. Moreover, since the partial distance evaluations (line 8 of Algorithm 1) that involve I are avoided, the indexing time may decrease (Section 6). (2) Under significant-path-based ordering, the order is not known before the algorithm starts. Therefore, we cannot predetermine I and we discard $L_{\leq}(v)$ of a vertex v only after v is verified to be in I.

Query Evaluation. Next, we discuss two query schemes to deal with a query (s, t). We assume *w.l.o.g.* that $s, t \in I$ and $s \neq t$. In this case, we have $R_s = \mathsf{nbr}(s)$ and $R_t = \mathsf{nbr}(t)$.

The `direct` scheme. In this scheme, to handle a query (s, t), a hash join is conducted between the labels of R_s and the labels of R_t. Specifically, let $H_d[\cdot]$ and $H_c[\cdot]$ be two arrays. For a vertex w that is a hub of some vertex in R_s, (1) $H_d[w]$ records the minimum label distance between w and R_s; that is, $H_d[w] = \min_{v \in R_s \mid w \text{ is a hub of } v} \mathsf{sd}(v, w)$; and (2) $H_c[w]$ records the aggregated label count between w and R_s; that is, $H_c[w] = \sum_{v \in R_s \mid w \text{ is a hub of } v,\ \mathsf{sd}(v,w) = H_d[w]} \sigma_{v,w}$. With both $H_d[\cdot]$ and $H_c[\cdot]$ in hand, for each label entry $(w, \cdot, \cdot)$ of the vertices in R_t, we inspect $H_d[w]$ and $H_c[w]$ to update the tentative shortest distance and shortest path count. Therefore, the time complexity is $O(\sum_{v \in R_s \cup R_t} |L_{\leq}(v)|)$.

The `filtered` scheme. Recall that $L_{\leq}(\cdot)$ comprises $L^c_{\leq}(\cdot)$ and $L^{nc}_{\leq}(\cdot)$, where $L^c_{\leq}(\cdot)$ keeps track of all the necessary information for distance queries. We observe that the size of $L^{nc}_{\leq}(\cdot)$ can be several times larger than that of $L^c_{\leq}(\cdot)$. For example, for the 10 graphs tested (Table 3), the size of $L^{nc}_{\leq}(\cdot)$ is 3.01 - 14.65 times as large as that of $L^c_{\leq}(\cdot)$ (Section 6). For this reason, in query evaluation, we should reduce the frequency to touch the large $L^{nc}_{\leq}(\cdot)$ part. To this end, we store $L^c_{\leq}(\cdot)$ and $L^{nc}_{\leq}(\cdot)$ separately rather than merging them into $L_{\leq}(\cdot)$. Let $R_s(t) = R_s \cap Q_{s,t}$. That is, $R_s(t)$ is the set of s's neighbors that lie on the shortest paths between s and t. Similarly we define $R_t(s)$. By definition, we have $\mathsf{spc}(s, t) = \mathsf{spc}(R_s(t), R_t(s))$. The `filtered` scheme consists of two phases. In the first phase, we identify $R_s(t)$ and $R_t(s)$ by hash-joining the $L^c_{\leq}(\cdot)$ labels of R_s and those of R_t with the help of $H_d[\cdot]$. Then, in the second phase, we obtain the final count by conducting a hash join between $R_s(t)$ and $R_t(s)$ using both $L^c_{\leq}(\cdot)$ and $L^{nc}_{\leq}(\cdot)$ labels. Note that the $L^{nc}_{\leq}(\cdot)$ labels of the vertices in $(R_s \cup R_t) \backslash (R_s(t) \cup R_t(s))$ are not touched. The time complexity thus is $O(\sum_{v \in R_s \cup R_t} |L^c_{\leq}(v)| + \sum_{v \in R_s(t) \cup R_t(s)} |L_{\leq}(v)|)$.

5 THEORETICAL RESULTS ON SPECIAL GRAPH CLASSES

In this section, we show that HP-SPC is able to exploit the intrinsic dimension of a graph if it is fed with an appropriate order. A hub labeling $L_{\leq}(\cdot)$ is said to be (α, β) bounded if $\sum_v |L_{\leq}(v)| = O(\alpha)$ and $\max_v |L_{\leq}(v)| = O(\beta)$. We remark that such a labeling requires only $O(\alpha)$ space and provides $O(\beta)$ query time. In the following, we focus on planar graphs

(Section 5.1), graphs with small treewidths (Section 5.2) and graphs of small highway dimension (Section 5.3).

5.1 Planar Graphs

The main result of this section is as follows.

THEOREM 5.1. *For a planar graph with n vertices, there is a labeling $L_{\leq}(\cdot)$ that is $(n^{1.5}, \sqrt{n})$ bounded.*

According to the planar separator theorem [35, 40, 41], in any n-vertex planar graph G, there is a partition of V into sets A, S and B such that (i) $\max\{|A|, |B|\} \leq 2n/3$, (ii) $|S| = O(\sqrt{n})$, and (iii) there does not exist an edge with one endpoint in A and the other in B. Here, S is called the separator of this partition. By recursively applying the separator theorem on G, we can obtain a binary tree $\mathcal{T}$ where each node corresponds to a separator. Particularly, the root of $\mathcal{T}$ is the separator S and its children are the separators of A and B separately. Let $\leq$ be the order that the vertices are visited when performing a preorder traversal on $\mathcal{T}$, breaking ties arbitrarily. Consider the labeling $L_{\leq}(\cdot)$ induced by $\leq$. It can be verified that, for a vertex v in a node t, only the vertices from t and the ancestors of t can be hubs in $L_{\leq}(v)$. Let $s(n)$ be the maximum size that a label of a vertex can reach on n-vertex graphs when constructed as above. Then, we have $s(n) \leq s(\lfloor 2n/3 \rfloor) + O(\sqrt{n})$, implying $s(n) = O(\sqrt{n})$. Therefore, $\max_v |L_{\leq}(v)| = O(\sqrt{n})$ and $\sum_v |L_{\leq}(v)| = O(n^{1.5})$.

We compare the above algorithm, denoted $\texttt{HP-SPC}_\texttt{P}$, with $\texttt{PL-SPC}$ [12] below. In $\texttt{PL-SPC}$, $\mathcal{T}$ is also constructed though implicitly. Based on $\mathcal{T}$, for a vertex v in a node t, the vertices from t and the ancestors of t are all appended to the label of v in $\texttt{PL-SPC}$, as long as v is reachable from them via breadth-first searches. Therefore, although $\texttt{HP-SPC}_\texttt{P}$ has the same asymptotic bound as $\texttt{PL-SPC}$ on label size, the hubs generated by $\texttt{HP-SPC}_\texttt{P}$ are only a subset of those by $\texttt{PL-SPC}$. In terms of indexing time, $\texttt{HP-SPC}_\texttt{P}$ may perform worse than $\texttt{PL-SPC}$ because although $\texttt{HP-SPC}_\texttt{P}$ is able to visit less vertices, it additionally needs to do many partial distance evaluations.

5.2 Graphs of Small Treewidth

The main result of this section is as follows.

THEOREM 5.2. *For a graph with treewidth ω, there is a labeling $L_{\leq}(\cdot)$ that is $(\omega n \log n, \omega \log n)$ bounded.*

Formally, a *tree decomposition* of G is a pair $(X, \mathcal{T})$, where $\mathcal{T}$ is a tree in which each node t is associated with a subset X_t, called a bag, of V and $X = \{X_t \mid t \in \mathcal{T}\}$. $(X, \mathcal{T})$ must satisfy: (i) $V = \bigcup_{X_t \in X} X_t$; (ii) for each edge $(u, v) \in E$, there exists a bag X_t such that $\{u, v\} \subseteq X_t$; and (iii) for each vertex u, the nodes whose bags contain u form a connected subgraph of $\mathcal{T}$. The *width* of a tree decomposition $(X, \mathcal{T})$ is the size of the largest bag minus 1. The *treewidth* of a graph is the minimum width among all possible tree decompositions of G. Consider a graph with treewidth ω and a tree decomposition $(X, \mathcal{T})$ of G with width ω. Without loss of generality, assume that there do not exist two nodes t_1 and t_2 such that $X_{t_1} \subseteq X_{t_2}$ or $X_{t_2} \subseteq X_{t_1}$. Let $\mathcal{T}'$ be the centroid decomposition of $\mathcal{T}$. As a result, (i) $\mathcal{T}'$ has the same nodes as $\mathcal{T}$, but the positions of the nodes may be reorganized; (ii) the height of $\mathcal{T}'$ is bounded by $O(\log n)$; and (iii) any path in G between a vertex in node $t_1 \in \mathcal{T}'$ and a vertex in node $t_2 \in \mathcal{T}'$ must contain some vertex in the node t^*, where t^* is the least common ancestor of t_1 and t_2 in $\mathcal{T}'$.

Next, we define the order. For each node t in $\mathcal{T}'$, remove from X_t those vertices that appear in the ancestor nodes of t. By doing so, each vertex $v \in V$ is guaranteed to appear in exactly one X_t. That is, the resulting X is a partition of V. Note that $|X_t| \leq \omega + 1$ still holds. Define $\leq$ such that vertices in an ancestor node have higher ranks than vertices in a descendant node. As a result, for a vertex v in node t, only the vertices A in the ancestor nodes of t may be hubs of v. This is because for any other vertex u with $u \leq v$, any shortest path between u and v must pass through one vertex w of A with $w \leq u$. We thus conclude that $L_{\leq}(v)$ is bounded by $O(\omega \log n)$, and thus the size of $L_{\leq}(\cdot)$ is $O(\omega n \log n)$.

5.3 Graphs of Small Highway Dimension

The main theorem of this section is as follows.

THEOREM 5.3. *For a graph G with highway dimension h, there is a labeling $L_{\leq}(\cdot)$ that is $(nh \log D, h \log D)$ bounded, where D is the diameter of G.*

For a vertex v and a positive r, let $B_{v,r}$ denote the ball of radius r centered at v; that is, $B_{v,r} = \{u \in V \mid \texttt{sd}(v, u) \leq r\}$. Following [3], a set C of vertices is called an (r, k)-*shortest-path cover* ((r, k)-$\texttt{SPC}$) if (i) $\forall v \in V$, $|C \cap B_{v,2r}| \leq k$; and (ii) for any shortest path p with $r < \texttt{len}(p) \leq 2r$, $p \cap C \neq \emptyset$. Intuitively, an (r, k)-$\texttt{SPC}$ has a bounded overlap with any ball of radius $2r$ and is able to cover all shortest paths of length between r and $2r$. Based on the notion of (r, k)-$\texttt{SPC}$, the *highway dimension* of a graph G can be defined as the smallest integer h such that an (r, h)-$\texttt{SPC}$ exists for any $r > 0$.

Consider a graph G with highway dimension h. We follow Abraham et al.'s method [3] to construct an order. In detail, let C_i be a $(2^i, h)$-$\texttt{SPC}$ for $-1 \leq i \leq \log D$, where D is the diameter of G. Specially, let $C_{-2} = V$. Let $L_i = C_i \setminus \bigcup_{j=i+1}^{\log D} C_j$ for $-2 \leq i \leq \log D$. One can easily verify that L_i's form a partition of V. With L_i's at hand, we define $\leq$ such that the vertices in L_i have lower ranks than the vertices in L_{i+1}.

Consider the resulting labeling $L_{\leq}(\cdot)$ induced by $\leq$. For a vertex v, let $w \in L_{\leq}(v)$ be a hub from L_i for some $i \geq -1$. We show $\texttt{sd}(v, w) \leq 2^{i+1}$ below. Assume the opposite. That is, $2^j < \texttt{sd}(v, w) \leq 2^{j+1}$ for some $j \geq i + 1$. Then, by the definition of (r, k)-$\texttt{SPC}$, each shortest path p between v and w

Graph	Notation	n	m	BFS Time
Facebook[2]	FB	63,731	817,035	7.59 ms
Gowalla[3]	GW	196,591	950,327	13.25 ms
WikiConflict[2]	WI	118,100	2,027,871	14.60 ms
Google[3]	GO	875,713	4,322,051	95.01 ms
DBLP[2]	DB	1,314,050	5,362,414	176.10 ms
Berkstan[3]	BE	685,230	6,649,470	48.73 ms
Youtube[2]	YT	3,223,589	9,375,374	432.62 ms
Petster[2]	PE	623,766	15,695,166	129.73 ms
Flickr[2]	FL	2,302,925	22,838,276	622.98 ms
Indochina[1]	IN	7,414,866	150,984,819	1010.68 ms

Table 3: The Statistics of the Graphs

is covered by C_j, whose vertices all have higher ranks than v and w with respect to $\leq$. In other words, there exist no trough shortest paths between v and w, indicating w cannot be a hub of v. A contradiction. We thus have proved $\mathsf{sd}(v, w) \leq 2^{i+1}$. Since there can be at most h vertices from C_i in $B_{v,2^{i+1}}$, the number of hubs from L_i in $L_{\leq}(v)$ is bounded by h. Therefore, we have $|L_{\leq}(v)| = O(h \log D)$ for any vertex v, implying $\sum_v |L_{\leq}(v)| = O(nh \log D)$.

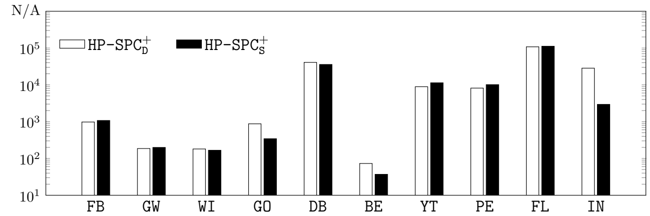
6 EXPERIMENTAL EVALUATION

We conducted experiments to evaluate our algorithms. The experiments were carried out on a Linux machine with Intel Xeon E7-8891 CPU and 250 GB main memory.

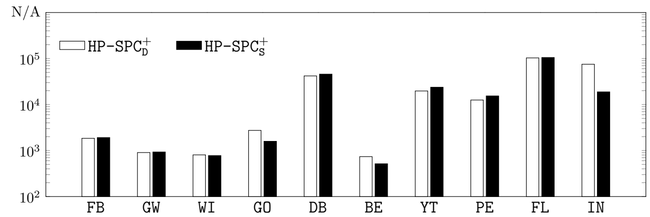
Datasets. We used 10 publicly available datasets, shown in Table 3. The largest dataset Indochina is a web graph from LAW[1]. The remaining 9 graphs, downloaded from KONECT[2] and SNAP[3], include an interaction network (WikiConflict), a coauthorship network (DBLP), a location-based social network (Gowalla), 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, 1,000,000 random queries were employed and the average time is reported.

Algorithms. We compared the following algorithms in our experiments: (i) HP-SPC (Algorithm 1), (ii) HP-SPC$^+$ (HP-SPC with both shell reduction (Section 4.1) and neighborhood-equivalence reduction (Section 4.2)), and (iii) HP-SPC*, which is obtained by applying the independent-set reduction (Section 4.3) to HP-SPC$^+$. HP-SPC* uses the filtered (Section 4.3) query scheme by default. Depending on the underlying order, we will add either D or S as a subscript to each notation. For example, HP-SPC$^+_{\mathsf{D}}$ and HP-SPC$^+_{\mathsf{S}}$ denote HP-SPC$^+$ under the degree-based order and the significant-path-based order, respectively. All algorithms above were implemented in C++ and compiled by g++ at -O3 optimization level. In the current implementation, each label entry $(w, \mathsf{sd}(v, w), \sigma_{v,w})$ is encoded in a 64-bit integer. Specifically, w, $\mathsf{sd}(v, w)$ and $\sigma_{v,w}$ are encoded using 23, 10 and 31 bits, respectively. In the rare case that $\sigma_{v,w}$ is greater than $2^{31} - 1$, it is treated as $2^{31} - 1$.

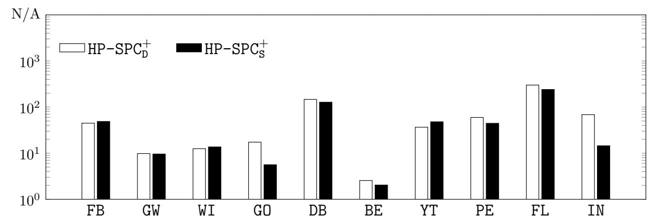
[1]http://law.di.unimi.it
[2]http://konect.uni-koblenz.de
[3]https://snap.stanford.edu



(a) Index Construction Time (Seconds)



(b) Index Size (MB)

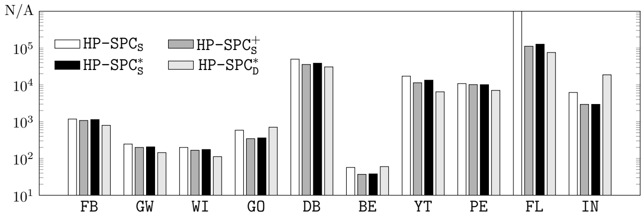


(c) Query Time (Microseconds)

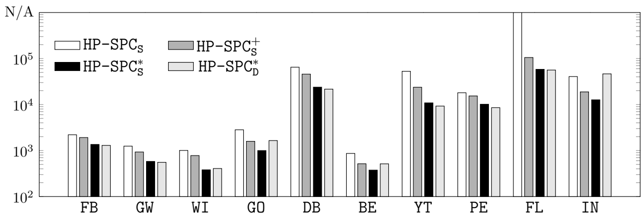
Figure 5: Degree-Based Versus Significant-Path-Based

Exp-1: Vertex Ordering. We compared the degree-based ordering and the significant-path-based ordering in this testing. Figure 5 shows the results of HP-SPC$^+_{\mathsf{D}}$ and HP-SPC$^+_{\mathsf{S}}$ on all the 10 graphs. The results of HP-SPC and HP-SPC* are similar and thus are omitted here. As can be observed in Figure 5, on graphs GO, BE and IN, HP-SPC$^+_{\mathsf{S}}$ performs much better than HP-SPC$^+_{\mathsf{D}}$. Indeed, it requires $\geq$ 49.3% less index construction time, $\geq$ 30.0% less index space, and $\geq$ 20.0% less query time. Particularly, on IN, HP-SPC$^+_{\mathsf{S}}$ reduces the index construction time, the index space, and the query time by 89.6%, 75.0% and 78.9%, respectively. For the remaining graphs, HP-SPC$^+_{\mathsf{S}}$ and HP-SPC$^+_{\mathsf{D}}$ provide comparable performance. Overall the significant-path-based ordering is better.

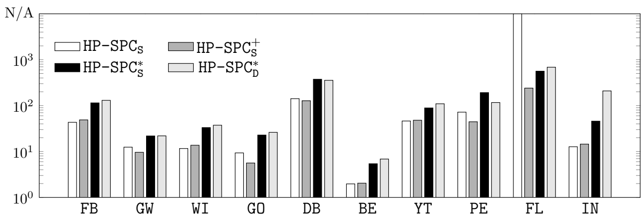
Exp-2: Performance Evaluation. We evaluated the performance of HP-SPC and the 3 index reduction techniques under the significant-path-based ordering. The corresponding results are summarized in Figure 6. We also include the results for HP-SPC$^*_{\mathsf{D}}$ to show the influence of vertex ordering on the independent-set-based reduction. We do not present



(a) Index Construction Time (Seconds)



(b) Index Size (MB)



(c) Query Time (Microseconds)

Figure 6: Performance Evaluation

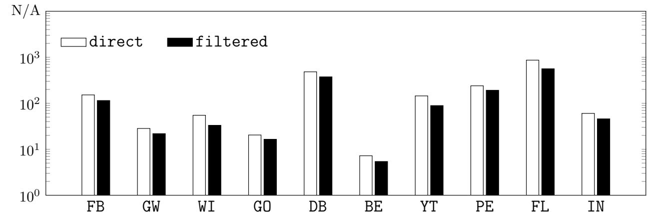


Figure 7: Comparison of Query Schemes

the results of HP-SPC$_S$ on FL since it ran out of memory. For one thing, the index generated by HP-SPC$_S$ on FL is huge. For another, for convenience, we used `std::vector` from C++ STL to store label entries for each vertex. Each time there is no more room to append a new label entry, `std::vector` will enlarge the underlying storage by a factor of 2, thereby leading to much allocated-but-unused memory space.

Index Construction Time. Figure 6a shows the index construction time taken by HP-SPC$_S$, HP-SPC$_S^+$ and HP-SPC$_S^*$. The results show that (i) across the graphs tested, HP-SPC$_S^+$ is from 1.06 to 2.11 times faster than HP-SPC$_S$; (ii) the index construction time of HP-SPC$_S^*$ is nearly the same as that of HP-SPC$_S^+$ since its only difference with HP-SPC$_S^+$ is that it removes the labels of a vertex once the vertex is confirmed to be in I; (iii) for most of the 10 graphs, their indices can be constructed by HP-SPC$_S^+$ in no more than 3 hours; particularly, for the largest graph IN, it takes only roughly 0.8 hours to compute the labeling; and (iv) the graphs FB, DB and FL are hard for HP-SPC$_S^+$ in that they take much more time than other graphs of similar size; however, so far, it is not clear to us what properties of these graphs lead to this inefficiency.

Index Space. Figure 6b shows the results on index size. We observe the following: (i) HP-SPC$_S^+$ is able to reduce the index size of HP-SPC$_S$ by at least 13.0% on the 10 graphs, indicating the effectiveness of the shell reduction and the equivalence reduction; particularly, for GO, BE, YT and IN, the index size of HP-SPC$_S$ is reduced by 43.8%, 40.4%, 55.0% and 53.6%, respectively; (ii) unlike HP-SPC$_S$, which runs out of memory, HP-SPC$_S^+$ is able to compute the index for FL; (iii) HP-SPC$_S^*$ further reduces the index size of HP-SPC$_S^+$ by from 27.1% to 54.3%; as a result, the index size of FL is further reduced from 103GB to 57GB and is much more practical than a count matrix; and (iv) overall, the 3 reduction techniques together can reduce the index size by from 38.4% to 79.4%.

Query Time. Figure 6c shows the average query time taken by HP-SPC$_S$, HP-SPC$_S^+$ and HP-SPC$_S^*$ over 1,000,000 random queries. As shown in the figure, the query time of HP-SPC$_S^+$ is comparable to that of HP-SPC$_S$. As for HP-SPC$_S^*$, while it is able to further reduce the index size, it requires much more time to deal with a query. Specifically, the query time of HP-SPC$_S^*$ is on average about 2.8 times of that of HP-SPC$_S^+$ on the 10 graphs tested. This is expected since HP-SPC$_S^*$ needs to aggregate the labels of the query vertices in the course of computation. We would like to emphasize that although more time is needed in a query evaluation, the query time of HP-SPC$_S^*$ remains of hundreds of microseconds. Therefore, it is still several orders of magnitude faster than breadth-first search in query evaluation (In Table 3, we show for each graph the average BFS time over 10,000 random queries).

HP-SPC$_S^*$ Versus HP-SPC$_D^*$. For graphs GO, BE and IN, on which HP-SPC$_S^+$ performs much better than HP-SPC$_D^+$ (see Exp-1), HP-SPC$_S^*$ is also much better than HP-SPC$_D^*$. As for the other 7 graphs, HP-SPC$_D^*$ requires less indexing time and index space at the cost of slightly worse query time overall.

Exp-3: Comparison of Query Schemes. In this testing, we compared the two query schemes of HP-SPC*, namely `filtered` and `direct`, discussed in Section 4.3. We show the results of HP-SPC$_S^*$ under these two schemes in Figure 7. The results of HP-SPC$_D^*$ are similar and thus are omitted here. As can be seen, compared with `direct`, the `filtered` query scheme can reduce the query time by from 19.2% to 39.4% since the $L_{\leq}^{nc}(\cdot)$ labels of some vertices are filtered out.

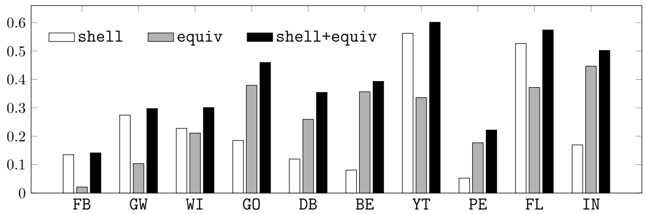


Figure 8: Shell Versus Equivalence

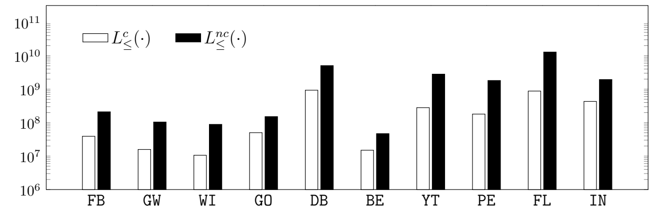
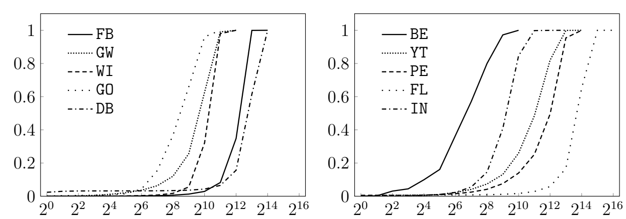


Figure 9: The Size of $L^c_\leq(\cdot)$ and $L^{nc}_\leq(\cdot)$

	40th	50th	60th	70th	80th	90th	Max
FB	1.00	1.25	1.50	1.78	2.19	3.10	49.67
GW	1.00	1.00	1.01	1.25	1.67	3.00	742.00
WI	1.00	1.00	1.21	1.50	2.00	3.39	457.00
GO	1.00	1.00	1.00	1.00	1.07	1.36	7,645.84
DB	1.00	1.00	1.25	1.50	2.00	2.67	45.33
BE	1.00	1.00	1.00	1.02	1.18	1.69	346.00
YT	1.00	1.03	1.33	1.82	2.76	6.78	4,735.00
PE	1.07	1.25	1.57	2.15	3.48	7.79	468.36
FL	1.00	1.25	1.56	2.00	2.90	5.11	885.50
IN	1.00	1.13	1.57	2.36	5.00	18.33	48,451.00

Table 4: Performance of $L^c_\leq(\cdot)$ in Counting



(a) The First 5 Graphs **(b) The Last 5 Graphs**

Figure 10: The Cumulative Distribution of $|L_\leq(v)|$

Exp-4: Shell Versus Equivalence. We evaluated the performance of the shell reduction (`shell`) and the equivalence reduction (`equiv`) in this experiment. Specifically, for each graph, we recorded the fraction of vertices removed after applying each of the following index reduction techniques: `shell`, `equiv`, and `shell+equiv`. The results are reported in Figure 8. We find that neither of `shell` and `equiv` is able to show consistent reduction power across the graphs. While `shell` can remove more than 50% vertices in YT and FL, it removes only 8% and 5% in BE and PE, respectively. A similar conclusion can also be obtained for `equiv`. In contrast, `shell+equiv` is more robust. Indeed, `shell+equiv` achieves the best reduction ratio for each graph tested. Moreover, it is able to remove at least 20% vertices for all graphs but one.

Exp-5: Analysis of $|L_\leq(\cdot)|$. Recall that $L_\leq(\cdot)$ comprises $L^c_\leq(\cdot)$ and $L^{nc}_\leq(\cdot)$. Although $L^c_\leq(\cdot)$ can be employed to compute the distance between any two vertices, it on its own is not sufficient for correct shortest path counting and should be supplemented with $L^{nc}_\leq(\cdot)$ to this end. In Figure 9, for each graph tested, we show the size of the $L^c_\leq(\cdot)$ and $L^{nc}_\leq(\cdot)$ generated by HP-SPC^+_S, *i.e.*, $\sum_v |L^c_\leq(v)|$ and $\sum_v |L^{nc}_\leq(v)|$, respectively. The results show that the size of $L^{nc}_\leq(\cdot)$ is from 3.01 to 14.65 times as large as that of $L^c_\leq(\cdot)$. That is to say, far more label entries are needed such that no shortest paths are missed.

Approximation. We conducted an experiment to test the performance of $L^c_\leq(\cdot)$ in shortest path counting. In detail, for each query (s, t), we computed an estimate $\mathsf{spc}_{\mathsf{approx}}(s, t)$ of $\mathsf{spc}(s, t)$ using only $L^c_\leq(\cdot)$ and recorded the ratio of $\mathsf{spc}(s, t)$ to $\mathsf{spc}_{\mathsf{approx}}(s, t)$. For each graph, we show in Table 4 the percentiles of the ratios over all the queries. We observe the following: (i) On all graphs but PE, correct results can be obtained from $L^c_\leq(\cdot)$ alone for 40% of the queries. As for PE, its 40th percentile is just 1.07. (ii) For the first 6 graphs, 90% of the queries have ratios below 3.39, and for the last 4 graphs, the ratios are as low as 5.00 for 80% of the queries. (iii) The ratios can be as high as hundreds for some queries. Specially, on GO and IN, the maximum ratio comes up to 7,645.84 and 48,451.00, respectively. To conclude, although $L^c_\leq(\cdot)$ is able to provide good approximations for a non-trivial part of the queries, it may significantly underestimate the true results in some cases. Worse yet, we cannot predetermine whether the result of a query is good. Adding some entries from $L^{nc}_\leq(\cdot)$ to $L^c_\leq(\cdot)$ may help to improve the accuracy. But thus far, we are unaware of a way to do this with a provable approximation guarantee. We consider this our future work.

The Distribution of $L_\leq(v)$. We show for each graph the cumulative distribution of $L_\leq(v)$ in Figure 10. As can be observed, for each graph, the sizes of the labels do not differ too much. For example, most of the labels of FB, DB and FL have sizes in the range $[2^{11}, 2^{13}]$, $[2^{12}, 2^{14}]$ and $[2^{13}, 2^{15}]$, respectively. Such an observation verifies the stability of our query evaluation scheme across different queries.

Exp-6: Comparison with [12]. In this testing, we compared HP-SPC with PL-SPC [12]. We denote the version of HP-SPC in Section 5.1 by HP-SPC$_P$. For both HP-SPC$_P$ and PL-SPC, we implemented the planar separator theorem following [35, 41]. We used the "Build Planar Graphs" script[4] to

[4]https://users.dcc.uchile.cl/ jfuentess/datasets/code.php

	Indexing Time	Index Space	Query Time
PL-SPC	0.59 hr	131.50 GB	94.10 µs
HP-SPC$_P$	7.06 hr	51.64 GB	54.23 µs
HP-SPC$_D$	0.72 hr	14.44 GB	25.63 µs
HP-SPC$_S$	1.02 hr	23.04 GB	39.22 µs

Table 5: Results on Delaunay

generate a planar graph by computing the Delaunay triangulation of n random plane coordinates. Unlike real graphs, the graphs generated this way contain an enormous number of shortest paths. For this reason, we set n to 500,000 such that for most pairs of vertices, the # of shortest paths fits in a 128-bit integer. The resulting graph, which contains 500,000 vertices and 1,499,980 edges, is called Delaunay. Accordingly, unlike in previous experiments, for a label entry we encoded w, $\mathsf{sd}(v, w)$ and $\sigma_{v,w}$ using 32, 32 and 128 bits, respectively. The results are shown in Table 5. We observe: (i) HP-SPC$_P$ requires about 12 times as much indexing time as PL-SPC; this is expected because starting from the same source as PL-SPC, HP-SPC$_P$ additionally needs to do many partial distance evaluations during the traversal; (ii) HP-SPC$_P$ results in smaller index size and query time than PL-SPC; indeed, the hubs generated by HP-SPC$_P$ are provably a subset of those by PL-SPC; and (iii) HP-SPC$_D$ and HP-SPC$_S$ show the best practical performance overall; however, unlike the others, no theoretical guarantee is known for them on planar graphs.

7 EXTENSION TO DIRECTED GRAPHS

For a weighted directed graph $G = (V, E, l(\cdot))$, where $l(e) > 0$ denotes the weight of an edge e, instead of $L_{\leq}(v)$, two labels $L^{in}_{\leq}(v)$ and $L^{out}_{\leq}(v)$ are computed for each vertex v. Specifically, a vertex w satisfying $w \leq v$ is picked as a hub in $L^{in}_{\leq}(v)$ (resp. $L^{out}_{\leq}(v)$) if and only if there exists at least one trough shortest path from w to v (resp. from v to w). To compute $L^{in}_{\leq}(\cdot)$, in HP-SPC, Dijkstra's algorithm rather than BFS is conducted from each vertex in the forward direction. After that, Dijkstra's algorithm is performed again in the reverse direction to compute $L^{out}_{\leq}(\cdot)$. A query (s, t) is evaluated by scanning $L^{out}_{\leq}(s)$ and $L^{in}_{\leq}(t)$ similarly to Algorithm 2.

To apply the 1-shell reduction, we first compute the 1-shell of G as if G is undirected. Then, for each connected component cc (in the undirected sense) of the 1-shell, we construct a reachability oracle for $\mathsf{cc} \cup a(\mathsf{cc})$. For any vertices $u, v \in \mathsf{cc} \cup a(\mathsf{cc})$, let $\mathsf{reach}(u, v) = 1$ if v is reachable from u in G and $= 0$ otherwise. Consider a query (s, t). If s and t come from the same cc, the # of shortest paths from s to t, $\mathsf{spc}(s, t)$, is exactly $\mathsf{reach}(s, t)$; otherwise, $\mathsf{spc}(s, t) = \mathsf{reach}(s, \mathsf{shr}(s)) \cdot \mathsf{spc}(\mathsf{shr}(s), \mathsf{shr}(t)) \cdot \mathsf{reach}(\mathsf{shr}(t), t)$.

For weighted directed graphs, we redefine neighborhood equivalence as follows. Let $\mathsf{nbr}_{in}(v)$ and $\mathsf{nbr}_{out}(v)$ be the in-neighbors and out-neighbors of a vertex v, respectively. We say vertex u is neighborhood equivalent to another vertex v if (1) $(u, v) \in E$ if and only if $(v, u) \in E$; furthermore, if they are both present, $l((u, v)) = l((v, u))$; (2) $\mathsf{nbr}_{in}(u)\backslash\{v\} = \mathsf{nbr}_{in}(v)\backslash\{u\}$; (3) for $\forall w \in \mathsf{nbr}_{in}(u) \cap \mathsf{nbr}_{in}(v)$, $l((w, u)) = l((w, v))$; (4) $\mathsf{nbr}_{out}(u)\backslash\{v\} = \mathsf{nbr}_{out}(v)\backslash\{u\}$; and (5) for $\forall w \in \mathsf{nbr}_{out}(u) \cap \mathsf{nbr}_{out}(v)$, $l((u, w)) = l((v, w))$. With this new equivalence relation, G is processed as described in Section 4.2 except that $L^{in}_{\leq}(\cdot)$ and $L^{out}_{\leq}(\cdot)$ are computed instead.

It is trivial to extend the independent-set reduction to weighted directed graphs. We thus omit the details here.

8 LIMITATIONS OF THIS WORK

We discuss some limitations of this work in the following.

Maintenance under Dynamic Updates. Given a graph G, a labeling $L(\cdot)$ for G and an update ΔG to G, the problem of dynamic labeling under ΔG is to maintain $L(\cdot)$ such that the resulting labeling is the corresponding one for $G \oplus \Delta G$, where $G \oplus \Delta G$ is the graph obtained by applying ΔG to G. For canonical distance labeling, the existing dynamic algorithms all fail to achieve high efficiency while retaining the minimality of the labeling [7, 18, 45], and it is still an open problem regarding how to address this issue. Since our labeling $L_{\leq}(\cdot)$ contains the canonical labeling $L^{c}_{\leq}(\cdot)$ as a subset, the same challenge persists when designing a dynamic algorithm for $L_{\leq}(\cdot)$. Worse yet, the necessity to maintain the information about $\sigma_{v,w}$ can further complicate the design.

Indexing Time and Index Size. As shown in the experiments, for some graphs such as DB and FL, our algorithms require far more indexing time and index space than other graphs of similar size. Unfortunately, thus far it is still not clear what properties of these graphs lead to this inefficiency. Is there an efficiently computable ordering that effectively handles these graphs? Or is the huge resource consumption inevitable even with an optimal ordering under the current labeling scheme? Both of the questions are also open.

9 CONCLUSIONS

We study the problem of counting the # of shortest paths between two vertices s and t. We propose exact shortest path cover (ESPC) to cover all shortest paths exactly once. Based on ESPC, we devise an algorithm to compute a hub labeling. The labeling scheme is able to exploit some intrinsic dimensions of graphs. We also apply index reduction techniques to further reduce the resulting index. Our experimental study verifies the effectiveness and efficiency of our algorithms.

ACKNOWLEDGMENTS

This work is supported by the Research Grants Council of Hong Kong, China under No. 14203618 and No. 14202919.

REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pages 230–241, 2011.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pages 24–35, 2012.

[3] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA*, pages 782–793, 2010.

[4] T. Akiba, T. Hayashi, N. Nori, Y. Iwata, and Y. Yoshida. Efficient top-k shortest-path distance queries on large networks by pruned landmark labeling. In *AAAI*, pages 2–8, 2015.

[5] T. Akiba, Y. Iwata, K.-i. Kawarabayashi, and Y. Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *ALENEX*, pages 147–154, 2014.

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

[7] T. Akiba, Y. Iwata, and Y. Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *WWW*, pages 237–248, 2014.

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

[9] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[10] H. Bast, S. Funke, and D. Matijević. Ultrafast shortest-path queries via transit nodes. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, pages 175–192. AMS, 2009.

[11] R. Bauer and D. Delling. SHARC: Fast and robust unidirectional routing. In *ALENEX*, pages 13–26, 2008.

[12] I. Bezáková and A. Searns. On counting oracles for path problems. In *ISAAC*, pages 56:1–56:12, 2018.

[13] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, 2011.

[14] P. Boldi and S. Vigna. The webgraph framework I: Compression techniques. In *WWW*, pages 595–602, 2004.

[15] U. Brandes. A faster algorithm for betweenness centrality. *J. Math. Sociol.*, 25(2):163–177, 2001.

[16] J. Cheng, Y. Ke, S. Chu, and C. Cheng. Efficient processing of distance queries in large graphs: A vertex cover approach. In *SIGMOD*, pages 457–468, 2012.

[17] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SICOMP*, 32(5):1338–1355, 2003.

[18] G. D’angelo, M. D’emidio, and D. Frigioni. Fully dynamic 2-hop cover labeling. *JEA*, 24(1):1.6:1–1.6:36, 2019.

[19] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Robust distance queries on massive networks. In *ESA*, pages 321–333, 2014.

[20] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *SIGMOD*, pages 157–168, 2012.

[21] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SICOMP*, 33(4):892–922, 2004.

[22] A. W.-C. Fu, H. Wu, J. Cheng, and R. C.-W. Wong. IS-Label: An independent-set based labeling scheme for point-to-point distance querying. *PVLDB*, 6(6):457–468, 2013.

[23] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[24] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *SODA*, pages 156–165, 2005.

[25] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Shortest path algorithms with preprocessing. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, pages 93–140. AMS, 2009.

[26] R. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *ALENEX*, pages 100–111, 2004.

[27] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[28] M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, pages 41–72. AMS, 2009.

[29] M. Holzer, G. Prasinos, F. Schulz, D. Wagner, and C. Zaroliagis. Engineering planar separator algorithms. In *ESA*, pages 628–639, 2005.

[30] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using Turán’s theorem. In *WWW*, pages 441–449, 2017.

[31] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong. Exact top-k nearest keyword search in large networks. In *SIGMOD*, pages 393–404, 2015.

[32] 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. *PVLDB*, 7(12):1203–1214, 2014.

[33] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD*, pages 445–456, 2012.

[34] S. Knopp, P. Sanders, D. Schultes, F. Schulz, and D. Wagner. Computing many-to-many shortest paths using highway hierarchies. In *ALENEX*, pages 36–45, 2007.

[35] D. C. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag New York, 1992.

[36] J. Kunegis. KONECT – The Koblenz Network Collection. In *WWW*, pages 1343–1350, 2013.

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

[38] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin. Scaling distance labeling on small-world networks. In *SIGMOD*, pages 1060–1077, 2019.

[39] Y. Li, M. L. Yiu, N. M. Kou, et al. An experimental study on hub labeling based shortest path algorithms. *PVLDB*, 11(4):445–457, 2017.

[40] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAP*, 36(2):177–189, 1979.

[41] K. Mehlhorn. *Data structures and algorithms 2: graph algorithms and NP-completeness*, volume 2. Springer-Verlag Berlin Heidelberg, 1984.

[42] 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 *SIGMOD*, pages 709–724, 2018.

[43] A. Pinar, C. Seshadhri, and V. Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *WWW*, pages 1431–1440, 2017.

[44] R. Puzis, Y. Elovici, and S. Dolev. Fast algorithm for successive computation of group betweenness centrality. *Phys. Rev. E*, 76(5):056709, 2007.

[45] 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. *WWWJ*, 20(5):915–937, 2017.

[46] X. Ren and J. Wang. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *PVLDB*, 8(5):617–628, 2015.

[47] Y. Ren, A. Ay, and T. Kahveci. Shortest path counting in probabilistic biological networks. *BMC Bioinformatics*, 19(1):465, 2018.

[48] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *DMKD*, 30(2):438–475, 2016.

[49] B. Roberts and D. P. Kroese. Estimating the number of s-t paths in a graph. *JGAA*, 11(1):195–214, 2007.

[50] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *ESA*, pages 568–579, 2005.

[51] D. Schultes and P. Sanders. Dynamic highway-node routing. In *WEA*, pages 66–79, 2007.

[52] Y. Tao, S. Papadopoulos, C. Sheng, and K. Stefanidis. Nearest keyword search in XML documents. In *SIGMOD*, pages 589–600, 2011.

[53] L. G. Valiant. The complexity of enumeration and reliability problems. *SICOMP*, 8(3):410–421, 1979.

[54] 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 *CIKM*, pages 563–572, 2007.

[55] F. Wei. TEDI: Efficient shortest path query answering on graphs. In *SIGMOD*, pages 99–110, 2010.

[56] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, pages 1601–1606, 2013.

[57] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: Towards bridging theory and practice. In *SIGMOD*, pages 857–868, 2013.