Divide-and-Conquer: Scalable Shortest Path Counting on Large Road Networks

MUHAMMAD FARHAN, Australian National University, Australia
HENNING KOEHLER, Massey University, New Zealand
QING WANG, Australian National University, Australia

The shortest path counting problem is crucial for various applications in road networks, such as network robustness analysis, traffic flow distribution, and navigation optimization. Unlike traditional shortest path problems, it requires enumerating all possible shortest paths, making it computationally challenging, especially in dense urban networks with numerous equal-length paths. Existing methods, such as 2-hop labeling schemes, precompute shortest-path distances and counts for efficient queries but struggle to scale in large networks. In this work, we propose a novel divide-and-conquer approach based on recursive vertex bipartitioning to address this limitation. At its core, we establish a count reconstruction theorem that efficiently combines shortest subpath counts from smaller subgraphs to accurately reconstruct shortest path counts for the entire graph. This approach significantly reduces computational overhead and storage requirements. We also introduce a 2-hop count labeling scheme that integrates effectively with this divide-and-conquer framework. Experimental results show that our approach significantly outperforms state-of-the-art solutions, doubling query processing speed, reducing label construction time to one-fourth, and requiring only around 20% of labeling space.

CCS Concepts: • **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Divide and conquer**; • **Information systems** → *Data structures*.

Additional Key Words and Phrases: Shortest-path; counting; vertex partitioning; 2-hop labelling

ACM Reference Format:
Muhammad Farhan, Henning Koehler, and Qing Wang. 2025. Divide-and-Conquer: Scalable Shortest Path Counting on Large Road Networks. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 149 (June 2025), **??** pages. https://doi.org/10.1145/3725400

1 Introduction

Given two vertices in a road network, how many distinct shortest paths exist between them? This question underpins the *shortest path counting* problem, a challenge with far-reaching implications across various domains. Solving this problem offers insights into network robustness, routing efficiency, and the structural significance of nodes and edges. Below are some examples illustrating its critical role in real-world road networks.

- *Network robustness analysis:* The number of shortest paths between two locations is often a key factor in measures of road network robustness [15, 26, 36]. Regions with multiple shortest paths are less vulnerable to disruptions, as a single road failure (e.g., due to accidents or maintenance) has less impact due to the availability of alternative routes.

Authors' Contact Information: Muhammad Farhan, Australian National University, Canberra, Australia, muhammad.farhan@anu.edu.au; Henning Koehler, Massey University, Palmerston North, New Zealand, h.koehler@massey.ac.nz; Qing Wang, Australian National University, Canberra, Australia, qing.wang@anu.edu.au.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2025 Copyright held by the owner/author(s).
ACM 2836-6573/2025/6-ART149
https://doi.org/10.1145/3725400

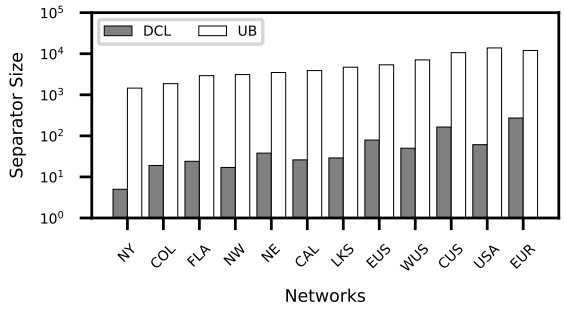


Fig. 1. Comparison of separator sizes produced by our method (DCL) on real-world road networks with the theoretical upper bound $UB = \sqrt{8n}$ as established by the planar separator theorem [9], where n represents the size of the network.

- *Traffic flow distribution:* Knowing how many shortest paths exist between key points in a road network allows for better traffic distribution [24]. Authorities can direct vehicles to use different shortest paths, avoiding congestion by spreading traffic more evenly across the available routes.
- *Navigation optimization:* For navigation systems, shortest path counting provides alternative routes of the same length, giving users multiple equally efficient options [21, 25]. This is particularly useful for reducing congestion, enhancing route flexibility, and improving overall travel efficiency.

However, the shortest path counting problem is both computationally and algorithmically more challenging than traditional shortest path problems. This is because shortest path counting requires enumerating and tracking *all possible shortest paths*, rather than simply identifying a single shortest path or the shortest path distance. The challenge is particularly pronounced in road networks, where intersections and parallel streets often create multiple routes of equal length. In dense urban areas, the number of such paths can increase rapidly, complicating the problem.

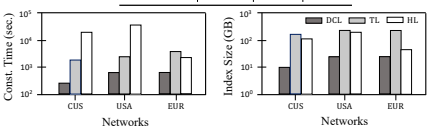
Dijkstra's algorithm can be adapted for shortest path counting, but it becomes inefficient for large road networks. To overcome this, several works have developed labeling techniques [32, 34, 40]. The main idea is to precompute a 2-hop label $L(v)$ for each vertex v, containing pairs $(u, \omega_{uv}, \delta_{uv})$, where u is a vertex, ω_{uv} is the distance between v and u, and δ_{uv} is the shortest-path count. To answer a shortest-path counting query between two vertices s and t, only their labels $L(s)$ and $L(t)$ are used, avoiding a full graph traversal. To guarantee accurate query results, the labeling must satisfy a cover constraint [40], which introduces computational overhead. Thus, Zhang and Yu [40] proposed the *Hub Labeling* (HL) method, which uses heuristic total ordering (e.g., based on vertex degree or significant paths) to prune unnecessary labels, similar to *Pruned Landmark Labeling* (PLL) [5]. Nevertheless, during a query, all vertices in the labels of s and t must still be considered to compute the shortest-path count. Qiu et al. [34] improve this by developing the *Tree Labeling* (TL) method, which constructs a vertex hierarchy using tree decomposition [10]. TL identifies a subset of common vertices in the labels of s and t, significantly reducing the search space for shortest-path counts. This method currently achieves state-of-the-art performance on road networks.

Challenges. Despite these advances, existing methods still struggle with scalability in large road networks, where the expanding network size and rapid increase in paths make shortest path

Methods	Index (Induced by a vertex order)		Query	Labelling Construction
HL [40]	2-Hop	Total order (*heuristic based*)	Full set	Subgraphs (*trough paths*)
TL [34]		Partial order (*tree decomposition*)	Subset	Subgraphs (*convex paths*)
DCL (ours)		Partial order (*divide-and-conquer*)	Subset	Subgraphs (*valley-chain paths*)

(a)

Methods	Query Time [μs]		
	CUSA	USA	EUR
HL	7.981	7.767	4.114
TL	2.613	2.685	3.195
DCL	1.045	1.029	1.507



(b)

Fig. 2. (a) High-level comparison of existing methods (HL and TL) and our method (DCL); (b) Empirical results on three real-world road networks (CUSA, USA, EUR) in terms of query time, construction time and index size.

counting computationally intensive. To address scalability issues, we leverage the power of divide-and-conquer by revisiting the planar separator theorem [27], given that road networks are nearly planar. However, we observe a significant discrepancy between the theoretical bound of $\sqrt{8n}$ for separators in a planar graph with n vertices and the sizes of possible separators in real-world road networks, as shown in Figure 1. This discrepancy prompts the question: can we devise a scalable solution for shortest path counting in large road networks using a practical, separator-based strategy? Such a solution would divide the problem into smaller subproblems, each focused on counting subpaths within different subgraphs, thereby enabling scalable computation. Nonetheless, previous research [9] suggests that combining subpath counts from subgraphs to reconstruct exact shortest path counts in the full graph is a complex challenge. This challenge unfolds in two critical dimensions:

(1) **Count reconstruction efficiency:** As separators disrupt graph connectivity, shortcuts are typically added within subgraphs to preserve distances and reconstruct counts [9]. However, this increases computational complexity and limits scalability.
(2) **Count computation efficiency:** Naively, we may apply Dijkstra's algorithm to count subpaths in each subgraph, but it would result in prohibitively high computational overhead, rendering the process impractical for large-scale networks.

Contributions. In this work, we aim to tackle the aforementioned challenges. The main contributions are summarized as follows:

Count Reconstruction Theorem. We introduce a scalable solution for shortest path counting using a divide-and-conquer method rooted in recursive vertex bipartitioning. Central to this solution is a count reconstruction theorem that combines shortest subpath counts from smaller subgraphs to reconstruct the shortest path counts in the full graph. This theorem guarantees that each shortest path is counted exactly once, preventing overcounting, undercounting, or miscounting along incorrect paths. Unlike prior work [9], it eliminates the need for shortcuts and reduces additional computation and storage for subpath counts, enabling an efficient algorithm design that scales effectively to large road networks.

2-Hop Count Labelling. We explore 2-hop labeling schemes for shortest path counting that mirror the divide-and-conquer nature of recursive vertex bipartitioning. We introduce a 2-hop set cover property, ensuring all shortest paths between any two vertices are captured within two hops. This property leverages a set of 2-hop pairs, each encoding a hub vertex and its corresponding subpaths to guarantee complete path coverage. To address the additional complexity of counting paths rather than just identifying them, we extend the 2-hop set cover property using our count reconstruction

theorem, resulting in a novel 2-hop count labeling scheme. Unlike previous work [40], our approach defines an explicit form of the 2-hop labeling scheme that captures exact shortest path counts.

Scalable Algorithm Design. We develop an efficient algorithm for constructing 2-hop count labels, grounded in the count reconstruction theorem within a divide-and-conquer framework. A key challenge is that recursive bipartitioning separators disrupt graph connectivity, complicating label computation. To overcome this, we observe that valley paths relative to separators are crucial, leading us to extending the concept of contraction hierarchies [19] to *shortcut count graphs*. This transforms a bi-directional upward search in the shortcut count graph into a simplified 2-hop label search, collapsing each valley-chain path into a single label entry for each direction, significantly improving count computation efficiency.

We conducted experiments on 12 real-world large road networks, including the entire road network of the USA [14] and the western Europe road network [3]. The results show that our method DCL consistently outperforms the state-of-the-art method, TL [34], achieving double the speed in query processing, four times faster construction time, and consuming around 20% of the labelling space across all networks. These improvements highlight the efficiency and scalability of our method DCL, making it a powerful solution for shortest path counting on real-world road networks. Figure 2 provides a high-level comparison between existing methods and our approach, highlighting key differences in algorithm design and how these differences are reflected in the empirical results.

2 Related Work

Below, we briefly discuss related work on both shortest-path distance and shortest-path counting.

2.1 Shortest Path Distance

The classical approach, Dijkstra's algorithm [37], traverses vertices in increasing distance from the source until reaching the target, but this is inefficient, especially for distant vertex pairs where the search space is large. A bidirectional scheme [33] runs Dijkstra's algorithm from both the source and destination, resulting in modest efficiency improvements. Many modern methods focus on precomputing an index structure based on the road network's topology. For example, several approaches [19, 31, 39, 42] leverage the hierarchical structure of road networks, introducing auxiliary edges, referred to as *shortcuts*, to guide the search towards relevant vertices for faster querying. *Contraction hierarchy* (CH) [19] is the most widely used method in this direction. Given a predefined vertex order τ, CH contracts each vertex v sequentially, adding shortcuts between its neighbors to preserve distance information through v.

Another important class of methods is hub-based labeling [1, 2, 4, 5, 11, 17, 23, 29, 41], which typically precomputes a 2-hop label for each vertex to capture distance information between all vertice pairs in the graph. At query time, only the distance labels are searched to compute distances. Until recently, the best performing approach has been *Hierarchical 2-Hop Index* (H2H-Index) [11, 29], which combines the advantages of hub labeling and hierarchical structures. However, a newer approach [17] called *Hierarchical Cut 2-Hop Labeling* has now shown to be the current state-of-the-art for answering distance queries on road networks.

2.2 Shortest Path Counting

A simple way to compute the number of shortest paths between two vertices is to run Dijkstra's search [37] from one vertex (e.g., the source) while tracking visited vertices by their distance from the source. For each visited vertex, the search maintains its shortest-path distance and count. When exploring a neighbor of the current vertex, if a shorter distance is found, the shortest-path distance is updated and the count is set to the number of paths to the neighbor. If the distance remains the

Table 1. Summary of Notations

Notation	Description
$G = (V, E, \omega)$	weighted graph
$d_G(v, u)$	distance between v and u in G
$L(v)$	label of a vertex v
$N_G(v)$	set of neighbors of vertex v in G
$P_G(v, u)$	shortest paths between v and u in G
$P_G(s, t; w)$	shortest paths between s and t through w in G
$P_G^d(s, t; w)$	shortest paths of length d between s and t through w in G
$\sigma_G(s, t)$	number of shortest paths between s and t in G
$\sigma_G^d(s, t)$	number of shortest paths of length d between s and t in G
$\zeta(s, t)$	number of shortest valley paths between s and t
$\preceq$	vertex partial order induced by partition ordering $\leq$
G_w	reduced graph w.r.t. vertex w
$\odot_w$	concatenation of two paths through w
$\oplus_w$	Cartesian concatenation of two sets of paths through w

same, the count is increased by adding the number of paths from the neighbor. No updates are made if the neighbor's distance is greater. Although effective, this approach does not scale well for large road networks, as it may require scanning all edges in the worst case.

To improve query response time, labeling-based methods [34, 40] were developed. These methods precompute an index L, called *hub labeling*, where each vertex v is assigned a label set $L(v)$ containing shortest distances and shortest path counts to selected hub vertices. The shortest path count between two vertices, e.g., s and t, can then be calculated in linear time by scanning $L(s)$ and $L(t)$. Zhang et al. [40] proposed a hub labeling scheme and a hub pushing algorithm to efficiently construct their hub labeling index, enabling real-time shortest path counting queries. During index construction, they assign a total order τ to all vertices and process them in descending order. For each vertex $v \in V$, a search is conducted from v within the induced subgraph of vertices whose orders are not higher than v, collecting the shortest-path distance and count to each vertex w. Later, Peng et al. [32] parallelized this approach to further enhance scalability for shortest-path counting queries on large graphs. Another recently proposed approach [34] utilizes the concept of tree decomposition [10] and introduces a tree-based labeling structure. This method allows searching through a subset of labels rather than the entire set in $L(s)$ and $L(t)$ at query time, leading to improved performance over [40]. We will discuss [40] and [34] in more detail in Section 4.

Several works in the literature address path counting problems for graphs other than road networks. Valiant [38] demonstrated that the s-t simple path counting problem is #P-complete. Bezakova et al. [9] presented a method for shortest path counting in planar graphs. Mihalák et al. [28] showed that counting all s-t paths up to a given length in a DAG is #P-complete. They also give a fully polynomial-time approximation scheme (FPTAS) to compute approximate distance and counting oracle in DAGs. Ren et al. [35] studied shortest path counting in probabilistic biological networks, while He et al. [20] proposed a data structure for categorical path counting in trees.

3 Preliminaries

Let $G = (V, E, \omega)$ be a weighted road network, where V is the vertex set, E is the edge set, and each edge $(u, v) \in E$ has a non-negative weight $\omega(u, v) \in \mathbb{R}$. We also use $V(G)$ and $E(G)$ to refer to the vertex set and edge set of G, respectively. We use $N_G(v)$ to denote the set of neighbors of a vertex v in G, i.e., $N_G(v) = \{u \mid (u, v) \in E\}$. A path is a sequence of distinct vertices $p = (v_1, v_2, \ldots, v_k)$ where $(v_i, v_{i+1}) \in E$ for each $1 \leq i \leq k-1$. The weight of a path p is defined as $\omega(p) = \sum_{i=1}^{k-1} \omega(v_i, v_{i+1})$. A

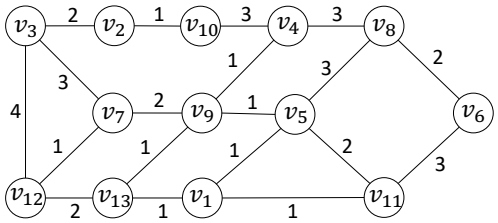


Fig. 3. An example road network G.

shortest path p between two vertices s and t is a path starting at s and ending at t such that $\omega(p)$ is no larger than any other path from s to t. The distance $d_G(s,t)$ is the weight of any shortest path between s and t in G, and $\sigma_G(s,t)$ refers to the number of such paths. We denote the set of all shortest paths between s and t as $P_G(s,t)$, and the subset of these paths that pass through w as $P_G(s,t;w)$. Table 1 summarizes the main notations.

Definition 3.1 (Vertex Cut). Given a road network G and two vertices $s,t \in V(G)$, a subset of vertices $C \subset V(G)$ is a *vertex cut* (or simply *cut*) that separates s and t in G iff every path between s and t passes through C. In particular, any set containing s or t is a vertex cut.

Definition 3.2 (Shortest Path Counting Problem). Given a road network $G = (V, E, \omega)$ and two vertices $\{s,t\} \in V(G)$, the *shortest path counting problem* is to efficiently compute $|P_G(s,t)|$ - the number of shortest paths between s and t in G.

Example 3.3. Consider the road network G in Figure 3. Between v_8 and v_{12}, there are multiple paths, such as $p_1 = \langle v_8, v_4, v_9, v_7, v_{12}\rangle$ with $\omega(p_1) = 7$ and $p_2 = \langle v_8, v_6, v_{11}, v_1, v_{13}, v_{12}\rangle$ with $\omega(p_2) = 9$. While p_2 is not a shortest path, p_1 is, with $d_G(v_8, v_{12}) = 7$. There are four other shortest paths, making $|P_G(v_8, v_{12})| = 5$.

4 Labelling-Based Solutions

We discuss two popular labeling-based solutions for shortest path counting: hub labeling index [40] and tree labeling index [34], with tree labeling index achieving state-of-the-art performance.

4.1 Hub Labelling

Zhang et al. [40] proposed a hub labelling (HL) method for shortest path counting. To ensure the correctness of labelling, Zhang et al. [40] further introduced the exact shortest path covering (ESPC) constraint. Let $T(v)$ denote a collection of entries of the form $(w, C_{v,w})$, where $C_{v,w} \subseteq P_G(v,w)$ represents a subset of the shortest paths from v to w. For any two vertices s and t, the ESPC constraint requires that all shortest paths between s and t are *exactly* covered by the two collections of entries $T(s)$ and $T(t)$. Based on this, a hub labelling L for shortest path counting assigns a label $L(v)$ to each vertex v in a road network G, which contains a set of label entries $\{(w_1, \omega_{vw_1}, \delta_{vw_1}), \ldots, (w_k, \omega_{vw_k}, \delta_{vu_k})\}$ where $\{w_1, \ldots, w_k\}$ are hub vertices of v, ω_{vw_i} refers to the shortest-path distance between v and w_i (i.e., $\omega_{vw_i} = d_G(v, w_i)$), and δ_{vw_i} is the number of a subset of shortest paths between v and w_i in G (i.e., $\delta_{vw_i} \leq |P_G(v, w_i)|$). Then, the shortest path count between s and t is computed by combining the entries in $L(s)$ and $L(t)$ such that

$$|P_G(s,t)| = \sum_{\substack{(w,\omega_{sw},\delta_{sw})\in L(s)\\(w,\omega_{tw},\delta_{tw})\in L(t)\\ \omega_{sw}+\omega_{tw}=d_G(s,t)}} \delta_{su} \times \delta_{tw},$$

Label	Entries
$L(v_1)$	$(v_1, 0, 1), (v_3, 7, 3), (v_5, 1, 1), (v_9, 2, 1)$
$L(v_2)$	$(v_2, 0, 1), (v_3, 2, 1), (v_4, 4, 1), (v_5, 6, 1), (v_9, 5, 1)$
$L(v_3)$	$(v_1, 7, 2), (v_3, 0, 1), (v_5, 6, 1), (v_9, 5, 1)$
$L(v_4)$	$(v_3, 6, 1), (v_4, 0, 1), (v_5, 2, 1), (v_9, 1, 1)$
$L(v_5)$	$(v_5, 0, 1)$
$L(v_6)$	$(v_1, 4, 1), (v_4, 5, 1), (v_5, 5, 2), (v_6, 0, 1), (v_8, 2, 1), (v_9, 6, 2), (v_{11}, 3, 1)$
$L(v_7)$	$(v_1, 4, 2), (v_3, 3, 1), (v_5, 3, 1), (v_7, 0, 1), (v_9, 2, 1)$
$L(v_8)$	$(v_3, 9, 1), (v_4, 3, 1), (v_5, 3, 1), (v_8, 0, 1), (v_9, 4, 2)$
$L(v_9)$	$(v_5, 1, 1), (v_9, 0, 1)$
$L(v_{10})$	$(v_2, 1, 1), (v_3, 3, 1), (v_4, 3, 1), (v_5, 5, 1), (v_9, 4, 1), (v_{10}, 0, 1)$
$L(v_{11})$	$(v_1, 1, 1), (v_5, 2, 2), (v_8, 5, 2), (v_9, 3, 1), (v_{11}, 0, 1)$
$L(v_{12})$	$(v_1, 3, 1), (v_3, 4, 2), (v_5, 4, 3), (v_7, 1, 1), (v_9, 3, 2), (v_{12}, 0, 1)$
$L(v_{13})$	$(v_1, 1, 1), (v_3, 6, 1), (v_5, 2, 2), (v_9, 1, 1), (v_{12}, 2, 1), (v_{13}, 0, 1)$

Fig. 4. An illustration of the hub labelling (HL) on the example road network G shown in Figure 3.

where $d_G(s,t)$ is computed based on the 2-hop cover property [12]:

$$d_G(s,t) = \min\{\omega_{sw} + \omega_{tw} \mid (w, \omega_{sw}, \delta_{sw}) \in L(s), (w, \omega_{tw}, \delta_{tw}) \in L(t)\}.$$

A hub-pushing algorithm was introduced to efficiently construct the hub labeling. The key idea is to assign a total order $\leq$ to all vertices. The algorithm performs a breadth-first search from each vertex w in descending order of $\leq$ to identify vertices that include w as a hub in their labels. A path is a *trough shortest path* if it is a shortest path and one of its endpoints has a higher rank than all the remaining vertices [22, 40]. Each search from w computes the shortest-path distance and count from w to other vertices v in the induced subgraph of vertices with ranks not higher than w. The labelling ensures the ESPC constraint by adding an entry $(v, \omega_{vw}, \sigma_{vw})$ to $L(v)$, where σ_{vw} is the count of all trough shortest paths from v to w for $w \leq v$, provided $\sigma_{vw} > 0$.

Example 4.1. Figure 4 shows the hub labeling for the road network depicted in Figure 3. The labels of any two vertices contain the distance to the highest-ranked vertex on their shortest paths. Consider the following total vertex ordering: $v_5 \leq v_9 \leq v_1 \leq v_3 \leq v_4 \leq v_7 \leq v_8 \leq v_{11} \leq v_{12} \leq v_{13} \leq v_2 \leq v_6 \leq v_{10}$, and two vertices v_8 and v_{12}, we have the shortest-path distance $d_G(v_8, v_{12}) = \min\{\omega(v_8, v_3) + \omega(v_{12}, v_3), \omega(v_8, v_5) + \omega(v_{12}, v_5), \omega(v_8, v_9) + \omega(v_{12}, v_9)\} = 7$, and the shortest-path count $|P_G(v_8, v_{12})| = \delta(v_8, v_5) \times \delta(v_{12}, v_5) + \delta(v_8, v_9) \times \delta(v_{12}, v_9) = 3 + 2 = 5$.

4.2 Tree Labelling

Observing that real road networks often have a low average degree and small treewidth [29, 30], Qiu et al. [34] proposed a tree labelling (TL) method for shortest path counting. The main idea behind TL is to derive a tree decomposition T_G from a road network G, and then construct a 2-hop labeling by storing, for each vertex v, the *convex* shortest-path distances and *convex* shortest-path counts to vertices with larger depth in the tree. A path $(v_1, v_2, \ldots, v_k)$ is *convex* if $v_2, \ldots, v_{k-1}$ appear in T_G with larger depth than one of the endpoints. Specifically, the label $L(v)$ for each vertex $v \in V(G)$ consists of: (1) a *distance array* $[\omega_{vw_1}, \ldots, \omega_{vw_k}]$ where ω_{vw_i} is the length of the convex shortest path between v and w_i; (2) a *count array* $[\delta_{vw_1}, \ldots, \delta_{vw_k}]$ where δ_{vw_i} is the convex shortest-path count between v and w_i; and (3) an *ancestor array* $[w_1, \ldots, w_k]$ storing the ancestors of v in T_G, separating vertex pairs for which v is the lowest common ancestor.

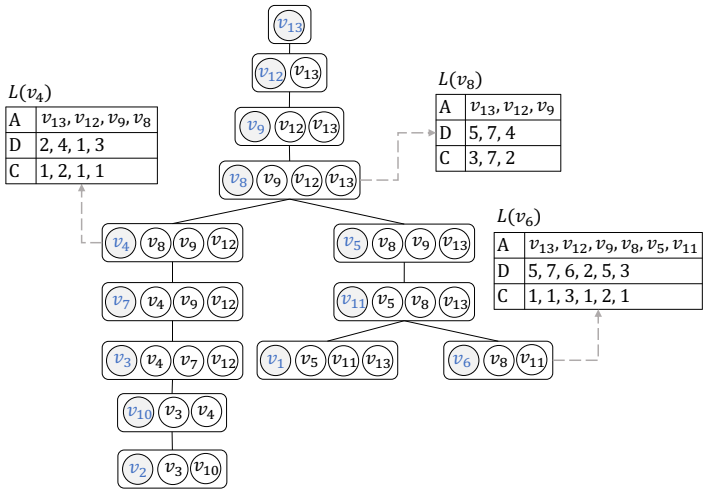


Fig. 5. An illustration of the tree labelling (TL) on the example road network G shown in Figure 3.

Let $L(v).D$ and $L(v).C$ represent the distance and count arrays in $L(v)$. The shortest-path count between two vertices s and t is computed as follows:

$$|P_G(s,t)| = \sum_{\substack{i \in CA(s,t) \\ d_G(s,t)=L(s).D(i)+L(t).D(i)}} L(s).C(i) \times L(t).C(i), \tag{1}$$

where $CA(s,t)$ denotes the set of vertices that are common ancestor of s and t, and $d_G(s,t) = \min\{L(s).D(i) + L(t).D(i) \mid i \in CA(s,t)\}$.

Example 4.2. Figure 5 illustrates the tree labeling for the road network in Figure 3. For a query between vertices v_4 and v_6, the label $L(v_4)$ includes: an ancestor array $A = [v_{13}, v_{12}, v_9, v_8]$ listing all ancestors of v_4 in T_G, a distance array $D = [2, 4, 1, 3]$ for the distances, and a count array $C = [1, 2, 1, 1]$ for the convex shortest-path counts from v_4 to its ancestors in T_G, with $L(v_6)$ structured similarly. We first obtain $CA(v_4, v_6) = \{v_{13}, v_{12}, v_9, v_8\}$, and then compute the distance $d_G(v_4, v_6) = \min\{L(v_4).D(i) + L(v_6).D(i) | i \in CA(v_4, v_6)\} = 5$, and the shortest-path count

$$|P_G(v_4, v_6)| = L(v_4).C(v_8) \times L(v_6).C(v_8) = 1.$$

4.3 Discussion

The hub labelling approach [40] has several limitations. First, to accelerate querying, it requires a pre-defined total ordering $\leq$ over all vertices. This total order is critical for label construction and may significantly affect the number of labels to be processed during query time. However, finding a vertex ordering that can minimize the size of hub labelling is known to be NP-hard [5, 7, 12]. Second, because labels are built based on this total order, where each vertex is processed in descending order, and the highest-ranked vertex is included in all other labels, the resulting label size is often large. Finally, the method searches through all label entries in $L(s)$ and $L(t)$ for a query pair (s,t). Consequently, the hub labelling solution suffers scalability issues due to the large size of labels.

The tree labeling approach [34] processes a subset of labels $L(s)$ and $L(t)$ for a query pair (s,t) using a hierarchy based on tree decomposition. However, finding a tree decomposition with minimal tree width is NP-complete [6]. TL-Index uses a sub-optimal algorithm [10] that computes a tree decomposition in $O(|V| \cdot (w^2 + \log(|V|)))$, but it can produce trees with large width w and

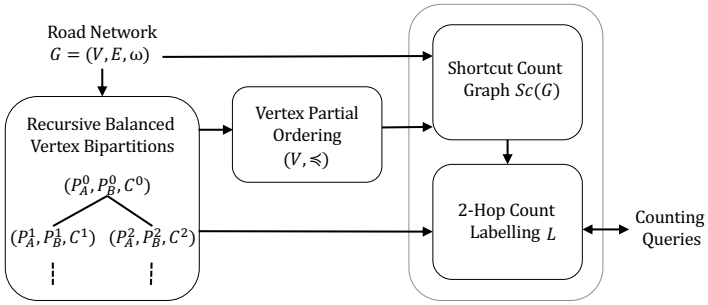


Fig. 6. High-level workflow of our divide-and-conquer approach discussed in Sections 5 and 6.

height h. Larger widths and heights increase label sizes and slow down queries as more hub vertices must be considered. TL-Index also employs the Range Minimum Query (RMQ) algorithm [8] to compute the lowest common ancestor (LCA) in $O(1)$ time, but the hidden constant in RMQ can significantly impact query speed.

To address the limitations of these existing solutions, we introduce a divide-and-conquer approach in the next section.

5 Divide-and-Conquer Approach

In this section, we first present recursive vertex partitioning, which achieves balanced cuts by dividing a road network into two components recursively. We then present a novel 2-hop count labeling framework and efficient algorithms for label construction. A high-level workflow is provided in Figure 6.

5.1 Recursive Vertex Bipartitioning

To solve the shortest path counting problem using a "divide-and-conquer" approach, we first introduce the concept of *balanced vertex bipartition* [27]. This concept allows us to recursively break down the shortest path counting problem into smaller sub-problems.

Definition 5.1 (Balanced Vertex Bipartition). Let $\beta \in (0, 0.5)$. A *balanced vertex bipartition* of a graph G is a triple (P_A, P_B, C) where

- P_A, P_B, and C form a partition of $V(G)$,
- no edge connects a vertex in P_A with a vertex in P_B, and
- $|P_i| \leq (1 - \beta) \cdot |V(G)|$ for $i \in \{A, B\}$.

Here, C is the *separator* of P_A and P_B in the bipartition (P_A, P_B, C), and β controls the balance between the partitions P_A and P_B.

By recursively applying balanced vertex bipartitioning, a graph G is divided into smaller partitions of similar sizes. The balance condition limits the recursion depth to $O(\log |V(G)|)$. Consequently, shortest paths in G are split into subpaths across partitions. A key challenge arises as the global structure of G becomes fragmented: *how to correctly recombine subpath counts across different partitions to reconstruct the shortest path count in G?* The following example illustrates scenarios where recombination can lead to overcounting or undercounting.

Example 5.2. Consider the pair (v_3, v_{11}) in Figure 7. To compute the shortest path count between vertices v_3 and v_{11}, counts for subpaths via v_9 and v_{13} (i.e., v_3–v_9, v_9–v_{11} and v_3–v_{13}, v_{13}–v_{11}) must be combined from the first-level partition shown in green. After combining these subpaths, the counts are 3 and 3, yielding a total of 6, which is overcounted because the path through edge (v_9, v_{13}) is

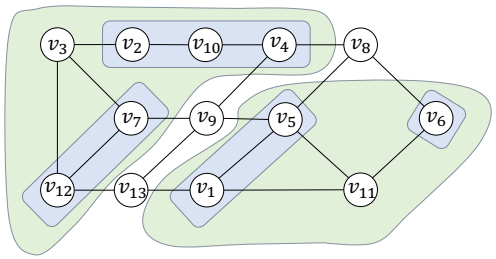


Fig. 7. Recursive vertex bipartition: first-level partitioning (green) with a separator $\{v_8, v_9, v_{13}\}$, and second-level partitioning (blue) with separators $\{v_3\}$ and $\{v_{11}\}$.

counted twice. While overcounting can be avoided by considering only one vertex in the separator during recombination, this risks undercounting shortest paths that pass through both vertices, such as those involving v_9 and v_{13}.

To ensure exact counting of shortest paths – where each shortest path between any two vertices $\{s, t\} \subset V(G)$ is counted *exactly once* and no non-shortest paths are mistakenly included – we rely on two simple but crucial observations. First, the separator C in each partition (P_A, P_B, C) acts as a vertex cut for vertices $s \in P_A \cup C$ and $t \in P_B \cup C$, meaning every path between s and t passes through C. Second, by counting on subgraphs built through step-by-step elimination of cut vertices from G, we can combine the subpath counts from these subgraphs to derive the exact shortest path counts in the original graph.

Below, we first introduce the related notions and then formalize the observations in Theorems 5.10 and 5.11.

Definition 5.3 (Partition Ordering). Let Γ be a set of vertex bipartitions recursively applied to a graph G, where each bipartition divides a subgraph induced by a previous partition. Γ induces a *partition ordering* $\leq$ on $V(G)$, such that for any bipartition $(P_A, P_B, C) \in \Gamma$, we have $w \leq v$ for all $w \in C$ and $v \in P_A \cup P_B$.

This partition ordering assigns lower ranks to vertices in the separator C compared to those in P_A and P_B, while the relative order between vertices in P_A and P_B is unimportant. To combine the counts of shortest subpaths from different partitions, we must further impose a total order within each separator C. An ordering $\preceq$ on $V(G)$ is a *vertex partial ordering* if it extends the partition ordering $\leq$ by imposing a total order on the vertices within each separator of G. This vertex partial ordering enables a sequence of reduced graphs during divide-and-conquer computations, ensuring accurate shortest path counts by progressively reducing the graph while preserving essential structural properties.

Example 5.4. Consider Figure 7, where the separators at the first, second, and third levels are $\{v_8, v_9, v_{13}\}$, $\{\{v_3\}, \{v_{11}\}\}$, and $\{\{v_{10}\}, \{v_1\}, \{v_{12}\}\}$, respectively. By ordering the vertices within each separator according to their vertex indices, we derive the vertex partial order $\preceq$ as $\{v_8 \leq v_9 \leq v_{13} \leq \ldots, v_3 \leq v_{10} \leq v_2, v_{10} \leq v_4, v_3 \leq v_{12} \leq v_7, v_{11} \leq v_6, v_{11} \leq v_1 \leq v_5\}$.

Definition 5.5 (Reduced Graph). Let $\preceq$ be a vertex partial ordering. For a vertex $w \in V(G)$, the *reduced graph* G_w with respect to w is the induced subgraph of G containing only the vertices ranked no lower than w under $\preceq$, i.e., $G_w = G[\{v \in V(G) \mid w \preceq v\}]$.

Counting shortest paths within subgraphs comes with the caveat that the path lengths may not match the distances in the full graph. In such cases, these subgraph paths are not shortest paths in G, meaning we cannot blindly sum their counts.

Example 5.6. Consider the pair (v_4, v_{12}) in Figure 7. After the first-level partition with the separator $\{v_8, v_9, v_{13}\}$, the shortest paths between v_4 and v_{12} within the partition have a length of 10. However, in the original graph G, their actual shortest paths are two paths of length 4.

To establish the connection between shortest path counts in the original graph and those within subgraphs, we define the notion of d-length shortest-path count.

Definition 5.7 (d-Length Shortest-Path Count). The *d-length shortest-path count*, denoted by $\sigma_G^d(s,t)$, is the number of shortest paths of length d between vertices s and t in the graph G. We denote by $P_G^d(s,t)$ the set of shortest paths of length d between s and t in G.

Note that $\sigma_G^d(s,t) = \sigma_G(s,t)$ if and only if $d = d_G(s,t)$; otherwise, $\sigma_G^d(s,t) = 0$. The following theorem, referred to as the *Count Reconstruction Theorem*, is crucial for solving the shortest path counting problem within a divide-and-conquer framework.

To begin, we establish some auxiliary results.

Proposition 5.8. *The common ancestors $W_{st} = \{w \mid w \preceq s, t\}$ of vertices s and t form a vertex cut separating s and t.*

Proof. It proceeds by induction on the construction of $\preceq$. □

Lemma 5.9. *Every path p contains a unique vertex w with $w \preceq v$ for all $v \in V(p)$. We call w the* minimal *vertex of p.*

Proof. Let $w \in V(p)$ be a vertex such that no strictly smaller vertex exists in p, i.e., $v \not\prec w$ for all $v \in V(p)$. Assume, for contradiction, that $w \not\preceq v$ for some $v \in V(p)$. By Proposition 5.8, the common ancestors of w and v form a vertex separator, so p must contain a common ancestor w'. This implies $w' \preceq w, v$, and thus $w' = w$, meaning $w \preceq v$. Uniqueness follows from the anti-symmetry of $\preceq$. □

We denote by $P_G^d(s,t;w)$ the subset of shortest paths in $P_G^d(s,t)$ that pass through w, or the empty set if $d \neq d_G(s,t)$.

Theorem 5.10 (Count Reconstruction Theorem). *Let $\{s,t\} \subset V(G)$ with $d = d_G(s,t)$ and $W_{st} = \{w \mid w \preceq s, t\}$. Then*

$$\sigma_G(s,t) = \sigma_G^d(s,t) = \sum_{w \in W_{st}} |P_{G_w}^d(s,t;w)|.$$

Proof. Since paths in $P_{G_w}^d(s,t;w)$ pass through w, and $w \preceq v$ for all $v \in G_w$, the set $P_{G_w}^d(s,t;w)$ contains exactly those shortest path of length d between s and t with w as their minimal vertex. By Lemma 5.9, every shortest path of length d between s and t has a unique minimal vertex w, with $w \in W_{st}$ by definition. Thus, each path is counted exactly once. □

As Theorem 5.10 only addresses shortest path count reconstruction for vertices separated by a vertex cut, we now generalize it to apply to any two vertices in the graph after recursive balanced bipartitioning. This enables shortest path counts to be reconstructed by combining subpath counts from a set of reduced graphs corresponding to earlier separators.

Theorem 5.11 (Shortest Path Counting Theorem). *Let $\{s,t\} \subset V(G)$ with $d = d_G(s,t)$ and $W_{st} = \{w \mid w \preceq s, t\}$. Define $\delta_{ab} = \sigma_{G_b}(a,b)$. For a balanced vertex bipartition (P_A, P_B, C) of G, the shortest path count is given by*

$$\sigma_G(s,t) = \sum_{P \in \{P_A, P_B\}} \sigma_{G[P]}^d(s,t) + \sum_{\substack{w \in C \cap W_{st} \\ d_{G_w}(s,w) + d_{G_w}(t,w) = d}} \delta_{sw} \cdot \delta_{tw}.$$

Proof. Since C separates P_A and P_B, a path lies entirely in P_A or P_B if and only if it does not intersect C. Thus, the first term counts s,t-paths of length d that do not intersect C.

It remains to show that the second term counts s,t-paths of length d intersecting C. Since $C \preceq P_A, P_B$ (i.e., $c \preceq v$ for $c \in C, v \in P_A \cup P_B$), the minimal vertex w in any such path p must lie in C, and thus in $C \cap W_{st}$. By the same argument as for Theorem 5.10, the number of these paths can be calculated as $\sum_{w \in W_{st} \cap C} |P^d_{G_w}(s,t;w)|$. Consider any such $P^d_{G_w}(s,t;w)$. If $d_{G_w}(s,w) + d_{G_w}(t,w) \neq d$, then no s,t-paths of length d passing through w exist. Otherwise, they are composed of shortest s,w-paths and shortest w,t-paths within G_w, and can thus be counted as $\delta_{sw} \cdot \delta_{tw}$. □

The above equation combines shortest path counts *within partitions* and *through separators*, ensuring accurate reconstruction of path counts in the original graph. There are two scenarios: (1) when the vertices s and t are in the same partition, we have $C \subseteq W_{st}$, allowing us to omit the intersection with W_{st}; (2) when the vertices s and t are in different partitions (or in the vertex separator), all shortest paths between s and t must pass through the vertex separator. Thus, the recursive components $\sigma^d_{G[P]}(s,t)$ for $P \in \{P_A, P_B\}$, which count shortest paths *not* passing through C, can be omitted. Together, these results lead to the following corollary.

Corollary 5.12. *If $\{s,t\} \subset P$ and $P \in \{P_A, P_B\}$, then*

$$\sigma_G(s,t) = \sigma^d_{G[P]}(s,t) + \sum_{\substack{w \in C \\ d_{G_w}(s,w)+d_{G_w}(t,w)=d}} \delta_{sw} \cdot \delta_{tw}. \tag{2}$$

Otherwise, if s and t lie in different partitions or in C, then

$$\sigma_G(s,t) = \sum_{\substack{w \in C \cap W_{st} \\ d_{G_w}(s,w)+d_{G_w}(t,w)=d}} \delta_{sw} \cdot \delta_{tw}. \tag{3}$$

Corollary 5.12 enables recursive computation of shortest path counts for any two vertices in a graph. The index size and computation time for $\sigma_G(s,t)$ and $d_{G_w}(\cdot,w)$ are proportional to the size of separators, making small separators crucial. Thus, we need to address the problem of finding small balanced separators.

The problem of finding the minimum balanced vertex separator is NP-hard in general [18]. However, the planar separator theorem ensures that for planar graphs, balanced vertex separators of size $O(\sqrt{V(G)})$ exist and can be found efficiently.

Theorem 5.13 ([27]). *If G is a planar graph and $\beta \leq 1/3$, there exists a vertex bipartition (P_A, P_B, C) of G such that $|C| \leq 2^{1.5}\sqrt{|V(G)|}$.*

As road networks are almost but not exactly planar, the algorithms associated with the planar separator theorem cannot be readily applied. However, the theoretical bounds established can still serve as guidelines for what should be achievable. We adopt a recursive bipartitioning approach from [13, 17]. The main idea is to select distant vertices s and t, ensure balance by contracting regions around s and t into two sets of vertices S and T, and then find a minimal ST-cut that separates S and T. Our experiments show that this produces small balanced separators, of sizes well below the upper bounds established in Theorem 5.13 (see Figure 1).

Remark 1. Previous work [9] explored recursively bipartitioning road networks using a divide-and-conquer approach. However, our method improves upon [9] in several ways. Let G'_w denote a subgraph of G induced by $(V(G) \setminus C) \cup \{w\}$ for $w \in C$, i.e., removing all vertices in the separator C except w. In [9], the authors add shortcuts after partitioning to bypass paths through separators, ensuring that distances between vertices are preserved, i.e., $d_G(s,t) = d_{G'_w}(s,t)$ for all $s,t \in V(G)$. They then store both $\sigma_G(w,v)$ and $\sigma_{G'_w}(w,v)$ for each $v \in V(G'_w)$, resulting in $O(|C||V(G)|) =$

$O(|V(G)|^{1.5})$ for label construction and space, and $O(\sqrt{|V(G)|})$ for query time. In contrast, our approach avoids the need for adding shortcuts for each vertex in the separator, eliminates the necessity to compute distances in two different graphs, and does not require storing separate distances and counts for subgraphs and the entire graph. As a result, our count reconstruction theorem enables the design of highly scalable and efficient algorithms, significantly reducing computational overhead and storage requirements compared to previous methods.

5.2 2-Hop Count Labelling

In the following, we discuss how the divide-and-conquer nature of recursive vertex bipartitioning can be exploited to establish a 2-hop labeling scheme for solving the shortest path counting problem.

We begin by formulating a cover property that ensures all shortest paths between any two vertices can be covered within 2 hops.

5.2.1 2-Hop Set Cover.

The concept of a *hop* is defined below.

Definition 5.14 (Hop). Given a graph G, a *hop* is a pair (w, P_{v*}), where $w \in V(G)$ and P_{w*} is a set of paths from w to another vertex $*$. The vertex w is called the *hop vertex*.

The following cover property states that for any two vertices s and t in a graph, there exists a combination of paths from a set of hops that can reconstruct *all shortest paths* between s and t.

Definition 5.15 (2-Hop Set Cover). A set of hops H on G is a *2-hop set cover* of all shortest paths in G if, for any two vertices $\{s,t\} \subset V(G)$ with $d_G(s,t) \in \mathbb{R}_{>0}$, there exists a set of 2-hop pairs

$$\{\langle (w_1, P_{w_1 s}), (w_1, P_{w_1 t})\rangle, \ldots, \langle (w_k, P_{w_k s}), (w_k, P_{w_k t})\rangle\}$$

where $\{(w_i, P_{w_i s}), (w_i, P_{w_i t})\} \subseteq H$ for $i \in [1,k]$, such that

$$\bigcup_{i=1}^{k} P_{w_i s} \oplus_{w_i} P_{w_i t} = P_G(s,t).$$

Here, $\oplus_w$ represents the *Cartesian concatenation* of all paths from two sets through their common hop vertex w. Formally, given two sets of paths P_1 and P_2, where all paths have w as a common endpoint, the operation $P_1 \oplus_w P_2$ is defined as

$$P_1 \oplus_w P_2 = \{p_1 \odot_w p_2 \mid p_1 \in P_1, p_2 \in P_2\},$$

where $p_1 \odot_w p_2$ concatenates p_1 and p_2 at the common vertex w, treating w as a single vertex in the resulting path. Such a vertex w is called a *join vertex*. Hence, $P_{ws} \oplus_w P_{wt}$ produces the set of shortest paths between s and t that pass through the vertex w.

Remark 2. While the 2-hop set cover property ensures that all shortest paths are covered within 2 hops, it does not guarantee exact counts. Solving the shortest path counting problem requires each path to be counted *exactly once*. Thus, this property is necessary but insufficient for a complete labeling. Here, a *complete* labeling means that every shortest path is covered, and the total count between any two vertices is accurate, without redundancy or omission.

5.2.2 2-Hop Count Labelling.

We define a 2-hop labeling scheme for shortest path counting by extending the 2-hop set cover property. The key idea, based on Theorem 5.10, is to utilize the partial order $\preceq$ on $V(G)$. This enables the construction of a sequence of reduced graphs $(G_{w_1}, G_{w_2}, \ldots, G_{w_k})$ with respect to $\preceq$, where $w_i \preceq w_{i+1}$ for $i \in [1, k-1]$. To reconstruct the shortest path count between s and t, we define a 2-hop pair as $\langle (w_i, \sigma^{d_s}_{G_{w_i}}(w_i, s)), (w_i, \sigma^{d_t}_{G_{w_i}}(w_i, t))\rangle$, where $d_s = d_G(w_i, s)$ and $d_t = d_G(w_i, t)$. One crucial aspect is that, instead of computing the shortest path count between a vertex v and a hop vertex w_i in the original graph G, we compute their counts in the reduced

Label	Distance	Count	Separator
$L(v_1)$	$[4, 2, 1], [1], [0]$	$[1, 2, 1], [1], [1]$	10
$L(v_2)$	$[7, 5, 8], [2], [1], [0]$	$[1, 1, 2], [1], [1], [1]$	000
$L(v_3)$	$[9, 5, 6], [0]$	$[3, 1, 2], [1]$	0
$L(v_4)$	$[3, 1, 12], [6], [3], [0]$	$[1, 1, 2], [1], [1], [1]$	001
$L(v_5)$	$[3, 1, 2], [2], [1], [0]$	$[1, 1, 1], [2], [1], [1]$	100
$L(v_6)$	$[2, 6, 5], [3], [0]$	$[1, 3, 1], [1], [1]$	11
$L(v_7)$	$[6, 2, 3], [3], [1], [0]$	$[2, 1, 1], [1], [1], [1]$	010
$L(v_8)$	$[0]$	$[1]$	λ
$L(v_9)$	$[4, 0]$	$[2, 1]$	λ
$L(v_{10})$	$[6, 4, 9], [3], [0]$	$[1, 1, 2], [1], [1]$	00
$L(v_{11})$	$[5, 3, 2], [0]$	$[3, 3, 1], [1]$	1
$L(v_{12})$	$[7, 3, 2], [4], [0]$	$[5, 2, 1], [2], [1]$	01
$L(v_{13})$	$[5, 1, 0]$	$[3, 1, 1]$	λ

Fig. 8. An illustration of 2-hop count labelling L.

graph, i.e., $\sigma_{G_{w_i}}(w_i, v)$. By Theorem 5.10, if $\{w_1, w_2, \ldots, w_k\}$ forms a vertex cut separating s and t in G, the shortest path count between s and t can be accurately reconstructed using the counts in the smaller partitions of G.

Each separator is assigned a unique bitstring, defined recursively as follows: The separator at the first partitioning level is assigned the empty bitstring λ. For the separator at the i-th partitioning level, the bitstrings are $s0$ and $s1$, where s represents the bitstring of the separator from the previous $(i-1)$-th level.

Based on these considerations, the following labeling is designed for shortest path counting.

Definition 5.16 (2-Hop Count Labeling). Let $\mathcal{C}(v) = \{C_1, \ldots, C_k\}$ denote the sequence of all separators whose vertices have lower ranks than v in the partial order $\preceq$. A *2-hop count labeling* is a set of labels $L = \{L(v) \mid v \in V(G)\}$, where each label $L(v)$ consists of:

(1) A separator identifier $\text{sep}(v)$, which is the bitstring associated with the separator containing vetex v;
(2) A sequence of distance arrays $(\text{dis}(C_1), \ldots, \text{dis}(C_k))$, where each $\text{dis}(C)$ for $C \in \mathcal{C}(v)$ is an array of distances:
$$[d_{G_{w_1}}(v, w_1), \ldots, d_{G_{w_{|C|}}}(v, w_{|C|})];$$
(3) A sequence of count arrays $(\text{cnt}(C_1), \ldots, \text{cnt}(C_k))$, where each $\text{cnt}(C)$ for $C \in \mathcal{C}(v)$ is an array of path counts:
$$[\sigma_{G_{w_1}}(v, w_1), \ldots, \sigma_{G_{w_{|C|}}}(v, w_{|C|})].$$

These label entries are stored consecutively in memory, leading to efficient caching and improving query performance.

Example 5.17. Consider Figure 8. Each vertex label $L(v)$ stores the shortest path distances and counts to vertices with lower ranks. For example, the label $L(v_{10})$ stores the shortest path distances and counts from vertex v_{10} to all vertices in its separator set $\mathcal{C}(v_{10}) = \{\{v_8, v_9, v_{13}\}, \{v_3\}, \{v_{10}\}\}$. Specifically, for v_{13} in the separator $\{v_8, v_9, v_{13}\}$, the shortest path distance from v_{10} to v_{13} is 9, with a corresponding shortest path count of 2.

5.2.3 Query Processing. For any two vertices $\{s, t\} \subset V(G)$, the distance and the count of shortest paths between s and t in G can be computed as follows.

Theorem 5.18. *Let $\{s,t\} \subset V(G)$ and $C^* = \bigcup_{C \in \mathcal{C}(s) \cap \mathcal{C}(t)} C$. Then, the distance between s and t in G is given by*

$$d_G(s,t) = \min\left(d_{G_w}(s,w) + d_{G_w}(t,w) \mid w \in C^*\right). \tag{4}$$

The shortest path count between s and t in G is given by

$$\sigma_G(s,t) = \sum_{\substack{w \in C^* \\ d_G(s,t) = d_{G_w}(s,w) + d_{G_w}(t,w)}} \sigma_{G_w}(s,w) \cdot \sigma_{G_w}(t,w). \tag{5}$$

Proof. For the distance formula (Eq. (4)), each term $d_{G_w}(s,w) + d_{G_w}(t,w)$ represents the length of an s-to-t path in G, ensuring $d_G(s,t) \leq \min(\ldots)$. Conversely, let p be a shortest path from s to t. By Lemma 5.9, p passes through a minimal vertex $w \in C^*$ and lies entirely in G_w, giving $d_G(s,t) = d_{G_w}(s,w) + d_{G_w}(t,w)$. For the count formula (Eq. (5)), by Theorem 5.11, the number of shortest paths through $w \in C^*$ is $\sigma_{G_w}(s,w) \cdot \sigma_{G_w}(t,w)$, provided $d_G(s,t) = d_{G_w}(s,w) + d_{G_w}(t,w)$. Summing over all such w gives $\sigma_G(s,t)$, completing the proof. □

Example 5.19. Consider a query pair (v_5, v_6) in G depicted in Figure 3. To compute the common separator vertices C^*, we use the bitstrings of v_5 and v_6, which are 100 and 11, respectively, as shown in Figure 8. The common prefix of the bitstrings 100 and 11 indicates that v_5 and v_6 are separated at partitioning level $\ell = 1$. Consequently, $C^* = \{v_8, v_9, v_{11}, v_{13}\}$, comprising all separator vertices at levels $0 \leq i \leq \ell$. Using the distance and count arrays in $L(v_5)$ and $L(v_6)$, we compute the shortest path distance $d_G(v_5, v_6) = 5$ using Eq. (4), and the shortest path count $\sigma_G(v_5, v_6) = 3$ using Eq. (5).

6 Labelling Construction

In this section, we introduce the *shortcut-based method*, which uses balanced vertex bipartitioning and shortcut count graph to accelerate label computation significantly. The core idea of this shortcut-based method is to compute labels recursively by reusing the label information of vertex u when computing the label for vertex v, where $u \preceq v$. This method enables the reconstruction of the shortest path counts without repeatedly processing the reduced graph, making the approach scalable for large graphs. A key technique in this method is the concept of a valley path [39].

Definition 6.1 (Valley Path). Given a road network G and a vertex partial order $\preceq$, a path p between vertices v and w with $w \preceq v$ is called a *valley path* iff $v \preceq u$ for every vertex $u \in V(p) \setminus \{v, w\}$.

Every edge in G is a trivial valley path. A *shortest valley path* between two vertices s and t is a valley path whose length is shortest among all valley paths between s and t. Note that a shortest valley path between two vertices is not necessarily a shortest path because shorter non-valley paths may exist. Based on valley paths, we define the notion of *shortcut count graph*.

Definition 6.2 (Shortcut Count Graph). A *Shortcut Count Graph* (Sc-Graph) over a road network G and a vertex partial order $\preceq$ is defined as $\mathrm{Sc}(G) = (V', E', \omega, \zeta)$, where

- the vertex set $V' = V(G)$ is the same as in G and the edge set E' consists of all edges of G and shortcut edges that correspond to all non-trivial valley paths in G, i.e., each $(v, w) \in E' \setminus E(G)$ represents a valley path between v and w;
- $\omega : V(G) \times V(G) \to \mathbb{R}_{\geq 0}$ assigns a distance to each edge, where $\omega(v, w)$ is the distance between v and w;
- $\zeta : V(G) \times V(G) \to \mathbb{N}$ assigns a count to each edge, where $\zeta(v, w)$ is the count of shortest valley paths between v and w.

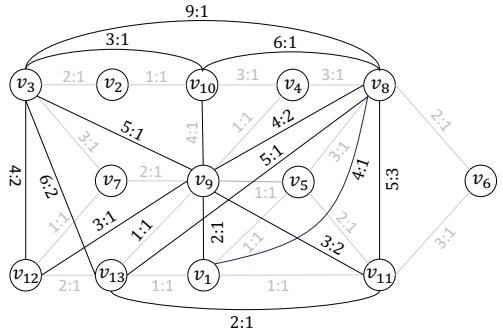


Fig. 9. An illustration of the shortcut count graph $\mathrm{Sc}(G)$ for the road network G depicted in Figure 3.

For each vertex $v \in V(G)$, let $up(v) = \{u \mid (u,v) \in E' \wedge u \preceq v\}$ denote its upward neighbors, and $down(v) = \{u \mid (u,v) \in E' \wedge v \preceq u\}$ denote its downward neighbors in $\mathrm{Sc}(G)$.

Definition 6.3 (Valley-Chain Path). A path p from s to t in a shortcut count graph $\mathrm{Sc}(G)$ is a *valley-chain path* if $p = p_1 \odot_{w_1} \cdots \odot_{w_{k-1}} p_k$, where each p_i is a valley path and one of the following holds:

- $\bigwedge_{1 \le i < j \le k-1} s \preceq w_i \preceq w_j \preceq t$;
- $\bigwedge_{1 \le i < j \le k-1} s \succeq w_i \succeq w_j \succeq t$.

Join vertices $\{w_1, \ldots, w_{k-1}\}$ in a valley-chain path are either ascending or descending in the partial order $\preceq$. The following lemma links shortest paths in G to their valley-chain paths in $\mathrm{Sc}(G)$.

Lemma 6.4. *Any shortest path p between vertices s and t in a road network G is either a valley-chain path or can be uniquely decomposed into two valley-chain paths, p_1 and p_2, such that $p = p_1 \odot_w p_2$. In the latter case, $w \preceq v$ for all $v \in V(p)$.*

Proof. Let w be the minimum vertex in p (see Lemma 5.9). If w is an endpoint of p, then p is a valley-chain path. Otherwise, we decompose p into $p_1 \odot_w p_2$, making p_1 and p_2 valley-chain paths by our preceding claim. Since the minimum vertex of a valley-chain path must be an endpoint, this decomposition is unique. It remains to show that p is a valley-chain path if w is an endpoint, and that decompositions of valley-chain paths into valley paths are unique. For this, denote by s the other (not w) endpoint of p, and by u the vertex in p closest to s with $v \prec s$. Then the s,u-prefix of p is a valley path, and by Definition 6.1 it is the only prefix of p forming an ascending valley path. Existence and uniqueness of p's decomposition into an ascending chain of valley paths follow by induction on its length. □

We note that the existence of such a decomposition was previously observed in [39, Observation 1]. Here, we also establish its uniqueness, which is crucial for counting shortest paths during label construction.

Based on Lemma 6.4, a bi-directional upward search in the shortcut count graph is simplified to a 2-hop label search, where a valley-chain path is reduced to a single label entry in each direction.

Example 6.5. Figure 9 shows the shortcut count graph over a road network G, as depicted in Figure 7(a). Solid black edges represent shortcut edges corresponding to non-trivial valley paths. For instance, consider the shortcut edge (v_9, v_{11}), where two valley paths $\langle v_9, v_5, v_{11}\rangle$ and $\langle v_9, v_5, v_1, v_{11}\rangle$ exist between v_9 and v_{11}. The count $\zeta(v_9, v_{11}) = 2$ and the distance $\omega(v_9, v_{11}) = 3$ are represented as $[3 : 2]$ next to the shortcut (v_9, v_{11}). Note that $\langle v_9, v_5, v_8, v_6, v_{11}\rangle$ is not a valley path since $v_8 \preceq v_9$. Now

consider the path $p = \langle v_4, v_{10}, v_3 \rangle$, which is a valley-chain path because it is formed by concatenating two valley paths, $p_1 = \langle v_4, v_{10} \rangle$ and $p_2 = \langle v_{10}, v_2, v_3 \rangle$, i.e., $p = p_1 \odot_{v_{10}} p_2$. Nonetheless, the path $\langle v_5, v_9, v_7 \rangle$ is not a valley-chain path as $v_9 \preceq v_5$.

To efficiently construct 2-hop count labels, we draw key observations about the upward neighbors in the shortcut count graph.

Lemma 6.6. *The following hold:*
(1) For $u_i, u_j \in up(v)$, either $u_i \preceq u_j$ or $u_j \preceq u_i$.
(2) For $w \preceq v$, $up(v)$ is a vertex cut separating v and w.
(3) For $u \in up(v)$, $\zeta(v,u) = \sigma_{G_u \setminus \{u' \in up(v) | u \prec u'\}}(v,u)$.

Proof. (1) Both u_i and u_j are ancestors of v, and all ancestors of a vertex are comparable by the construction of $\preceq$. (2) The proof follows directly from Lemma 6.4. (3) Every path from v to u in G_u is a valley-chain path. By Lemma 6.4, such a path is a valley path iff it contains no upward neighbor of v other than u. □

Statement (1) asserts that all vertices in $up(v)$ are totally ordered. Statement (2) states that $up(v)$ forms a vertex cut in the reduced graph G_w, separating v from w, where w is ranked no higher than any vertex in $up(v)$. Statement (3) says that the number of shortest valley paths between v and any vertex u in $up(v)$ equals the number of shortest paths in the reduced graph G_u which do not pass other upward neighbours.

We derive the following theorem, which is crucial for efficiently computing 2-hop count labels in a divide-and-conquer manner.

Theorem 6.7. *For any two vertices $\{v, w\} \subset V(G)$ with $w \preceq v$, we have the following:*

$$\sigma_{G_w}(v,w) = \sum_{\substack{u \in up(v), w \preceq u \\ d_{G_w}(v,w) = d_{G_w}(v,u) + d_{G_w}(w,u)}} \zeta(v,u) \cdot \sigma_{G_w}(u,w).$$

Proof. By Lemma 6.4, every shortest path p in G_w from v to w is a valley-chain path, and its decomposition $p = p_1 \odot_{w_1} \cdots \odot_{w_{k-1}} p_k$ into valley paths is unique. We can partition all (shortest) paths by their first valley path p_1, the endpoint of which must lie in $up(v)$, and add up the shortest path counts of all partitions. For the partition where p_1 ends in u, the shortest path count (provided any shortest path from v to w passes through u) is simply $\zeta(v,u)$, the number of valley paths corresponding to p_1, times $\sigma_{G_w}(u,w)$, the number of shortest paths from u to w (as each such path is a valley chain). □

This theorem enables the recursive computation of each label via valley-chain paths. Specifically, $\sigma_{G_w}(v,w)$ is computed using $\zeta(v,u)$ in $\mathrm{Sc}(G)$ and $\sigma_{G_w}(u,w)$ for $u \in up(v)$. Similarly, $\sigma_{G_w}(u,w)$ depends on $\zeta(u,u')$ and $\sigma_{G_w}(u',w)$ for $u' \in up(u)$, proceeding recursively until reaching w. The divide-and-conquer decomposition of each valley-chain path into a valley path and the remaining chain is unique. Thus, Theorem 6.7 ensures that every shortest path in $P_{G_w}(v,w)$ is counted exactly once, ensuring complete labeling for shortest path counts based on the 2-hop set cover property.

6.1 Proposed Algorithm

To efficiently construct our labeling, we use a two-step approach. First, we build a *shortcut count graph (Sc-Graph)* that preserves distances and shortest valley path counts by adding shortcut edges to bypass valley paths. Second, we construct labels directly from the Sc-Graph.

We adopt the approach from [17] to construct recursive vertex bipartitioning, without considering the distance-preserving property for each partition. For each separator, we compute a partition bitstring and a depth value τ. The partition bitstrings enable constant-time identification of the

Algorithm 1: Shortcut Count Graph Construction

```
1  Function Sc-Graph(G, ≤)
2      initialize shortcut count graph G' ← G
       // compute shortcuts
3      foreach v ∈ V in decreasing order of ≤ do
4          foreach (u, ω), (u', ω') ∈ N_{G'}(v) with u, u' ≤ v do
5              if ⟨u, u'⟩ ∉ E(G') then
6                  E(G') ← E(G') ∪ {(u, u')}
7                  ω(u, u') ← ω + ω'
8                  ζ(u, u') ← ζ(u, v) × ζ(u', v)
9              else
10                 if ω(u, u') > ω + ω' then
11                     ω(u, u') ← ω + ω'
12                     ζ(u, u') ← ζ(u, v) × ζ(u', v)
13                 else if ω(u, u') = ω + ω' then
14                     ζ(u, u') ← ζ(u, u') + ζ(u, v) × ζ(u', v)
15     return G'
```

Algorithm 2: Shortcut-Based Labelling Construction

```
1  Function DCL(G', ≤)
       // initialize
2      foreach v ∈ V do
3          D_v ← [∞, ..., ∞], C_v ← [0, ..., 0]
4          D_v[v] ← 0, C_v[v] = 1
       // compute label distances
5      foreach v ∈ V in increasing order of ≤ do
6          foreach u ∈ up(v) do
7              foreach w s.t. w ≤ u do
8                  ω+ ← ω(v, u) + D_u[w]
9                  ζ+ ← ζ(v, u) × C_u[w]
10                 if D_v[w] > ω+ then
11                     D_v[w] ← ω+
12                     C_v[w] ← ζ+
13                 else if D_v[w] = ω+ then
14                     C_v[w] ← C_v[w] + ζ+
```

lowest common ancestor for two nodes s and t, and the depth of this ancestor is then used to determine which parts of $L(s)$ and $L(t)$ are needed for computing their distance.

Sc-Graph Construction. Algorithm 1 shows the details. Given a road network G and a partial-vertex order $\preceq$, for every edge $(v, w) \in E(G)$, the count value is initialized to 1 (Line 2). For each

vertex $v \in V(G)$ in the decreasing order of $\preceq$, we apply the contraction operation [31] to v as follows. For every pair of neighbors $(w, \omega), (w', \omega')$ of v in G, if the edge (w, w') does not exist, we create the edge and assign its weight and count values as $\omega + \omega'$ and $\zeta(w, v) \times \zeta(w', v)$, respectively (Lines 5-8). If there already exists an edge (w, w'), we check if the new weight of (w, w') is shorter than the old weight. We replace the old values of $\omega(w, w')$ and $\zeta(w, w')$ with $\omega + \omega'$ and $\zeta(v, u) \times \zeta(u, w)$, respectively (Lines 11-12). If the new and old weights of the edge (v, w) are the same, we increase the corresponding count value of the edge $\zeta(w, w')$ to $\zeta(w, w') + \zeta(w, v) \times \zeta(w', v)$ (Line 14).

Lemma 6.8. *Algorithm 1 returns a Sc-Graph of G.*

Labelling Construction. We present a bottom-up approach to efficiently construct the 2-hop count labeling L, described in Algorithm 2. Given the shortcut count graph G' and the partial-vertex order $\preceq$, for each $v \in V$, we maintain two lists, $D[*]$ and $C[*]$, and initialize their distances and counts to self as 0 and 1, respectively (Lines 2-4). We compute the label $L(v)$ for each vertex v in increasing order w.r.t. $\preceq$ (Lines 5-14). For each $u \in up(v)$, we inspect all vertices w whose ranks are lower than or equal to u. If a shorter path between v and w is found (i.e., $D_v[w] > \omega_+$), we update $D_v[w]$ and $C_v[w]$ with ω_+ and ζ_+, respectively. Otherwise, if $D_v[w] = \omega_+$, we increase the shortest path count between v and w by adding ζ_+ to $C_v[w]$.

Example 6.9. Consider Figures 7 and 9. Suppose the labels for all vertices with ranks lower than 6, namely $\{v_8, v_9, v_{13}, v_{11}\}$, have already been computed. We now calculate the label $L(v_6)$ for vertex v_6. From Figure 9, we know $up(v_6) = \{v_8, v_{11}\}$. First, we consider $v_8 \in up(v_6)$ and the reduced graph G_{v_8}. The shortest path distance and count between v_6 and v_8 are $d_{G_{v_8}}(v_6, v_8) = 2$ and $\sigma_{G_{v_8}}(v_6, v_8) = 1$, respectively. For v_8, the distance and count to itself are $d_{G_{v_8}}(v_8, v_8) = 0$ and $\sigma_{G_{v_8}}(v_8, v_8) = 1$, already initialized as $D_{v_8}[v_8]$ and $C_{v_8}[v_8]$ in $L(v_8)$. Therefore, $d_{G_{v_8}}(v_6, v_8)$ remains as 2, calculated as $d_{G_{v_8}}(v_6, v_8) + d_{G_{v_8}}(v_8, v_8) = 2$. Since $D_{v_6}[v_8] = \infty > 2$, we update $D_{v_6}[v_8] = 2$ and $C_{v_6}[v_8] = 1$ (Lines 11-12, Algorithm 2).

Next, consider $v_{11} \in up(v_6)$. Distances and counts to vertices $\{v_8, v_9, v_{13}, v_{11}\}$ have already been computed since their ranks are lower than v_{11}. For example, in the reduced graph G_{v_8} ($v_8 \preceq v_{11}$), the shortest path distance and count between v_6 and v_{11} are $d_{G_{v_8}}(v_6, v_{11}) = 3$ and $\sigma_{G_{v_8}}(v_6, v_{11}) = 1$. Similarly, the distance and count between v_{11} and v_8 in G_{v_8} are $d_{G_{v_8}}(v_{11}, v_8) = 5$ and $\sigma_{G_{v_8}}(v_{11}, v_8) = 3$, already stored as $D_{v_{11}}[v_8]$ and $C_{v_{11}}[v_8]$ in $L(v_{11})$ (see Figure 8). Since $D_{v_6}[v_8] = 2$ is smaller than $d_{G_{v_8}}(v_6, v_{11}) + d_{G_{v_8}}(v_{11}, v_8) = 8$, no update is made to $D_{v_6}[v_8]$ or $C_{v_6}[v_8]$. Similarly, distances and counts from v_6 to $\{v_9, v_{13}, v_{11}\}$ are computed.

6.2 Parallel Construction

We can parallelize Algorithm 2 to enhance performance by computing vertex labels at each partitioning level i concurrently using multi-threading. At each partitioning level i, threads are assigned to process vertices v independently, computing D_v and C_v, since these distance labels are exclusive to each thread. Specifically, we can parallelize Line 6 to process vertices in parallel in the increasing order of levels i. However, the parallelization is limited to the number of available physical threads, as a significant portion of the work (over 10%) remains sequential or partially parallelized, primarily due to the computations required for top-level labels.

6.3 Complexity Analysis

Table 2 compares the time and space complexity of state-of-the-art methods with our proposed method DCL for constructing labels and querying shortest-path counts. For each vertex v, we recursively compute and store distances and counts to all vertices w where $w \preceq v$, using $up(v)$. Let c represent the maximum size of the set $|\{w \mid w \preceq v\}|$, and d' the maximum size of $up(v)$. The

Table 2. Complexity analysis and comparison.

Method	Labelling Construction		Query Time
	Time	Space	
CO [9]	$O(\|V(G)\|^{1.5})$	$O(\|V(G)\|^{1.5})$	$O(\sqrt{\|V(G)\|})$
HL [40]	$O(\|V(G)\| \cdot l \cdot (l+d))$	$O(\sum_{v \in V(G)} \|L(v)\|)$	$O(l)$
TL [34]	$O(\|V(G)\| \cdot h \cdot w)$	$O(\|V(G)\| \cdot h)$	$O(h)$
DCL	$O(\|V(G)\| \cdot c \cdot d')$	$O(\|V(G)\| \cdot c)$	$O(c)$

time and space complexity of Algorithm 2 to construct 2-hop count labels are $O(|V(G)| \cdot c \cdot d')$ and $O(|V(G)| \cdot c)$, respectively. As c can in principle become as large as $|V(G)|$, due to large cuts, the Counting Oracle (CO) [9] offers better theoretical complexity guarantees in terms of $|V(G)|$, though only for strictly planar graphs. We note that for the road-networks examined, c falls well below the $\sqrt{8 \cdot |V(G)|}$ bound for cut size that the Counting Oracle is based on, as show in Figure 1. HL [40] has time complexity $O(|V(G)| \cdot l \cdot (l+d))$, where l is the maximum label size and d is the maximum degree. While their l value is roughly comparable to our c, the worst-case analysis for query answering obscures the fact that HL must traverse (almost) all entries of $L(s)$ and $L(t)$, while TL usually only needs to inspect a small subset. The complexity of TL [34] is similar to DCL; however, their values for h and w (representing height and width of their tree decomposition) tend to be significantly larger than ours which we compare in Table 4 for all the networks examined in this paper.

7 Experiments

We conducted experiments on a Linux server Intel Xeon W-2175 with 2.50GHz CPU, 28 cores, and 512GB of main memory. We implemented the proposed method in C++20 and compiled it with the GNU C++ compiler 11.4.0 using optimization level 3.

Datasets. We used twelve graph instances, among them eleven are USA based road networks containing a road network of whole USA that has about 24 million vertices and 58 million edges and they can be obtained from the 9th DIMACS Implementation Challenge [14]. One is Western Europe obtained from PTV AG [3] with about 18 million vertices and 42 million edges. We used travel times as the weights of edges and treated all of them as undirected graphs.

Table 3. Summary of datasets.

Network	Region	$\|V\|$	$\|E\|$	*diam.*
NY	New York City	264,346	733,846	720
COL	Colorado	435,666	1,057,066	1,245
FLA	Florida	1,070,376	2,712,798	2,058
NW	Northwest US	1,207,945	2,840,208	1,994
NE	Northeast US	1,524,453	3,897,636	2,098
CAL	California	1,890,815	4,657,742	2,315
LKS	Great Lakes	2,758,119	6,885,658	4,127
EUS	Eastern USA	3,598,623	8,778,114	4,461
WUS	Western USA	6,262,104	15,248,146	4,420
CUS	Central USA	14,081,816	34,292,496	5,533
USA	United States	23,947,347	58,333,344	8,440
EUR	Western Europe	18,010,173	42,560,279	3,175

Baselines. We compare our method DCL with the state-of-the-art method TL [34] for shortest path counting query processing on real-world large road networks. The code for TL was generously

provided by their authors and implemented in C++. We omit comparisons with HL [40], as TL has already been shown to outperform it, and with [9], as their separators are orders of magnitude larger, making their approach impractical, and no algorithm implementation is provided in the paper. We set the balance partition threshold to $\beta = 0.2$ to construct recursive vertex bipartitioning.

7.1 Performance Comparison

We compare the proposed method DCL with the state-of-the-art method TL with regard to query processing time, labelling construction time, and labelling size (space usage).

Table 4. Comparison of average query times, average hop sizes, labeling sizes, and construction times between our method (DCL) and the state-of-the-art method (TL). Here, d' and c represent the separator size and maximum label size for DCL, respectively, while w and h denote the tree width and height for TL.

Network	Avg. Query Time [μs]		Avg. Hop Size		Labelling Size		Const. Time [s]		DCL		TL	
	DCL	TL	DCL	TL	DCL	TL	DCL	TL	d'	c	w	h
NY	0.408	0.998	29	191	206 MB	818 MB	2	12	91	284	134	505
COL	0.477	1.082	38	205	271 MB	1.20 GB	4	11	111	387	146	465
FLA	0.547	1.312	33	207	657 MB	3.20 GB	10	32	75	277	136	520
NW	0.535	1.236	31	181	698 MB	3.80 GB	10	33	82	292	146	548
NE	0.647	1.323	53	239	1.63 GB	6.40 GB	21	79	174	462	219	828
CAL	0.655	1.532	51	322	1.58 GB	8.00 GB	25	79	156	482	215	713
LKS	0.776	1.801	72	479	4.24 MB	20.0 GB	52	218	292	653	370	1,325
EUS	0.796	1.625	79	362	4.45 GB	21.0 GB	64	189	158	561	272	1,022
WUS	0.815	1.743	80	412	7.34 GB	37.0 GB	108	332	200	326	646	1,041
CUS	1.045	2.613	130	916	29.8 GB	186 GB	426	1,832	349	1,067	660	2,433
USA	1.029	2.685	129	889	53.8 GB	322 GB	637	2,822	435	1,182	693	2,564
EUR	1.507	3.195	314	1,303	54.9 GB	321 GB	824	3,056	523	1,430	1,085	3,536

7.1.1 Query time. We first compare query time under two different settings: (1) queries with random distances, and (2) queries with varying distances.

Querying with Random Distance. For each dataset, we randomly sampled one million queries and report the average query time as shown in Table 4. DCL is significantly faster than TL across all datasets. This is mainly because DCL visits a significantly small number of hops compared with TL when counting shortest paths as shown in the column "Avg. Hop Size" in Table 4. For example, in the EUR dataset, DCL takes $1.5\mu s$ on average while TL requires $3.2\mu s$. Overall, DCL is more than 2 times faster than TL.

Querying with Varying Distance. Following [34], we also evaluate the query performance using query pairs with varying distances. We generate 10 sets of query pairs $Q_1, Q_2, \ldots, Q_{10}$ for each network. Let $x = (\frac{l_{max}}{l_{min}})^{1/10}$, where $l_{min} = 1,000$ meters and l_{max} is the maximum distance of any query pair. For $\forall 1 \leq i \leq 10$, we generate 10,000 queries in each set Q_i with distances in the range $(l_{min} \cdot x^{i-1},\ l_{min} \cdot x^i]$. We report the average query time for both DCL and TL over 10,000 queries in each set Q_1 to Q_{10} in Figure 10. We can see that our proposed method DCL outperforms TL in every query set, showing good performance for both short and long distance query pairs. We observe that when the query distance increases, the time cost of both methods decreases. This is because two distant vertices may have only a small number of common ancestors, as their lowest common ancestor typically lies at higher levels of the tree, which is characterized by small separators (or cuts). However, TL has a huge labelling sizes due to very large tree width w and height h which heavily deteriorate its performance.

7.1.2 Construction Time. Table 4 compares the labeling construction times of DCL and TL, with times measured using a single thread. DCL is significantly faster than TL across all networks, particularly on large datasets like CUS and USA, where it is over four times faster. Most of DCL's construction time is spent on bipartitioning, while label construction is extremely fast. Overall, DCL is more than three times faster than TL on all road networks. Parallelization can further reduce construction times, as shown in Figure 11 (and will be discussed further in Section 7.2).

7.1.3 Labelling Size. Table 4 reports the labelling size for DCL and TL. The labelling size of our DCL is significantly smaller than that of TL. Specifically, the labelling size of DCL is 4-6 times smaller than that of TL. This is primarily because TL is designed based on a tree decomposition with potentially very large tree width and height which can cause each vertex to store a large number of label entries. In contrary, DCL is constructed based on a tree hierarchy generated using a partitioning technique which produces small cuts to partition a graph. Our distance labels thus store a significantly smaller number of label entries than TL.

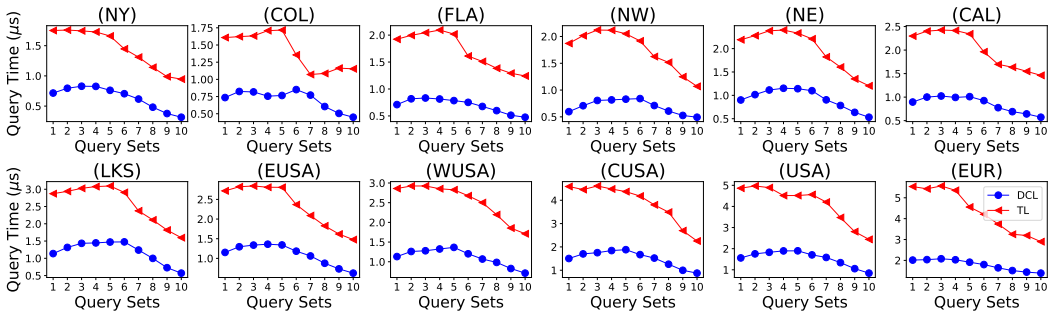


Fig. 10. Query performance using sets of query pairs with varying distances.

7.2 Performance Analysis

We analyse the performance of our proposed method DCL against TL based on the number of hops visited in query processing.

Visited Label Entries. When processing queries, both DCL and TL need to check and compare a number of label entries to compute the shortest-path counts. In Table 4 next to query times, we report the average number of visited label entries (#Hop) for processing 1 million random queries. We can see that the #Hop of DCL is significantly smaller than TL on all the datasets. For example, on the largest two datasets USA and WEUR, the #Hop of DCL is 129 and 314, and that of TL is 889 and 1,303, respectively. That means, on average, TL visits 889 and 1,202 label entries to answer the query, while DCL only visits 129 and 314 label entries. This result is consistent with our query performance evaluation in Tabel 4.

Height and Width. In Table 4, we also analyse the maximum label size c and maximum separator size d' of our method DCL against the tree width w and height h of the state-of-the-art method TL. We can clearly see that DCL has significantly smaller values of c and d' compared with h and w of TL for all datasets. This is because our method DCL finds balanced separators using [17] to partition a network which has shown to produce significantly small separators and labels compared to existing algorithms that are based on heuristic techniques for finding tree decomposition.

Scalability w.r.t. # of Cores. In Figure 11, we also test the performance of DCL on all the networks by varying the # of cores from 1 to 8. As we can see, DCL shows near linear performance improvement with respect to # of cores on all the networks. Particularly, the performance improvement is quite noticeable for the largest three datasets, namely CUS, USA and EUR as they are more likely to benefit from increased # of cores because of large amount of vertices at each level to process in parallel.

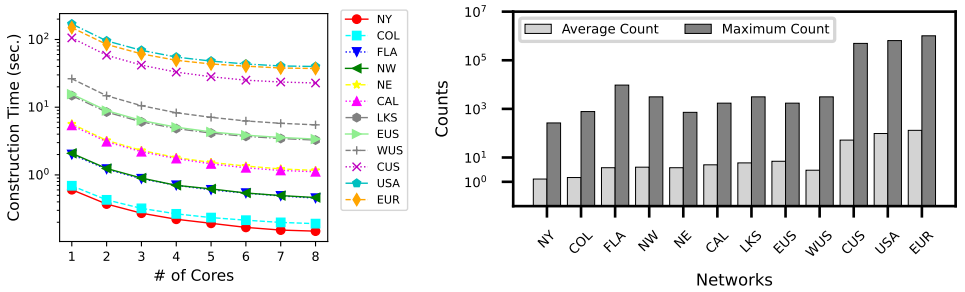


Fig. 11. (left) Construction performance with varying cores; (right) Shortest path counts on different networks.

7.3 Network Robustness Analysis

Following [16], we evaluate network robustness using the redundancy ratio (RR), which measures the average number of shortest paths between all vertex pairs in a network, with higher values indicating greater robustness. We also reported the maximum shortest path counts for all networks and present them in Figure 11. As shown in Figure 11, larger road networks tend to have higher shortest path counts, making them more robust. This is because larger road networks often include more central points, increasing the likelihood of multiple shortest paths with the same shortest distance. For small networks such as NY, BAY, and COL, the average shortest path counts are approximately 1.5. For medium-sized networks like FLA and WUS, the average shortest path counts range from 3 to 7. In contrast, large networks like USA and EUR exhibit significantly higher average shortest path counts of 97 and 131, respectively.

7.4 Extension to Directed Graphs

Our approach can be extended to directed graphs by storing shortest path distances and counts from both directions in each label. We compute them by performing searches in both directions when conducting label construction. However, road networks tend to be *almost* undirected with some famous exceptions, such as Stockholm, in which case the shortest path distances and counts stored for both directions within each label will frequently be identical. This raises intriguing possibility for future research on how to exploit this symmetry to reduce the size of labels in directed road networks.

8 Conclusion

Inspired by the planar separator theorem, this work proposes a divide-and-conquer solution for shortest path counting. The central idea is the *count reconstruction theorem*, which efficiently combines subpath counts from smaller subgraphs to reconstruct exact shortest path counts. Building on this, we propose a *2-hop count labeling* scheme and develop algorithms for efficient vertex label computation. Our proposed solution significantly enhances scalability, as demonstrated in our

experiments on 12 large road networks, where our method consistently outperforms state-of-the-art techniques in querying, labelling construction efficiency, and memory usage.

While our solution demonstrates remarkable efficiency in road network scenarios, it has limitations. For instance, it is not expected to perform well on complex networks, such as social networks, regardless of separator optimality. Indexing approaches for such networks differ significantly from those used for road networks.

Acknowledgments

This research was supported partially by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (project DP210102273).

References

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *Proceedings of the 10th International Conference on Experimental Algorithms*. 230–241.

[2] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. 2012. Hierarchical Hub Labelings for Shortest Paths. In *Proceedings of the 20th Annual European Conference on Algorithms*. 24–35.

[3] PTV AG. [n.d.]. Western europe dataset. http://www.ptv.de

[4] Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. 2014. Fast shortest-path distance queries on road networks by pruned highway labeling. In *2014 Proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX)*. 147–154.

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

[6] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. 1987. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods* 8, 2 (1987), 277–284.

[7] Maxim Babenko, Andrew V Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller. 2015. On the complexity of hub labeling. In *International Symposium on Mathematical Foundations of Computer Science: Mathematical Foundations of Computer Science (MFCS)*.

[8] Michael A. Bender and Martín Farach-Colton. 2000. The LCA Problem Revisited. In *LATIN 2000: Theoretical Informatics*, Gaston H. Gonnet and Alfredo Viola (Eds.). Springer Berlin Heidelberg, 88–94.

[9] Ivona Bezáková and Andrew Searns. 2018. On counting oracles for path problems. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

[10] Hans L Bodlaender. 2006. Treewidth: characterizations, applications, and computations. In *32nd International Workshop of Graph-Theoretic Concepts in Computer Science*. 1–14.

[11] Zitong Chen, Ada Wai-Chee Fu, Minhao Jiang, Eric Lo, and Pengfei Zhang. 2021. P2h: Efficient distance querying on road networks by projected vertex separators. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 313–325.

[12] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[13] Daniel Delling, Andrew V. Goldberg, Ilya P. Razenshteyn, and Renato Fonseca F. Werneck. 2011. Graph Partitioning with Natural Cuts. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011 - Conference Proceedings*. IEEE, 1135–1146.

[14] Camil Demetrescu, Andrew V Goldberg, and David S Johnson. 2009. *The shortest path problem: Ninth DIMACS implementation challenge*. Vol. 74. American Mathematical Soc.

[15] Yingying Duan and Feng Lu. 2014. Robustness of city road networks at different granularities. *Physica A: Statistical Mechanics and its Applications* 411 (2014), 21–34.

[16] Rawia Ahmed El-Rashidy and Susan Grant-Muller. 2016. The evaluation of redundancy for road traffic networks. *Transport* 31, 4 (2016), 427–439.

[17] Muhammad Farhan, Henning Koehler, Robert Ohms, and Qing Wang. 2024. Hierarchical Cut Labelling–Scaling Up Distance Queries on Road Networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

[18] Uriel Feige and Mohammad Mahdian. 2006. Finding small balanced separators. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, Jon M. Kleinberg (Ed.). 375–384.

[19] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International workshop on experimental and efficient algorithms*. 319–333.
[20] Meng He and Serikzhan Kazi. 2022. Data structures for categorical path counting queries. *Theoretical Computer Science* 938 (2022), 97–111.
[21] Bin Jiang and Christophe Claramunt. 2004. Topological analysis of urban street networks. *Environment and Planning B: Planning and design* 31, 1 (2004), 151–162.
[22] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *arXiv preprint arXiv:1403.0779* (2014).
[23] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor Lee. 2012. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 445–456.
[24] Alec Kirkley, Hugo Barbosa, Marc Barthelemy, and Gourab Ghoshal. 2018. From the betweenness centrality in street networks to structural invariants in random planar graphs. *Nature communications* 9, 1 (2018), 2501.
[25] Steven M LaValle. 2006. *Planning algorithms*. Cambridge university press.
[26] Minwei Li. 2008. Robustness Analysis for Road Networks. *Trail thesis series* (2008).
[27] Richard J Lipton and Robert Endre Tarjan. 1979. A separator theorem for planar graphs. *SIAM J. Appl. Math.* 36, 2 (1979), 177–189.
[28] Matúš Mihalák, Rastislav Šrámek, and Peter Widmayer. 2016. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory of Computing Systems* 58 (2016), 45–59.
[29] Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. 2018. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 709–724.
[30] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Progressive top-k nearest neighbors search in large road networks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1781–1795.
[31] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees. *Proceedings of the VLDB Endowment* 13, 5 (2020), 602–615.
[32] You Peng, Jeffrey Xu Yu, and Sibo Wang. 2023. PSPC: efficient parallel shortest path counting on large-scale graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 896–908.
[33] Ira Sheldon Pohl. 1969. *Bi-Directional and Heuristic Search in Path Problems*. Ph.D. Dissertation. Stanford, CA, USA. AAI7001588.
[34] Yu-Xuan Qiu, Dong Wen, Lu Qin, Wentao Li, Rong-Hua Li, Zhang Ying, et al. 2022. Efficient shortest path counting on large road networks. *Proceedings of the VLDB Endowment* (2022).
[35] Yuanfang Ren, Ahmet Ay, and Tamer Kahveci. 2018. Shortest path counting in probabilistic biological networks. *BMC bioinformatics* 19 (2018), 1–19.
[36] Wen-Long Shang, Yanyan Chen, Chengcheng Song, and Washington Y Ochieng. 2020. Robustness analysis of urban road networks from topological and operational perspectives. *Mathematical Problems in Engineering* 2020, 1 (2020), 5875803.
[37] Robert Endre Tarjan. 1983. *Data structures and network algorithms*. SIAM.
[38] Leslie G Valiant. 1979. The complexity of enumeration and reliability problems. *siam Journal on Computing* 8, 3 (1979), 410–421.
[39] Victor Junqiu Wei, Raymond Chi-Wing Wong, and Cheng Long. 2020. Architecture-intact oracle for fastest path and time queries on dynamic spatial networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1841–1856.
[40] 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*. 1813–1828.
[41] Yikai Zhang and Jeffrey Xu Yu. 2022. Relative Subboundedness of Contraction Hierarchy and Hierarchical 2-Hop Index in Dynamic Road Networks. In *Proceedings of the 2022 International Conference on Management of Data*. 1992–2005.
[42] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 857–868.

Received October 2024; revised January 2025; accepted February 2025