



Fully Dynamic Contraction Hierarchies with Label Restrictions on Road Networks

Zi Chen¹ · Bo Feng¹ · Long Yuan² · Xuemin Lin³ · Liping Wang¹

Received: 17 May 2023 / Revised: 3 August 2023 / Accepted: 9 August 2023 / Published online: 4 September 2023
© The Author(s) 2023

Abstract

In the real world, road networks with weight and label on edges can be applied in several application domains. The shortest path query with label restrictions has been receiving increasing attention recently. To efficiently answer such kind of queries, a novel index, namely Contraction Hierarchies with Label Restrictions (CHLR), is proposed in the literature. However, existing studies mainly focus on the static road networks and do not support the CHLR maintenance when the road networks are dynamically changed. Motivated by this, in this paper, we investigate the CHLR maintenance problem in dynamic road networks. We first devise a baseline approach to update CHLR by recomputing the potential affected shortcuts. However, many shortcuts recomputed in baseline do not change in fact, which leads to unnecessary overhead of the baseline. To overcome the drawbacks of baseline, we further propose a novel CHLR maintenance algorithm which can only travel little shortcuts through an update propagate chain with accuracy guarantee. Moreover, an optimization strategy is presented to further improve the efficiency of index maintenance. Considering the frequency of edge changes, we also propose a batch index maintenance algorithm to handle batch edge changes which can process a large number of edge changes at once. Furthermore, a parallel method is proposed to further accelerate calculations. Extensive and comprehensive experiments are conducted on real road networks. The experimental results demonstrate the efficiency and effectiveness of our proposed algorithms.

Keywords Contraction hierarchy with label restrictions · Dynamic road networks · Shortest path query

1 Introduction

Shortest path query (SPQ) in road networks is a fundamental problem in graph analysis and has been widely studied due to its applications in various fields. Recently,

with the proliferation of applications, the information of road networks becomes more complex and diversified, such as edge weight (e.g., transit time, toll fee) and edge label (e.g., toll road, trucks prohibited road). Existing works mainly aim to address the SPQ with minimum weight, i.e., find the shortest path with minimum distance or toll fee between two locations [1, 3, 6, 13, 16, 34]. However, the label information of edges can service more personalized scenarios. For example, if the drivers do not want to pay the toll fee, all toll roads should be neglected in the route planning [5]. Moreover, if a truck driver requires path service, the path returned should avoid all trucks prohibited roads [23]. The SPQ with considering the label constraints has many practical applications, such as personalized location-based services [35], logistics and commercial transportation [14]. [27] formalizes this problem as Label Constrained Shortest Path (LCSP) query. Formally, given a restricted label set R , LCSP query aims to find the shortest path with minimum weight avoiding the edges with labels in R . To efficiently answer the LCSP query, [27] proposes a novel index which is based on Contraction

✉ Long Yuan
longyuan@njust.edu.cn

Zi Chen
zchen@sei.ecnu.edu.cn

Bo Feng
bfeng@stu.ecnu.edu.cn

Xuemin Lin
xuemin.lin@sjtu.edu.cn

Liping Wang
lipingwang@sei.ecnu.edu.cn

¹ East China Normal University, Shanghai, China

² School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China

³ The Shanghai Jiao Tong University, Shanghai, China

Hierarchies, namely Contraction Hierarchies with Label Restrictions (CHLR). In CHLR, a *shortcut* $e = ((u, v), w, l)$ denotes a path from u to v with weight w and label l . By shortening the path traversal, CHLR can significantly improve the LCSP query processing performance.

1.1 Motivation

In the real world, road networks can be dynamically changed. For each edge, the toll fee can be changed at any time by traffic department and the transit time is time-dependent due to the change of traffic conditions [25, 37]. Specifically, the traffic conditions can be updated by collecting the travel information of taxi-hailing app users and drivers (e.g., Didi, Uber). However, the number of app users and drivers is huge. In 2022, Didi has 15 million drivers and 493 million users [12], and Uber has 3.5 million drivers and 93 million users [10]. They can generate massive updates for traffic conditions per second. Accordingly, the edge weight can be dynamically changed at a high frequency. Moreover, the label of edges can be changed as well. For example, roads can be labeled as light traffic road, medium traffic road and heavy traffic road depending on the traffic conditions which will change in real time [22]. Unfortunately, most existing works for CHLR focus on static road networks and do not support the CHLR maintenance in dynamic road networks, which makes them unpractical for real applications. Motivated by this, in this paper, we study the CHLR maintenance problem in dynamic road networks.

1.2 Our Approach

To solve the CHLR maintenance problem in dynamic road networks, we first propose a **Baseline** approach. When an edge e_o changes its weight and label, a naive approach is to find all potential affected shortcuts due to the change of e_o and then recompute the affected shortcuts by invoking the existing CHLR construction algorithm [27]. However, the **Baseline** would recompute lots of shortcuts that actually do not change, which leads to expensive overhead of **Baseline**. To overcome the drawbacks of **Baseline**, in this paper, we devise a more efficient CHLR maintenance algorithm. Specifically, we design a novel *update propagate chain* to avoid massive invalid updates in **Baseline**. In the update propagate chain, the *neighborhood of shortcuts* are defined as *parents*, *child* and *partner*. We observe that a shortcut only affects its child, and the weight of child can be obtained by its parents. We give an *update order* for the update propagate chain following the observation. Moreover, to further improve the efficiency, we propose an optimization strategy based on *weight count* of shortcuts, which

can guarantee that shortcuts will only be recomputed when necessary. We prove that the optimized algorithm has a tightly theoretical boundedness in terms of the number of changed shortcuts E_Δ , i.e., $O(|E_\Delta| \cdot \log|E_\Delta| + |E_\Delta| \cdot d_{\max})$, where $d_{\max}$ is the maximum degree.

In real applications, the travel platforms, like Didi and Uber, need to update their map service in a real-time fashion. As mentioned before, they update their traffic conditions based on the navigation data collected from app users and drivers who complete 30 million trips per day on average in Didi [12]. Similarly, more than 16 million trips are completed in Uber per day [10]. The navigation data further invoke the map service update. Faced with the massive updating requirements, a naive idea is to process them edge by edge. In this way, one shortcut may be updated multiple times due to different edge changes, which will lead to unnecessary time cost. Motivated by it, we further propose a batch index maintenance algorithm to process them batch by batch instead of one by one. We build the hierarchy of shortcuts by splitting shortcuts into multiple layers and prove that the shortcut updates on the same layer are independent. Hence, a parallel method is further proposed to process the edges on the same layer in parallel, which can accelerate calculations effectively.

1.3 Contributions

This paper has the following contributions.

1. *The first work for CHLR maintenance in dynamic road networks.* In this paper, we aim to incrementally maintain the CHLR when the road networks are dynamically changed. To the best of our knowledge, this is the first work to address the CHLR maintenance problem in dynamic road networks.
2. *Efficient CHLR maintenance algorithms.* In this paper, we devise an efficient CHLR maintenance algorithm. Specifically, we design a novel update propagate chain to only propagate the updated shortcuts, which can avoid massive invalid updates in **Baseline**. Moreover, we propose an optimization strategy based on weight count of shortcuts, which can guarantee that shortcuts will only be recomputed when necessary. We prove that the optimized algorithm has a tightly theoretical boundedness.
3. *Optimized algorithms to handle batch update.* To handle batch edge updates, we propose a batch index maintenance algorithm by building the hierarchy of shortcuts and updating shortcuts layer by layer in a bottom-up manner. We also show that shortcut changes within the same layer do not affect each other; thus, a

parallel method is proposed to further improve the efficiency.

4. *Extensive experiments on the real road networks.* We conduct extensive and comprehensive experiments on real road networks. As shown in our experiments, the optimized CHLR maintenance algorithm has good efficiency and effectiveness. It can achieve up to 2 orders of magnitude speedup compared with **Baseline**. We also evaluate the efficiency of our batch index maintenance algorithm and parallel method.

2 Related Work

2.1 Shortest Path Query in General Graphs

Graph analysis has been receiving increasing attention recently [7–9, 21, 32, 38]. Shortest path query in general graphs is an important problem in graph analysis [17, 26, 29, 30, 33, 40]. [29] computes the shortest path graph with exactly all shortest paths for any pair of vertices. [40] proposes a disk-based index that supports both single source distance queries and single source shortest path queries with small I/O and CPU costs. Both [26] and [30] provide parallel method to speed up shortest paths counting. [33] focuses on the constrained shortest path(CSP) query over a large time-dependent graph with time function and weight function on each edge. However, none of the above works consider the label information on edges, like toll road and trucks prohibited road, which are also important to help discover more personalized shortest paths. Considering the label information of edges, [17] studies the edge-constrained shortest path (ECSP) query problem where the target is to find a shortest path after filtering some edges based on user-specified query labels.

2.2 Shortest Path Indexes in Road Networks

In the literature, many shortest path query (SPQ) algorithms are proposed based on traditional methods, such as Dijkstra [13]; they compute the shortest path on weighted road networks directly. Recently, many index-based SPQ algorithms are proposed achieving remarkable results on speeding up the query process. Hub Labeling (HL) [1, 3, 18] is one of the most important shortest path indexes, which can improve the efficiency of SPQ by reducing the points on the shortest path. Pruned Landmark Labeling (PLL) [2] and Parallel Shortest-distance Labeling (PSL) [19] are further proposed based on HL. Arterial Hierarchy (AH) [39] index can improve the query time efficiently by splitting a graph into grid structure. Contraction Hierarchies (CH) [4, 11, 16] is a fundamental

index that can reduce the number of iterations during the SPQ process by introducing shortcuts. Moreover, Hierarchical 2-Hop Labeling (H2H) [24] and P2H [6] combine the advantages of 2-hop labeling and hierarchy among all vertices. [27] studies the Label Constraint Shortest Path (LCSP) query problem in labeled and weighted road networks which can explore the shortest path $P_{s,t}^R$ between two query vertices and avoid labels in R . Based on CH, [27] proposes an important shortest path index under the label restrictions (CHLR). In CHLR, there are a series of shortcuts $e = ((u, v), w, l)$, the shortcuts can build bridges among vertices to speed up the shortest path search process. Based on CHLR, the shortest path $P_{s,t}^R$ can be found faster than online search. [27] also proposes an efficient CHLR construction algorithm. [20, 31] experimentally investigate the performance of different SPQ algorithms in multiple dimensions. In the above literature, most of them focus on the shortest path query on static road networks. However, in real world, the road networks can be constantly changed. Therefore, it is necessary to study the method of maintaining the shortest path index in dynamic road networks.

2.3 Shortest Path Index Maintenance in Dynamic Road Networks

In real world, road networks often change in practice. Hence, it is necessary to study the maintenance of shortest path indexes. [36] gives a solution for maintaining HL index in dynamic road networks. [6] introduces mechanisms for P2H index maintenance for edge weight updating. In order to maintain CH index, a number of methods [25] are proposed. Moreover, for the theoretical boundedness of the dynamic maintenance of CH and H2H, [37] presents the theoretical analysis according to the weight increase and decrease scenario separately. However, most of the existing works on dynamically maintaining shortest path indexes focus on the scenario with edge weight change. But the labels of the edges in road networks will be changed as well. To the best of our knowledge, for the important shortest path index CHLR, the corresponding index maintenance method is still to be investigated, which is studied in this paper.

3 Preliminaries

Let $G = (V, E, w, \Sigma, \mathcal{L})$ be a directed, weighted and labeled road network, where V and E are the set of vertices and edges in the road network, respectively. w represents the weight function $w : E \rightarrow \mathbb{R}^+$ mapping each edge to a positive weight number. Σ is a finite alphabet used for

labeling edges in G and ℓ is a function $\ell : E \rightarrow \Sigma$ mapping edges to labels in Σ . Besides, a vertex order function $\phi : V \rightarrow \{1, \dots, |V|\}$ is used to sort vertices in G . The neighbors of a vertex v are denoted as $N(v)$. The neighbors with higher/lower order than v are denoted as $N^{+/-}(v)$. We use n and m to denote the number of vertices and edges in the road network, respectively. A path in G is denoted as $P_{s,t} = \{e_1, e_2, \dots, e_k\}$ passing from s to t . The length of the path is expressed as $w(P_{s,t}) = \sum_{1 \leq i \leq k} w(e_i)$. The label of path is denoted as $\ell(P_{s,t}) = \ell(e_1) \cup \ell(e_2) \dots \cup \ell(e_k)$. The order of P is the minimum order of vertices in the path, denoted as $\phi(P)$. For simplicity, G is denoted as $G = (V, E)$ if the context is self-evident. Note that the approach proposed in this paper can be smoothly adapted to undirected road networks as an undirected edge can be considered as two directed edges. The descriptions for the symbols and notations used in the paper are shown in Table 1.

Definition 1 (Label Constrained Shortest Path (LCSP)) Given a road network $G = (V, E)$, a restricted label set R and two vertices s and t , the LCSP under restricted labels R from s to t is a path $P_{s,t}^R = \{e_1, e_2, \dots, e_k\}$, such that (1) for $1 \leq i \leq k$, $\ell(e_i) \cap R = \emptyset$, i.e., the path avoids all restricted

labels in R ; (2) $w(P_{s,t}^R)$ is minimum, i.e., $P_{s,t}^R$ is the shortest path satisfying condition (1).

Example 1 Considering the road network G shown in Fig. 1a, for a LCSP query with $s = v_{11}$, $t = v_9$, $R = \{c\}$, there are two LCSPs $P_1 = \{(v_{11}, v_2), (v_2, v_8), (v_8, v_9)\}$ and $P_2 = \{(v_{11}, v_{10}), (v_{10}, v_9)\}$. For both paths, the length is 7, the restricted label c is avoided, and there is no other path shorter than them without label c .

3.1 Contraction Hierarchies with Label Restrictions

To address LCSP query efficiently, [27] proposes a novel shortest path index based on Contraction Hierarchies (CH) with label restrictions.

Definition 2 (Contraction Hierarchies with Label Restrictions (CHLR)) Given a road network $G = (V, E)$ and a vertex order function ϕ , for two edges $e' = (u, q)$ and $e'' = (q, v)$ where $\phi(q) < \phi(u)$, $\phi(q) < \phi(v)$, a shortcut $e = ((u, v), w, l)$ is created with $w = w(e') + w(e'')$ and $l = \ell(e') \cup \ell(e'')$ if there is no path P' shorter than $P = \{e', e''\}$ satisfying $\phi(P') > \phi(P)$ and $\ell(P') \subseteq \ell(P)$. The CHLR index on G is the set of shortcuts, denoted as E' . The road network with CHLR is called augmented graph, denoted as $G' = (V, E \cup E')$.

Note that e can represent a shortcut or an original edge depending on the context. To distinguish them, since a shortcut $e = ((u, v), w, l)$ may also be an original edge in G , the ‘edge’ is only be used to denote the original edge in G . Moreover, e can be simplified as $e = (u, v)$ when the context is self-evident.

Example 2 Considering the road network G in Fig. 1a, the order of each vertex is marked close the vertex. The CHLR index on G is shown in Fig. 1b, which is represented in blue dotted lines. For instance, shortcut $e' = ((v_1, v_9), 4, \{a, b, c\})$ is a shortcut for edges (v_1, v_3) and (v_3, v_9) , as $\phi(v_3) < \phi(v_1)$, $\phi(v_3) < \phi(v_9)$ and there is no path P' from

Table 1 The list of symbols and notations

Notation	Description
G	The road network
V	The set of vertices in G
E	The set of edges in G
n, m	The number of vertices and edges in G , respectively
G'	The road network with CHLR in G
E'	The set of shortcuts in G'
w	The weight function of edges
ℓ	The label function of edges
Σ	A finite alphabet used for labeling edges
ϕ	The order function of vertices
$N(v)$	The neighbors of a vertex v
$d_{\max}$	The maximum degree of vertices in G'
$P_{s,t}$	A path passing from s to t
$P_{s,t}^R$	A LCSP passing from s to t without labels in R
$N^-(e)$	The parents of e
$N^+(e', e'')$	The child of e' and e''
$N^=(e)$	The partner of e
$\text{cnt}(e)$	The number of parents supporting weight of e
E_{Δ}	The changed shortcuts in CHLR
$\text{Lay}(e)$	The layer number of e

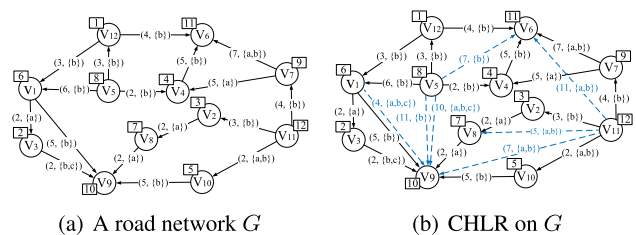


Fig. 1 An example for road network G and CHLR on G

v_1 to v_9 shorter than $P = \{(v_1, v_3), (v_3, v_9)\}$ satisfying $\phi(P') > \phi(P)$ and $\mathcal{L}(P') \subseteq \mathcal{L}(P)$.

3.2 CHLR-based LCSP Query

Given a road network G , and G' with CHLR, to address a LCSP query for $P_{s,t}^R$, the result can be retrieved by using a bidirectional Dijkstra query variant which performs a simultaneous forward query from s in upward graph $G'^{\uparrow}$ and backward query from t in the downward graph $G'^{\downarrow}$, where $E(G'^{\uparrow}) = \{e = (u, v) | \phi(u) < \phi(v) \wedge \mathcal{L}(e) \cap R = \emptyset\}$ and $E(G'^{\downarrow}) = \{e = (u, v) | \phi(u) > \phi(v) \wedge \mathcal{L}(e) \cap R = \emptyset\}$. The query terminates once the minimum key for the priority queue of Dijkstra's algorithm exceeds the tentative shortest distance and $P_{s,t}^R$ on G is obtained.

3.3 CHLR Construction

Given a road network G and a vertex order function ϕ , CHLR can be constructed through contracting vertices in the increasing order of ϕ . When contracting a vertex q , based on Definition 2, for each pair of neighbors (u, v) in $N^+(q)$, a potential shortcut $e = (u, v)$ is considered with $\mathcal{L}(e) = \mathcal{L}((u, q)) \cup \mathcal{L}((q, v))$ and $w(e) = w((u, q)) + w((q, v))$. If there is a path P' shorter than $P = \{(u, q), (q, v)\}$ satisfying $\phi(P') > \phi(P)$ and $\mathcal{L}(P') \subseteq \mathcal{L}(P)$, e is unnecessary and can be omitted in G' . Otherwise, e is created as a shortcut and inserted into G' . P' can be retrieved by Dijkstra's algorithm from u to v through vertices with higher order than q and edges with labels in $\mathcal{L}(P)$. When all vertices are contracted, all shortcuts in CHLR are obtained.

3.4 Problem Statement

In this paper, we investigate the CHLR maintenance problem on dynamic road networks. Given a road network G and edge change including weight increase/ decrease and label change, we incrementally update the CHLR index on G after the edge change.

4 Baseline Approach

To address the CHLR maintenance problem, in this section, we first explore a **Baseline** approach. Considering a changed edge $e = (q, v)$ with new weight or label, the main idea of **Baseline** is to explore the affected shortcuts and recompute the affected shortcuts. There are two steps: (1)

Explore the affected shortcuts. Due to the change of $e = (q, v)$, assuming $\phi(q) < \phi(v)$, the shortcuts like $e' = (u, v)$ produced by contracting q and incident to e may be changed. As e' can be updated by recontracting q who can produce e' , q is marked as the vertex to be recontracted and stored into Θ . Recursively, the change of e' may further affect more shortcuts. Similarly, the vertices supporting the affected shortcuts are stored into Θ . (2) *Recompute the affected shortcuts.* When all effected shortcuts are explored, they can be updated by recontracting the vertex in Θ by invoking the existing CHLR construction algorithm [27].

Example 3 Consider the road network G shown in Fig. 1a. If an edge $e = ((v_4, v_6), 5, \{b\})$ is changed to $e = ((v_4, v_6), 1, \{c\})$, following the **Baseline**, at step 1, the vertices to be recontracted in Θ are $\{v_4, \dots, v_7, v_{11}, v_{12}\}$. Then, we recontract them in the increasing order of ϕ at step 2. For instance, when recontracting v_4 , it will produce two new shortcuts $e_1 = ((v_5, v_6), 3, \{b, c\})$ and $e_2 = ((v_7, v_6), 6, \{a, c\})$.

Theorem 1 Given a road network G and a changed edge e , the time complexity of **Baseline** to update CHLR index is $O(|\Theta| \cdot d_{\max}^2 \cdot m \cdot \log n)$, where $d_{\max}$ is the maximum degree of vertices in G' .

Proof At step 1, for each affected shortcut e , exploring the shortcuts that e further affects and finding the vertices that can update e consume $O(d_{\max})$ time, and the number of the affected shortcuts is $O(|\Theta|)$. Next, at step 2, for each vertex in Θ , it can produce $O(d_{\max}^2)$ shortcuts. Furthermore, to generate a shortcut, [27] invokes Dijkstra's algorithm with time complexity $O(m \cdot \log n)$. Therefore, the total time complexity of **Baseline** is $O(|\Theta| \cdot d_{\max}^2 \cdot m \cdot \log n)$. $\square$

4.1 Drawbacks of Baseline

Unfortunately, the **Baseline** approach may introduce unnecessary overhead. First, the affected shortcuts explored at step 1 may do not change their weights and labels. Second, at step 2, all shortcuts incident to vertices in Θ will be recomputed by vertex contraction. However, most of them are unchanged in fact. All of these lead to unnecessary overhead of **Baseline**.

5 CHLR Maintenance Algorithm

In this section, we explore a new CHLR maintenance algorithm to update index more efficiently by avoiding the overhead of **Baseline** for unchanged shortcuts.

5.1 Framework

Considering a road network G and a changed edge e , due to the weight or label change of e , we can recursively update shortcuts in a BFS-like way. In our new CHLR maintenance algorithm, we aim to reveal an update propagate chain among shortcuts. Only when current shortcut e' is changed, it will extend the update propagate chain to update more shortcuts incident to e' . In this way, the unnecessary overhead of **Baseline** for updating unchanged shortcuts can be reduced sharply.

Definition 3 (*Neighborhood of Shortcuts*) Given the CHLR index on the road network G , for a shortcut $e = (u, v)$ and two shortcuts $e' = (u, q)$ and $e'' = (q, v)$, if $\phi(q) < \phi(u)$, $\phi(q) < \phi(v)$ and $\mathcal{L}(e) = \mathcal{L}(e') \cup \mathcal{L}(e'')$, then, e' and e'' are the parents of e , denoted as $(e', e'') \in N^-(e)$. e is the child of e' and e'' , denoted as $e = N^+(e', e'')$. e' is a partner of e'' , denoted as $e' \in N^=(e'')$.

Lemma 1 Given a road network G and a shortcut e , the weight of e can be computed by $w(e) = \min\{w, w(e') + w(e'') \mid \forall (e', e'') \in N^-(e)\}$, where w is the weight of e in G if e is an original edge in G .

5.2 Algorithm of CalWeight

Based on Lemma 1, we first propose a weight calculation algorithm to compute the weights of shortcuts on the augmented graph G' , which is named **CalWeight**. The pseudocode is shown in Algorithm 1. Since the pseudocode is self-explanatory, we omit the detailed description for brevity. Note that **cnt** of Algorithm 1 can be ignored here which will be introduced later.

Algorithm 1 CalWeight

Input: shortcut: e
Output: the weight of e : k

```

1:  $k \leftarrow \infty$ ;  $\text{cnt}(e) \leftarrow 0$ ;
2: if  $e \in G$  and  $w(e) \neq \infty$  then  $k \leftarrow w(e)$ ;  $\text{cnt}(e) \leftarrow 1$ ;
3: for each  $(e', e'') \in N^-(e)$  do
4:   if  $w(e') + w(e'') < k$  then  $k \leftarrow w(e') + w(e'')$ ;  $\text{cnt}(e) \leftarrow 1$ ;
5:   else if  $w(e') + w(e'') = k$  then  $\text{cnt}(e) \leftarrow \text{cnt}(e) + 1$ ;
6: return  $k$ ;
```

After obtaining the weights of shortcuts, based on Definition 2, the shortcuts with same (u, v) but different labels may conflict with each other in G' . For instance, there are two shortcuts $e = ((u, v), w_1, \{a, b\})$ and $e' = ((u, v), w_2, \{a, b, c\})$, $w_1 < w_2$. Then, for any LCSP

query with restricted label set R , if the target $P_{s,t}^R$ passes through u and v , e has higher priority than e' and $P_{s,t}^R$ will never pass through e' . It's because e' has heavier weight and stricter label constraint than e . To distinguish the conflict relationship, we propose the formal definition of shortcut dominance below.

Definition 4 (*Shortcut Dominance*) Given a road network G and CHLR on G , considering two shortcuts e and e' for vertices u and v with different weights or labels, if $w(e) \leq w(e')$ and $\mathcal{L}(e) \subset \mathcal{L}(e')$, then e dominates e' , e' is dominated by e . $\Phi(e)$ is the dominant shortcut set of e , $e' \in \Phi(e)$.

Lemma 2 Given a road network G and CHLR on G , considering two shortcuts e and e' , if e dominates e' , then e' can be removed from CHLR safely.

Algorithm 2 KeepSCDom

Input: shortcut: e ; type
Output: the queue: Q

```

1: if  $\exists e' \in G'$ , s.t.,  $e'.u = e.u$  and  $e'.v = e.v$  and  $\mathcal{L}(e') \subset \mathcal{L}(e)$  and  $w(e') \leq w(e)$  then
2:    $w(e) \leftarrow \infty$ ;  $\text{cnt}(e) \leftarrow 0$ ;  $\Phi(e') \leftarrow \Phi(e') \cup e$ ;
3: else
4:   if type = 'inc' then
5:     for each  $e' \in \Phi(e)$  do
6:        $k \leftarrow \text{CalWeight}(e')$ ;
7:       if  $k < w(e)$  then
8:         if  $(e', \text{'dec'}) \notin Q$  then  $Q.\text{push}(e', \text{'dec'})$ ;
9:          $w(e') \leftarrow k$ ;  $\Phi(e) \leftarrow \Phi(e) \setminus e'$ ;  $G' \leftarrow G' \cup \{e'\}$ ;
10:   if type = 'dec' then
11:     for each  $e' \in G'$ , s.t.,  $e'.u = e.u$  and  $e'.v = e.v$  and  $\mathcal{L}(e') \supset \mathcal{L}(e)$  do
12:       if  $w(e') \geq w(e)$  then
13:         if  $(e', \text{'inc'}) \notin Q$  then  $Q.\text{push}(e', \text{'inc'})$ ;
14:          $w(e') \leftarrow \infty$ ;  $\text{cnt}(e') \leftarrow 0$ ;  $\Phi(e) \leftarrow \Phi(e) \cup e'$ ;
15: return  $Q$ ;
```

5.3 Algorithm of KeepSCDom

Following Lemma 2, we propose a **KeepSCDom** algorithm to keep the shortcut dominance, the pseudocode is shown in Algorithm 2. Given a shortcut e , **type** is used to mark the weight increase or decrease of e , Q is a priority queue to store the changed shortcuts. Based on Definition 4, if e can be dominated by e' in the augmented graph G' , e is marked as invalid shortcut in G' , the weight of e is assigned as ∞ (line 1–2). Otherwise, if the weight of e increases, we recompute the weight of shortcut e' in $\Phi(e)$, if $w(e') < w(e)$, e' is not dominated by e anymore. In this case, e' becomes valid in G' (line 4–9). On the other hand, if the weight of e decreases, it can dominate more shortcuts who have heavier weight and longer label than e . Such

dominated shortcuts become invalid in G' (line 10–14). Since the weights of shortcuts in line 9 and line 14 are updated, they are pushed into Q to further update more shortcuts.

Now, we analyze the update propagate chain among shortcuts.

Definition 5 (*Update Propagate Chain*) When the weight of a shortcut e decreases or increases, e only changes its child shortcuts e'' in $N^+(e, e')$, where e' is a partner of e in $N^-(e)$. Specifically,

- when $w(e)$ decreases, update $w(e'') = w(e) + w(e')$ if $w(e'') > w(e) + w(e')$;
- when $w(e)$ increases, recompute $w(e'')$.

If $w(e'')$ changes, **KeepSCDom** algorithm is utilized to keep the shortcut dominance and e'' is pushed into Q to be the next shortcut of the update propagate chain.

Definition 6 (*Update Order*) Given the augmented graph G' , the update order of edges $e = ((u, v), w, l)$ is the order to be processed in the priority queue Q , denoted as $\phi(e)$. For other arbitrary edge $e' = ((u', v'), w', l')$, $\phi(e) < \phi(e')$ if $\min\{\phi(u), \phi(v)\} < \min\{\phi(u'), \phi(v')\}$ or $\min\{\phi(u), \phi(v)\} = \min\{\phi(u'), \phi(v')\}$ and $|l| < |l'|$.

Theorem 2 Based on Definition 5 and Definition 6, when an edge change for e arrives, performing the update propagate chain from e and propagating the update following the update order can update CHLR correctly.

Proof Theorem 2 holds based on two significant facts. (1) Given an updated shortcut e , e only affects its child shortcuts e'' . The weight of e'' can be updated following Definition 5. Specifically, if $w(e)$ decreases, it only needs to determine whether e will produce a shorter path for e'' , if so, $w(e'') = w(e) + w(e')$. Otherwise, if $w(e)$ increases, it's necessary to recompute $w(e'')$ by invoking **CalWeight**. This is because due to the weight increase of e , the shortest path for e'' may be destroyed by e and a new shortest path should be generated by other parents of e'' . Moreover, for other shortcuts incident to e but with lower order than e , since e will only affect the shortcuts with higher order than it based on the contracting process of CHLR, they will not be affected by e . (2) For accuracy guarantee, parents shortcuts should be updated before their child shortcut. By Definition 3 and Definition 6, for

shortcut $e'' = N^+(e, e')$, we have $\phi(e) < \phi(e'')$ and $\phi(e') < \phi(e'')$. Hence, updating the shortcut with lowest order in Q at each time ensures that their child shortcut will be updated correctly as the weights of their parents shortcuts are up to date now. In this way, CHLR can be updated correctly. $\square$

Algorithm 3 CHLRMaintenance

Input: augmented graph: G' ; changed edge: $e_o = ((u, v), w_o, l_o)$; new weight: w_n ; new label: l_n
Output: the updated augmented graph: G'

```

1:  $Q$ : priority queue;  $Q \leftarrow \emptyset$ ;
2: if  $l_n \neq l_o$  then
3:    $e_o \leftarrow ((u, v), \infty, l_o)$ ;  $e_n \leftarrow ((u, v), w(e_n), l_n)$ ;  $w(e_o) \leftarrow$ 
     CalWeight( $e_o$ );
4:   if  $w(e_o) > w_o$  then  $Q.push(e_o, 'inc')$ ; KeepSCDom( $e_o, 'inc'$ );
5:   if  $w(e_n) > w_n$  then  $w(e_n) \leftarrow w_n$ ;  $Q.push(e_n, 'dec')$ ;
     KeepSCDom( $e_n, 'dec'$ );
6: else
7:    $e_o \leftarrow ((u, v), w_n, l_o)$ ;  $w(e_o) \leftarrow$  CalWeight( $e_o$ );
8:    $Q.push(e_o, w(e_o) > w_o ? 'inc' : 'dec')$ ; KeepSCDom( $e_o, w(e_o) > w_o ? 'inc' : 'dec'$ );
9: while ! $Q.empty()$  do
10:   $(e, type) \leftarrow Q.pop()$ ;
11:  /* update weight for child shortcut */
12:  for each  $e' \in N^-(e)$  do
13:     $e'' \leftarrow N^+(e, e')$ ;
14:    if  $type = 'inc'$  and  $w(e'') \neq \infty$  then
15:      /* recompute weight of  $e''$  */
16:       $k \leftarrow$  CalWeight( $e''$ );
17:      if  $w(e'') < k$  and  $(e'', 'inc') \notin Q$  then
18:         $w(e'') \leftarrow k$ ;  $Q.push(e'', 'inc')$ ; KeepSCDom( $e'', 'inc'$ );
19:      if  $type = 'dec'$  then
20:        /* directly update weight of  $e''$  */
21:        if  $w(e'') > w(e) + w(e')$  then
22:           $w(e'') \leftarrow w(e) + w(e')$ ;
23:          if  $(e'', 'dec') \notin Q$  then  $Q.push(e'', 'dec')$ ;
     KeepSCDom( $e'', 'dec'$ );
```

5.4 Algorithm of CHLRMaintenance

Based on Theorem 2, we propose our CHLR maintenance algorithm, the pseudocode is shown in Algorithm 3. Given the augmented graph G' , a changed edge e_o with new weight w_n and new label l_n , we first update e_o and create e_n with new weight and label (line 2–8). If the label of e_o is changed, we compute the weight of e_o with old label, if the weight increases, e_o is pushed into Q with $type = 'inc'$, meanwhile, if weight of e_n with new label decreases, e_n is pushed into Q with $type = 'dec'$ (line 3–5). Otherwise, e_o is pushed into Q depending on the weight increase or

To further improve the efficiency of CHLR maintenance, we explore an optimization strategy based on the following observation. Reconsidering the update operation of Algorithm 3, for a shortcut e from Q , when the weight of a shortcut e increases, we have to recompute the weight of each child shortcut e'' of e (line 16 of Algorithm 3) whenever $w(e'')$ will change or not. However, if e does not support the current weight of e'' , i.e., $w(e'') < w(e) + w(e')$ where e and e' are the parents of e'' , $w(e'')$ will not be affected by e . Moreover, even e and e' can support the current weight of e'' , i.e., $w(e'') = w(e) + w(e')$, if there exists another couple of parents e_1 and e'_1 that can support $w(e'')$, $w(e'')$ will not change as well. To avoid the invalid computation for weight, we propose an optimized CHLR maintenance algorithm based on weight count. We first give the definition of weight count.

Definition 7 (*Weight Count*) Given a road network G and the CHLR on G , for each shortcut e in CHLR, the weight count of e is the number of parents that can support the weight of e , denoted as $\text{cnt}(e)$, i.e., $\text{cnt}(e) = |\{(e_1, e_2) | w(e) = w(e_1) + w(e_2), (e_1, e_2) \in N^-(e)\}|$. Note that if e is an original edge in G with weight $w(e)$, $\text{cnt}(e)$ is increased by one.

Lemma 3 *In the update propagate chain, when the weight of a shortcut e increases, for e 's each child shortcut e'' , $e'' = N^+(e, e')$, if $w(e'') < w(e) + w(e')$ or $\text{cnt}(e'') > 1$, $w(e'')$ will not change, the weight update for e'' can be skipped safely.*

5.6 Algorithm of CHLRMaintenance*

Following Lemma 3, we devise our optimized algorithm CHLRMaintenance*. The pseudocode is shown in Algorithm 4. Given the augmented graph G' , a changed edge e_o , new weight w_n and new label l_n for e_o , we first update the weight count cnt for e_o and e_n . The weight $w(e_o)$ is destroyed, $\text{cnt}(e_o)$ is assigned as 0 and e_o is pushed into Q with $\text{type} = \text{'inc'}$ (line 3). For shortcut e_n , if $w(e_n) = w_n$, we increase $\text{cnt}(e_n)$ by one (line 4). Besides, if $w(e_n)$ decreases due to w_n , $\text{cnt}(e_n)$ is assigned as 1 and e_n is pushed into Q with $\text{type} = \text{'dec'}$ (line 5). On the other hand, if the label of e_n is unchanged, $\text{cnt}(e_o)$ can be updated similarly depending on the change of $w(e_o)$ (line 7–10). After that, the update propagate chain starts from the shortcuts in Q . At each time, Q selects the shortcut e with lowest order $\phi(e)$ to propagate update. If e will increase its weight, for each child shortcut e'' , $e'' = N^+(e, e')$, based on Lemma 3, only if $\text{cnt}(e'') \geq 1$ and $w(e) + w(e') = w(e'')$, $\text{cnt}(e'')$ is decreased by 1. Then, if $\text{cnt}(e'') < 1$, e'' is pushed into Q with $\text{type} = \text{'inc'}$ (line 15–18). Otherwise, if e has decreased its weight and

$w(e'') = w(e) + w(e')$, then we add $\text{cnt}(e'')$ by 1. If $w(e'') > w(e) + w(e')$, we directly update $w(e'')$ by $w(e) + w(e')$ and assign 1 to $\text{cnt}(e'')$. Meanwhile, e'' is pushed into Q with $\text{type} = \text{'dec'}$ (line 19–23). Finally, we update $w(e)$ if $\text{type} = \text{'inc'}$ and invoke KeepSCDom algorithm (line 24–25). Note that, in line 14 of KeepSCDom algorithm, we omit the update for $w(e')$ as it will be updated later in line 24 of CHLRMaintenance* algorithm. When all shortcuts in Q finish their update propagation, we obtain the new CHLR on G' correctly.

Example 5 Considering the road network G in Fig. 1a and CHLR on G shown in Fig. 1b, Fig. 3 shows the update process of Algorithm 4. Given the same updated edge $e_o = ((v_4, v_6), 5, \{b\})$, and new weight 7 and new label $\{c\}$ for e_o , when process the child shortcuts of e_o , compared with Algorithm 3, the update operation for $e_1 = ((v_5, v_6), 7, \{b\})$ is skipped as $\text{cnt}(e_1) = 2$, Algorithm 4 just needs to decrease $\text{cnt}(e_1)$ by 1. Hence, Algorithm 4 can improve the efficiency of CHLR maintenance.

Theorem 3 *Given a road network G and a changed edge e , the time complexity of CHLRMaintenance* algorithm to update CHLR index is $O(|E_\Delta| \cdot \log|E_\Delta| + |E_\Delta| \cdot d_{\max})$, where E_Δ denotes the changed shortcuts in CHLR.*

Proof In CHLRMaintenance* algorithm, only the changed shortcuts will be pushed into the priority queue Q . Q needs to sort shortcuts based on their update order which consumes $O(|E_\Delta| \cdot \log|E_\Delta|)$ time. Moreover, when update a shortcut $e = (u, v)$ in Q , if $\text{type} = \text{'dec'}$, e can be updated in $O(1)$ time (line 22 of Algorithm 4), otherwise, if $\text{cnt}(e) < 1$, we need to recompute the weight of e by visiting its parents which costs $O(d_{\max})$ time. Meanwhile, to keep the shortcut dominance, in the worst case, the weights of all shortcuts incident to (u, v) should be computed (line 6 of Algorithm 2) with $O(d_{\max})$ time. Therefore, the total time complexity of CHLRMaintenance* is $O(|E_\Delta| \cdot \log|E_\Delta| + |E_\Delta| \cdot d_{\max})$. $\square$

6 CHLR Maintenance for Batch Update

On the real road network, the frequency of edge updates is high. Many edges may change in a very short period of time. When dealing with a large number of edge updates, a naive idea is to process them edge by edge through invoking CHLRMaintenance* algorithm. In this way, one shortcut may be updated multiple times due to different edge changes, which will lead to unnecessary time cost. Motivated by it, in this section, we propose a CHLR batch maintenance algorithm for batch update. The main thought is that build the hierarchy of edges/shortcuts by splitting them into multiple layers and update shortcuts in a bottom-

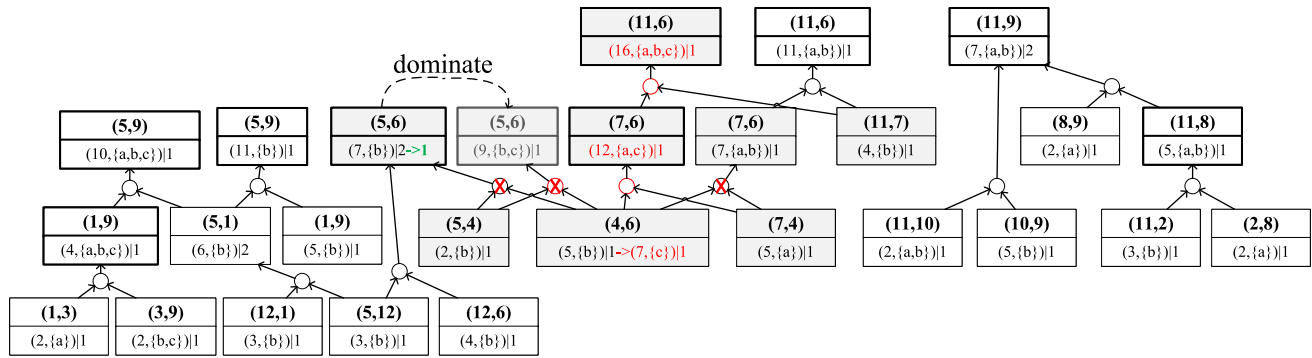


Fig. 3 Example for update process of CHLRMaintenance* algorithm

up manner. Following the thought, all shortcuts will be only updated once when necessary. We first define the layer of edges as follows.

Definition 8 (*Layer of Shortcut*) Given a road network G' with CHLR, for each shortcut e in G' , the layer number of e is denoted as $Lay(e)$. $Lay(e) = 1$ if the child set of e is empty, otherwise $Lay(e) = \max\{Lay(e')\} + 1$, where e' is a child of e .

Example 6 For the road network G in Fig. 1a and the CHLR of G in Fig. 1b, Fig. 3 gives the neighborhood of shortcuts. By Definition 8, we find that shortcut $((11, 9), 7, \{a, b\})$ has no child, so its layer is 1, and the layer number of its parent $((8, 9), 2, \{a\})$ is the maximum number of layers of all children plus 1, which is 2.

Lemma 4 For each shortcut layer l , shortcuts update on l only affect the shortcuts on the upper layers and have no effect on the shortcuts on the lower layers.

Proof For each shortcut e , e' and e'' are the parents of e . Based on Lemma 1, e will be only affected by its parents e' and e'' . By Definition 8, the layer of e is upper than that of e' and e'' . Hence, the lemma holds. $\square$

Lemma 5 Update shortcuts from lower layer to upper layer in a bottom-up manner, the correctness of this approach can be guaranteed.

Proof Reconsidering the update process of CHLRMaintenance* algorithm, when each update is propagated, for each shortcut e , it must wait for all its parents to be updated before getting its new value. Based on Definition 8, all parents of e are on the lower layers; when update e , all its parents have already been updated. Hence, the correctness of our bottom-up approach can be guaranteed. $\square$

Algorithm 5 CHLRMaintenance*-Batch

Input: augmented graph: G' ; changed edges: $U = \{e_1 = ((u_1, v_1), w_{o1}, l_{o1}, w_{n1}, l_{n1}), \dots, e_k = ((u_k, v_k), w_{ok}, l_{ok}, w_{nk}, l_{nk})\}$
Output: the updated augmented graph: G'

```

1:  $B$ : BinArray;  $B \leftarrow \emptyset$ ;
2: for each  $e_i \in U$  do  $e_o \leftarrow e_i$   $e_n \leftarrow ((u_i, v_i), w(e_n), l_{ni})$ ;
3:   if  $l_n \neq l_o$  then
4:     if  $w(e_o) = w_o$  and  $\text{cnt}(e_o) < 2$  then  $\text{cnt}(e_o) \leftarrow 0$ ;
        $B[Lay(e_o)] \leftarrow B[Lay(e_o)] \cup (e_o, \text{'inc'})$ ;
5:     if  $w(e_n) = w_n$  then  $\text{cnt}(e_n) \leftarrow \text{cnt}(e_n) + 1$ ;
6:     if  $w(e_n) > w_n$  then  $w(e_n) \leftarrow w_n$ ;  $\text{cnt}(e_n) \leftarrow 1$ ;
        $B[Lay(e_n)] \leftarrow B[Lay(e_n)] \cup (e_n, \text{'dec'})$ ;
7:   else
8:     if  $w(e_o) = w_o$  then  $\text{cnt}(e_o) \leftarrow \text{cnt}(e_o) - 1$ ;
9:     if  $w(e_o) = w_n$  then  $\text{cnt}(e_o) \leftarrow \text{cnt}(e_o) + 1$ ;
10:    if  $w(e_o) > w_n$  then  $w(e_o) \leftarrow w_n$ ;  $\text{cnt}(e_o) \leftarrow 1$ ;
        $B[Lay(e_o)] \leftarrow B[Lay(e_o)] \cup (e_o, \text{'dec'})$ ;
11:    if  $w(e_o) < w_n$  and  $\text{cnt}(e_o) < 1$  then  $\text{cnt}(e_o) \leftarrow 1$ ;
        $B[Lay(e_o)] \leftarrow B[Lay(e_o)] \cup (e_o, \text{'inc'})$ ;
12:  $i \leftarrow \max \text{layer in } B$ ;
13: while  $B \neq \emptyset$  do
14:   for each  $(e, \text{type}) \in B[i]$  do
15:     for each  $e' \in N^-(e)$  do
16:        $e'' \leftarrow N^+(e, e')$ 
17:       if  $\text{type} = \text{'inc'}$  and  $w(e'') \neq \infty$  then
18:         if  $\text{cnt}(e'') \geq 1$  and  $w(e) + w(e') = w(e'')$  then
19:            $\text{cnt}(e'') \leftarrow \text{cnt}(e'') - 1$ ;
20:           if  $\text{cnt}(e'') < 1$  and  $(e'', \text{'inc'}) \notin B[Lay(e'')]$  then
21:              $B[Lay(e'')] \leftarrow B[Lay(e'')] \cup (e'', \text{'inc'})$ ;
22:           if  $\text{type} = \text{'dec'}$  then
23:             if  $w(e'') = w(e) + w(e')$  then  $\text{cnt}(e'') \leftarrow$ 
24:                $\text{cnt}(e'') + 1$ ;
25:             else if  $w(e'') > w(e) + w(e')$  then
26:                $w(e'') \leftarrow w(e) + w(e')$ ;  $\text{cnt}(e'') \leftarrow 1$ ;
27:               if  $(e'', \text{'dec'}) \notin B[Lay(e'')]$  then
28:                  $B[Lay(e'')] \leftarrow B[Lay(e'')] \cup (e'', \text{'dec'})$ ;
29:             if  $\text{type} = \text{'inc'}$  then  $w(e) \leftarrow \text{CalWeight}(e)$ ;
30:             KeepSCDom  $(e, \text{'inc'})$ ;
31:             else KeepSCDom  $(e, \text{'dec'})$ ;
32:            $i \leftarrow i - 1$ ;

```

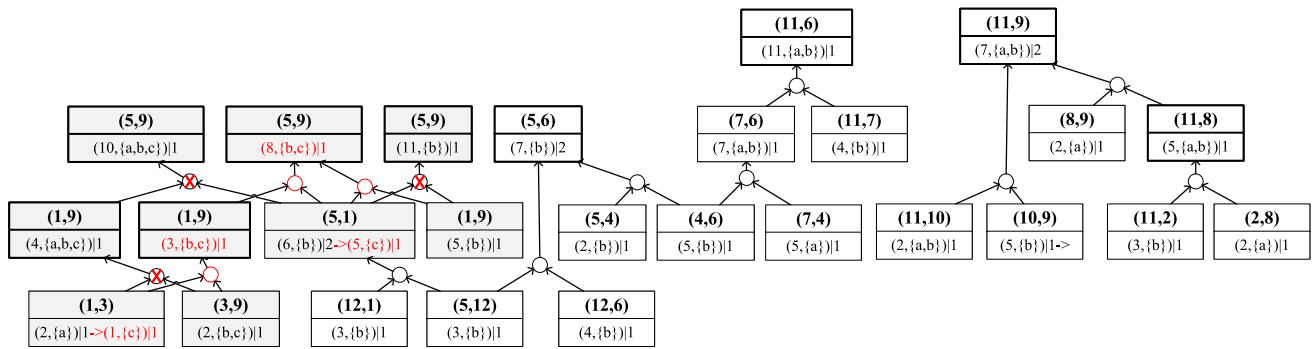


Fig. 4 Example for batch update of CHLRMaintenance*-Batch algorithm

6.1 Algorithm CHLRMaintenance*-Batch

Following Lemma 5, given a set of changed edges U , we propose our batch index maintenance algorithm CHLRMaintenance*-Batch. The pseudocode is shown in Algorithm 5. A bucket array B is used to store the shortcuts with different layers and initialized as $\emptyset$ (line 1). Similar to Algorithm 4, we first update edges in U one by one (lines 2–11). Then, we process the shortcuts on the bucket corresponding to the largest layer first (line 12). For each shortcut e that has changed, the same update propagate chain is used; e 's child e'' to be updated is deposited into $B[\text{Lay}(e'')]$ (lines 15–27). Following Lemma 4, after a layer has been processed, the next layer is processed (line 28). When all layers are processed, the CHLR index is updated correctly.

Example 7 Considering the road network G shown in Fig. 1a, given two edge changes, edge $((1, 3), 2, \{a\})$ changes to $((1, 3), 1, \{c\})$ and $((5, 1), 6, \{b\})$ changes to $((5, 1), 5, \{c\})$; according to Definition 8, it can be found that $\text{Lay}((1, 3), 2, \{a\}) = 3$ and $\text{Lay}((5, 1), 6, \{b\}) = 2$. Then according to Algorithm 5, shortcut $((1, 3), 2, \{a\})$ will be added into $B[3]$ and $((5, 1), 6, \{b\})$ will be added into $B[2]$. According to Algorithm 5, the update process starts with shortcuts in $B[3]$ and then move to upper layers. This update process is shown in Fig. 4. Compared with CHLRMaintenance* algorithm, the value of $((5, 9), \{b, c\})$ is recalculated twice due to the two edge changes. However, in Algorithm 5, the value of $((5, 9), \{b, c\})$ is only calculated once when processing $B[1]$. Thus, our Algorithm 5 can reduce unnecessary computations.

6.2 Vertex/Edge Deletion or Insertion

In this paper, we focus on the CHLR index maintenance for edge changes. However, our algorithms can also address the index maintenance for vertex/edge deletion or insertion. Specifically, when a new vertex v is inserted, we

assign $o_{\max} + 1$ to v as its order where $o_{\max}$ is the largest order in G . Then, deleting/inserting a vertex is equivalent to deleting/inserting a series of edges, so we can just focus on the deleting/inserting edge scenario. When an edge e_1 is deleted, it is equivalent to the weight of that edge being increased to infinity. Hence, the input of our Algorithm 5 is $U = \{e_1 = ((u_1, v_1), w_o, l_o, \infty, l_o)\}$ where w_o is e_1 's old weight and l_o is the label of e_1 . Then, our algorithm can update the index correctly. When an edge e_2 is inserted with weight w_n , it is equivalent to the weight of that edge being decreased from infinity to w_n . Hence, we can update our index by invoking Algorithm 5 with $U = \{e_2 = ((u_2, v_2), \infty, l_n, w_n, l_n)\}$ where l_n is the label of e_2 .

6.3 Parallel Method

To further improve the efficiency of index maintenance, we propose a parallel method to update shortcuts in parallel based on the following lemma.

Lemma 6 For each shortcut e with layer l , updating e does not affect other shortcuts on the l -layer.

Proof A shortcut change only affects its descendants via its children. When two shortcuts belong to the same layer, by Definition 8, they are not each other's child or descendant, so they do not affect each other. $\square$

Based on Lemma 6, the order of updates within the same layer can be arbitrary. According to our batch index maintenance algorithm, combined with Lemma 6, they provide an opportunity to process shortcuts at the same layer in parallel. In this way, the efficiency of maintaining index can be obviously improved through distributing the processing of shortcuts on the same layer to multiple threads, and the specific implementation is given below.

We use OpenMP¹ for parallel computing. OpenMP is a multi-threaded programming solution for shared memory parallel systems. Serial code can be easily transformed into

¹ <https://www.openmp.org/>.

Table 2 Statistic of the real datasets

Dataset	n	m	$ \Sigma $	$d_{\max}$	Dataset	n	m	$ \Sigma $	$d_{\max}$
NV	261,155	311,043	15	145	FL	1,048,506	1,330,551	23	133
LA	413,574	499,254	23	147	CA	1,613,325	1,989,149	28	139
OH	676,058	842,872	24	142	TX	2,073,870	2,584,159	25	152
IL	793,336	1,012,817	23	129	US	23,538,407	57,131,384	32	258

parallel execution with OpenMP. By simply adding `#pragma` instructions to the code, programs can be automatically processed in parallel, which greatly reduces the difficulty and complexity of parallel programming. Therefore, we rely on OpenMP to process updates of shortcuts within the same layer in parallel. Specifically, in Algorithm 5, the *for* loop starting at line 14 is used to process the updates of the shortcuts in the bucket with layer i , so the *for* loop can be run in parallel. *Openmp parallel for* assigns the computation load for different shortcuts on the same layer to multiple threads to achieve our parallel method for batch update.

7 Performance Studies

7.1 Datasets

We evaluate our algorithms on 8 real road networks, in which the weight of edges is the length of each road and the label shows the category of different roads. The datasets are downloaded from DIMACS.² Table 2 gives the statistic of datasets, where $|\Sigma|$ is the number of labels and $d_{\max}$ represents the maximum degree of vertices in the corresponding augmented graphs. All the experiments are conducted on a machine with Xeon(R) Gold 6258R CPU 2.70GHz and 1TB RAM running Linux. The core number of CPUs is 32.

7.2 Algorithms

We compare the performance among these algorithms: (1) Rebuild: the state-of-the-art algorithm to construct CHLR index [27]; (2) Baseline: the Baseline approach proposed in Sect. 4; (3) CHLRM: the CHLRMaintenance algorithm proposed in Sect. 5; (4) CHLRM*: the algorithm CHLRMaintenance* with optimization strategy proposed in Sect. 5.5; (5) CHLRM*-B: CHLRMaintenance*-batch algorithm proposed in Sect. 6.

For the vertex order in the road networks, we generate vertex order following the method used in [16, 27] with the same setting.

7.3 Exp-1: Efficiency of Algorithms When Varying the Percent of Weight Decrease/Increase

In this experiment, we evaluate the efficiency of Baseline, CHLRM and CHLRM* with varying the percent of weight decrease/increase on all datasets. On each dataset, we randomly select 1000 edges as changed edges. We use $\mathcal{W}_o$ to denote the original weights of the changed edges. We first change edges' labels and decrease their weights as $20\% \mathcal{W}_o - 80\% \mathcal{W}_o$. The running time of three algorithms with different percent of weight decrease is recorded in Fig. 5. Then, we increase their weights as $120\% \mathcal{W}_o - 200\% \mathcal{W}_o$. The corresponding running time is shown in Fig. 6.

On all datasets, CHLRM* and CHLRM can achieve up to 2 orders of magnitude speedup compared with Baseline. It's because that Baseline often performs the update for edges whose weights are not changed, which leads to unnecessary overhead of Baseline. However, our CHLRM* and CHLRM can avoid the unnecessary overhead by a novel update propagate chain, in which only the changed shortcuts can further affect other shortcuts. For the efficiency of algorithms when varying the percent of weight increase, Fig. 6 shows similar phenomena with Fig. 5. CHLRM* is fastest among three algorithms. Furthermore, CHLRM* outperforms CHLRM on all datasets. It's because CHLRM* utilizes the optimization strategy based on weight count, which can further avoid the invalid update. Moreover, the running time of algorithms changed with varying the percent of weight decrease/increase is inapparent, which demonstrates our algorithms have good scalability with the percent of weight change.

7.4 Exp-2: Efficiency of Algorithms When Varying the Number of Changed Edges

In this experiment, we compare the efficiency of Rebuild, Baseline, CHLRM and CHLRM* algorithms with changing $0.1\%|E| - 100\%|E|$ edges. The results are shown in Fig. 7.

As shown in Fig. 7, since Rebuild is the method to rebuild the CHLR index which directly reconstructs the index, the number of changed edges has negligible effect on the running time of Rebuild, the running time of

² <https://www.diag.uniroma1.it/challenge9/data/tiger/>.

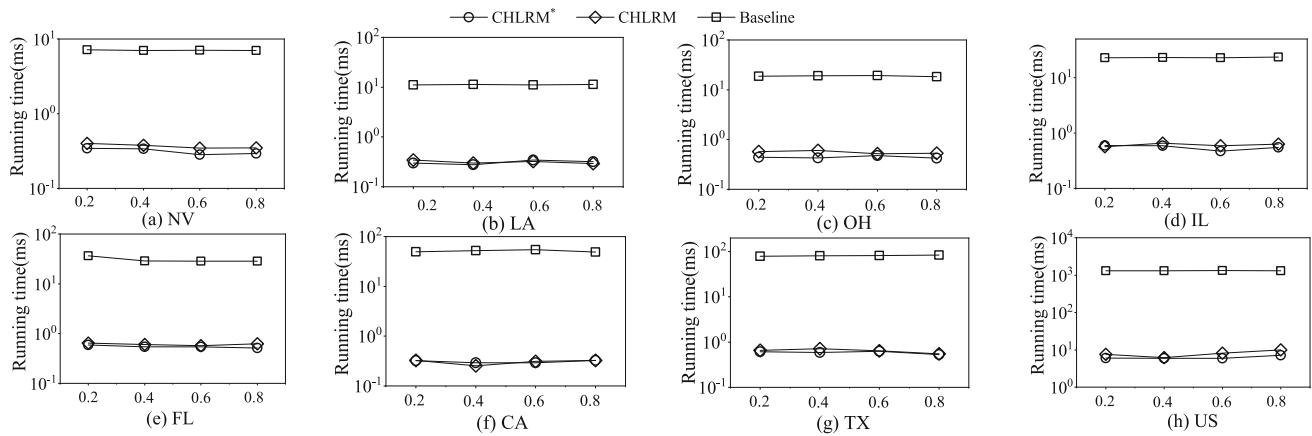


Fig. 5 Running time of different algorithms varying the percent of weight decrease

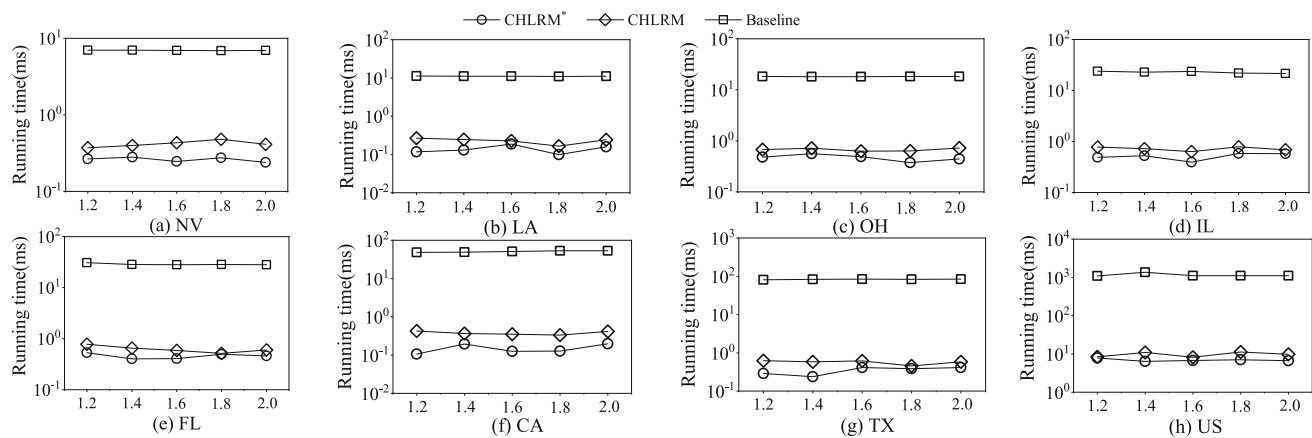


Fig. 6 Running time of different algorithms varying the percent of weight increase

Rebuild is nearly unchanged. And the running time of Baseline increases when the number of changed edges increases. It's because more shortcuts need to be updated. Moreover, CHLRM and CHLRM* are always faster than Baseline. Compared with Rebuild, CHLRM and CHLRM* spend less time updating the CHLR when 10% edges change. Meanwhile, when 100% edges are changed, the running time of CHLRM and CHLRM* is near to the time of rebuilding the CHLR by Rebuild algorithm. The experiment results indicate the significant efficiency of our CHLRM and CHLRM* algorithm.

7.5 Exp-3: Effectiveness of CHLR Maintenance Algorithms

In this experiment, we evaluate the effectiveness of different algorithms by comparing the hit rate of them. The hit rate is denoted as $\frac{|E_\Delta|}{|E_U|}$, where E_Δ denotes the shortcuts whose weights are changed and E_U denotes the shortcuts recomputed by algorithms.

As shown in Fig. 8, with the increasing of the number of changed edges, the hit rate of three algorithms increases as well because more shortcuts will be further changed. Besides, CHLRM performs better than Baseline, CHLRM* achieves the highest hit rate among them on all update cases. It verifies that our CHLRM can avoid the unnecessary update compared with Baseline, and CHLRM* can further reduce the invalid update benefitted from the optimization strategy, which also confirms the reasons for the efficiency of our optimized algorithm at Exp-1 and Exp-2.

7.6 Exp-4: Efficiency of CHLRM*-B Algorithm for Batch Update

In this experiment, we compare the running time between CHLRM* algorithm and CHLRM*-B algorithm with changing $0.1\%|E| - 100\%|E|$ edges. Figure 9 shows the results.

As shown in Fig. 9, CHLRM*-B algorithm outperforms CHLRM* algorithm on all datasets. It's because CHLRM*

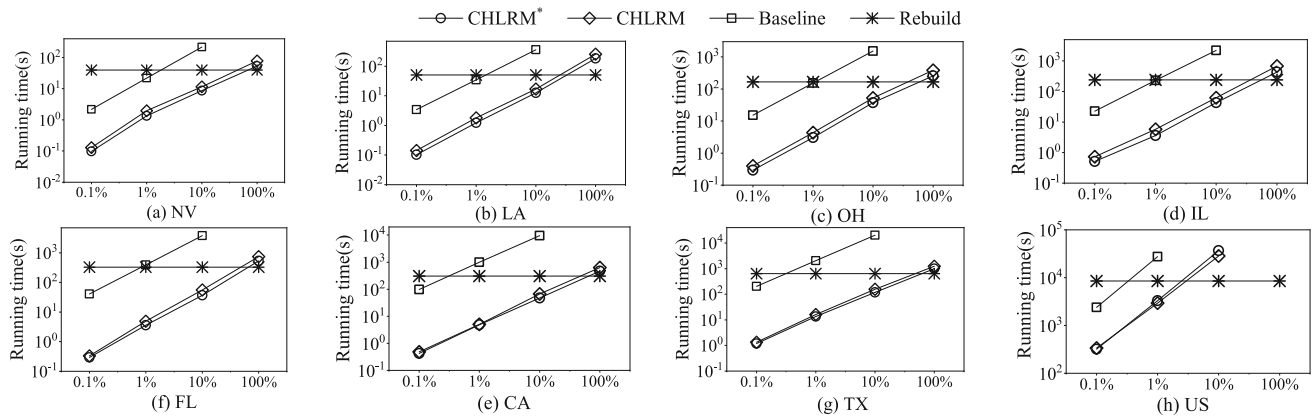


Fig. 7 Running time of different algorithms varying the number of changed edges

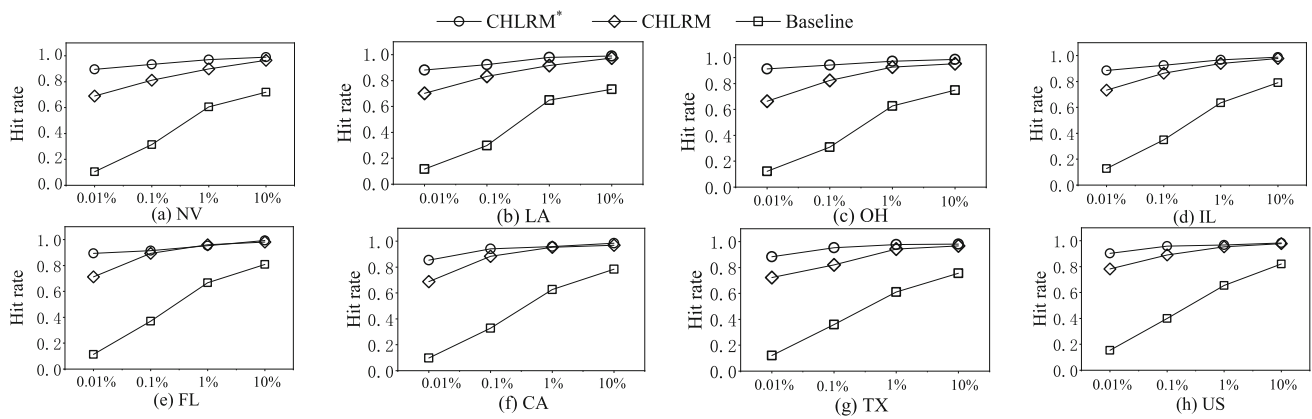


Fig. 8 Hit rate of different algorithms varying the number of changed edges

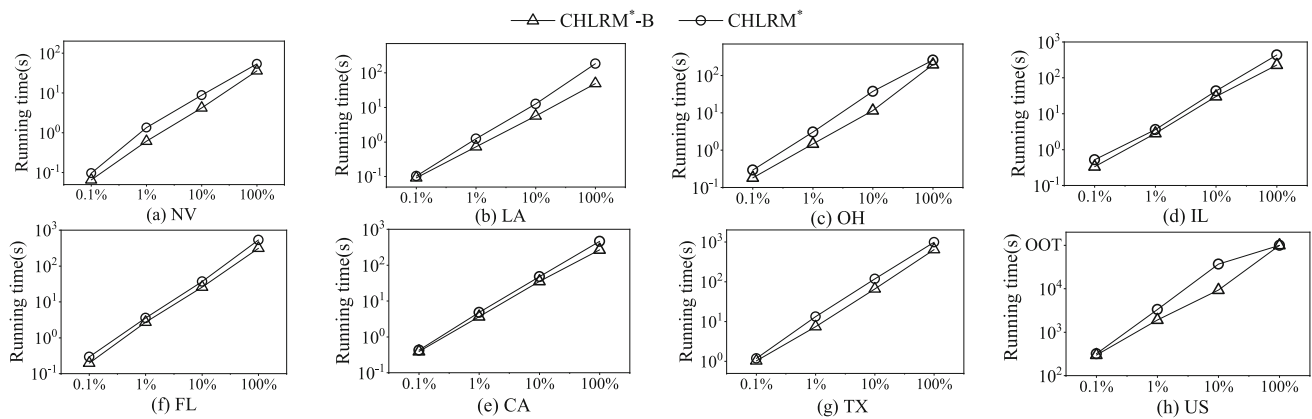


Fig. 9 Running time of CHLRM* and CHLRM*-B varying the number of changed edges

algorithm may update the shortcuts on the same layer multiple times due to different edge changes, which will introduce unnecessary time overhead. Meanwhile, the advantage of CHLRM*-B is that the shortcuts are updated layer by layer, avoiding processing the same shortcut multiple times, which improves the efficiency of index maintenance. The running time gap between CHLRM* and

CHLRM*-B increases as the number of changed edges ranges from 0.1 to 10%. But when the number of changed edges is more than 10%, the running time gap may decrease such as Fig. 9c. This is because as the number of edges in the batch increases, the number of updated shortcuts also increases, which weakens the advantages of

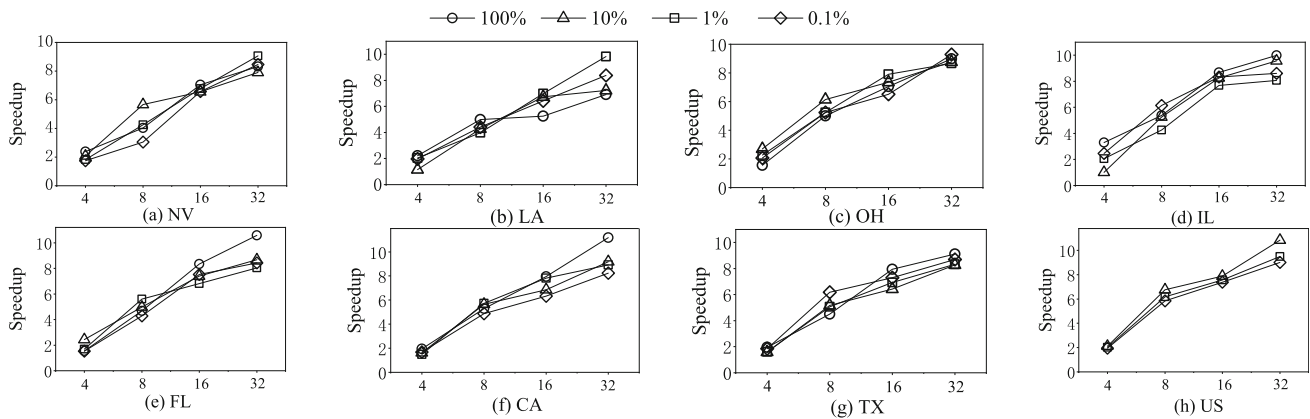


Fig. 10 The speedup with the increasing thread number for parallel method

CHLRM*-B against CHLRM*. Even though, CHLRM*-B still outperforms CHLRM*.

7.7 Exp-5: Speedup of Parallel Method When Varying the Number of Threads

In this experiment, we compare the speedup with different numbers of threads when running the parallel version of CHLRM*-B. We record the speedup of running time compared with single thread with varying the number of threads. The results are shown in Fig. 10.

By observing the results, it demonstrates that the speedup of our proposed parallel method increases as the number of threads increases. It's because that, benefited from our parallel method, the edges from the same layer can be updated in parallel. And, more threads will provide higher computational efficiency.

8 Conclusion

In this paper, we propose a novel CHLR maintenance algorithm, in which an update propagate chain is designed to propagate the shortcuts whose weights are actually changed without losing the accuracy guarantee. Moreover, an optimization strategy is presented to make sure that shortcuts are updated when necessary, it can further improve the efficiency of index maintenance. Considering the frequency of edge changes, we also propose a batch index maintenance algorithm to handle batch edge changes. Furthermore, a parallel method is proposed to further accelerate calculations. Extensive and comprehensive experiments are conducted on real road networks. The experimental results demonstrate the efficiency and effectiveness of our optimized algorithms which can achieve up to 2 orders of magnitude speedup compared with Baseline.

We also evaluate the efficiency of our batch index maintenance algorithm and parallel method.

Acknowledgements This work is supported by National Key RD Program of China 2022YFF0712100, CPSF 2021M701214, NSFC61902184, and Science and Technology on Information Systems Engineering Laboratory WZC20205250411. The preliminary version of this article has been published in DASFAA 2023 [15].

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Abraham I, Delling D, Goldberg AV, Werneck RFF (2011) A hub-based labeling algorithm for shortest paths in road networks. In: Proceedings of SEA vol 6630, pp 230–241
2. Akiba T, Iwata Y, and Yoshida Y (2013) Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: SIGMOD 349–360
3. Anirban S, Wang J, Islam MS, Kayesh H, Li J, Huang ML (2022) Compression techniques for 2-hop labeling for shortest distance queries. WWW 25(1):151–174
4. Buchhold V, Sanders P, Wagner D (2019) Real-time traffic assignment using engineered customizable contraction hierarchies. ACM J. Exp. Algorithmics 24(1):2.4:1–2.4:28
5. Chen Y, Perera L, Thompson RG (2018) An advanced method to provide best route information in city logistics with toll roads. In: ATRF
6. Chen Z, Fu AW, Jiang M, Lo E, and Zhang P (2021) P2H: efficient distance querying on road networks by projected vertex separators. In: SIGMOD, pp 313–325. ACM

7. Chen Z, Yuan L, Han L, Qian Z (2021) Higher-order truss decomposition in graphs. In: IEEE TKDE, pp 1–1
8. Chen Z, Yuan L, Lin X, Qin L, Yang J (2020) Efficient maximal balanced clique enumeration in signed networks. In: WWW, pp 339–349. ACM/TW3C2
9. Chen Z, Zhao Y, Yuan L, Lin X, Wang K (2023) Index-based biclique percolation communities search on bipartite graphs. In: ICDE, pp 2699–2712
10. Dean B (2022) Uber statistics 2022: how many people ride with uber? <https://backlinko.com/uber-users>
11. Dibbelt J, Strasser B, Wagner D (2016) Customizable contraction hierarchies. ACM J. Exp. Algorithmics 21(1):1.5:1–1.5:49
12. Didi (2022) Didi statistics and facts. <https://expandedramblings.com/index.php/didi-chuxing-facts-statistics/>
13. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271
14. Engström R (2016) The roads' role in the freight transport system. Transp Res Proc 14:1443–1452
15. Feng B, Chen Z, Yuan L, Lin X, Wang L (2023) Contraction hierarchies with label restrictions maintenance in dynamic road networks. In: Database systems for advanced applications, pp 269–285
16. Geisberger R, Sanders P, Schultes D, Delling D (2008) Contraction hierarchies: faster and simpler hierarchical routing in road networks. In: Proceedings of WEA, vol 5038, pp 319–333
17. Hassan MS, Aref WG, Aly AM (2016) Graph indexing for shortest-path finding over dynamic sub-graphs. In: SIGMOD, pp 1183–1197. ACM
18. Lakhota K, Kannan R, Dong Q, Prasanna VK (2019) Planting trees for scalable and efficient canonical hub labeling. Proc VLDB Endow 13(4):492–505
19. Li W, Qiao M, Qin L, Zhang Y, Chang L, Lin X (2019) Scaling distance labeling on small-world networks. In: SIGMOD, pp 1060–1077
20. Li Y, U LH, Yiu ML, Kou NM (2017) An experimental study on hub labeling based shortest path algorithms. Proc VLDB Endow 11(4):445–457
21. Liu B, Yuan L, Lin X, Qin L, Zhang W, Zhou J (2019) Efficient (α, β) -core computation: an index-based approach. In: WWW, pp 1130–1141. ACM
22. Mahajan B Classification of roads in india. <https://civiconcepts.com/blog/classification-of-roads>
23. Mwakalonge JL, Moses R (2012) Evaluation of truck lane restriction on non-limited access urban arterials. IJTST 1(2):191–204
24. Ouyang D, Qin L, Chang L, Lin X, Zhang Y, Zhu Q (2018) When hierarchy meets 2-hop-labeling: efficient shortest distance queries on road networks. In: SIGMOD, pp 709–724
25. Ouyang D, Yuan L, Qin L, Chang L, Zhang Y, Lin X (2020) Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees. Proc VLDB Endow 13(5):602–615
26. Peng Y, Yu JX, Wang S (2023) PSPC: efficient parallel shortest path counting on large-scale graphs. In: ICDE, pp 896–908. IEEE
27. Rice MN, Tsotras VJ (2010) Graph indexing of road networks for shortest path queries with label restrictions. Proc VLDB Endow 4(2):69–80
28. Transportation (2023) Urban public transportation in china-statistics facts. <https://www.statista.com/topics/5662/urban-public-transportation-in-china/>
29. Wang Y, Wang Q, Koehler H, Lin Y (2021) Query-by-sketch: scaling shortest path graph queries on very large networks. In: SIGMOD, pp 1946–1958. ACM
30. Wang Y, Yuan L, Chen Z, Zhang W, Lin X, Liu Q (2023) Towards efficient shortest path counting on billion-scale graphs. In: ICDE, pp 2579–2592. IEEE
31. Wu L, Xiao X, Deng D, Cong G, Zhu AD, Zhou S (2012) Shortest path and distance queries on road networks: an experimental evaluation. Proc VLDB Endow 5(5):406–417
32. Yuan L, Qin L, Zhang W, Chang L, Yang J (2018) Index-based densest clique percolation community search in networks. IEEE Trans Knowl Data Eng 30(5):922–935
33. Yuan Y, Lian X, Wang G, Ma Y, Wang Y (2019) Constrained shortest path query in a large time-dependent graph. Proc VLDB Endow 12(10):1058–1070
34. Zhang J, Li W, Yuan L, Qin L, Zhang Y, Chang L (2022) Shortest-path queries on complex networks: experiments, analyses, and improvement. Proc VLDB Endow 15(11):2640–2652
35. Zhang J, Yuan L, Li W, Qin L, Zhang Y (2021) Efficient label-constrained shortest path queries on road networks: a tree decomposition approach. Proc VLDB Endow 15(3):686–698
36. Zhang M, Li L, Hua W, Mao R, Chao P, Zhou X (2021) Dynamic hub labeling for road networks. In: ICDE, pp 336–347
37. Zhang Y, Yu JX (2022) Relative subboundedness of contraction hierarchy and hierarchical 2-hop index in dynamic road networks. In: SIGMOD, pp 1992–2005
38. Zhao Y, Chen Z, Chen C, Wang X, Lin X, Zhang W (2022) Finding the maximum k -balanced biclique on weighted bipartite graphs. In: IEEE TKDE, pp 1–14
39. Zhu AD, Ma H, Xiao X, Luo S, Tang Y, Zhou S (2013) Shortest path and distance queries on road networks: towards bridging theory and practice. In: SIGMOD, pp 857–868
40. Zhu AD, Xiao X, Wang S, Lin W (2013) Efficient single-source shortest path and distance queries on large graphs. In: SIGKDD, pp 998–1006. ACM

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.