# TRAPP: An Efficient Point-to-Point Path Planning Algorithm for Road Networks with Restrictions

Hanzhang Chen
Northeastern University
Shenyang, China
chenhz@stumail.neu.edu.cn

Xiangzhi Zhang
Northeastern University
Shenyang, China
zhangxz@stumail.neu.edu.cn

Shufeng Gong
Northeastern University
Shenyang, China
gongsf@mail.neu.edu.cn

Feng Yao
Northeastern University
Shenyang, China
yaofeng@stumail.neu.edu.cn

Song Yu
Northeastern University
Shenyang, China
yusong@stumail.neu.edu.cn

Yanfeng Zhang
Northeastern University
Shenyang, China
zhangyf@mail.neu.edu.cn

Ge Yu
Northeastern University
Shenyang, China
yuge@mail.neu.edu.cn

## ABSTRACT

Path planning is a fundamental problem in road networks, with the goal of finding a path that optimizes objectives such as shortest distance or minimal travel time. Existing methods typically use graph indexing to ensure the efficiency of path planning. However, in real-world road networks, road segments may impose restrictions in terms of height, width, and weight. Most existing works ignore these road restrictions when building indices, which results in returning infeasible paths for vehicles. To address this, a naive approach is to build separate indices for each combination of different types of restrictions. However, this approach leads to a substantial number of indices, as the number of combinations grows explosively with the increase in different restrictions on road segments. In this paper, we propose a novel path planning method, TRAPP (Traffic Restrictions Adaptive Path Planning algorithm), which utilizes traffic flow data from the road network to filter out rarely used road restriction combinations, retain frequently used road restriction combinations, and build indices for them. Additionally, we introduce two optimizations aimed at reducing redundant path information storage within the indices and enhancing the speed of index matching. Our experimental results on real-world road networks demonstrate that TRAPP can effectively reduce the computational and memory overhead associated with building indices while ensuring the efficiency of path planning.

## 1 INTRODUCTION

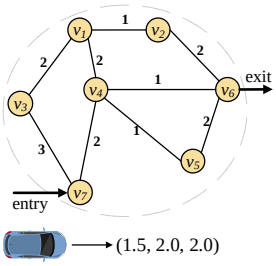
Path planning is one of the fundamental operations in various fields, such as road networks [8, 13, 33, 38, 49], bioinformatics [4, 25, 35, 44], and robot navigation [14, 18, 26, 37, 48]. With the development of the transportation industry, the issue of path planning on road networks has become increasingly critical. In general, a road network can be denoted as $G = (V, E)$, where a vertex $v \in V$ represents an intersection and an edge $e \in E$ represents a road segment. Path planning in a given road network aims to find a path with the shortest distance or time between a given source vertex $s \in V$ and destination vertex $d \in V$. Although traditional path planning algorithms, such as Dijkstra's algorithm [10], can correctly answer path queries in road networks, they need to traverse the entire road network, resulting in inefficiency for large-scale road networks. The A* search algorithm [15] can reduce unnecessary traversals by utilizing the Euclidean distance from the current vertex to the destination vertex as a heuristic function. However, the path planning time remains significantly long. This is because when the algorithm steps into an area with dense road segments, such as the center of a city, it is required to evaluate all road segments to determine the shortest path through the dense area.

To enhance the efficiency of path planning, various index-based path planning methods are proposed in the literature [5, 8, 13, 20, 33, 36, 38, 39, 41, 43, 46, 49]. These methods pre-compute the shortest paths between selected vertices in a given road network, then store and maintain these paths within an index structure to reduce the time required for path planning. CRP (Customizable Route Planning) [8] is one of the state-of-the-art index-based path planning algorithms. It partitions the entire road network into multiple dense cells and stores pre-computed shortest paths between the entry and exit vertices of each cell as *shortcuts* (a type of index structure). These shortcuts are utilized to bypass the path search in dense areas.

However, in real-world road networks, there may be restrictions on factors such as height, width, and weight [2, 7, 27, 50]. This renders index-based methods inapplicable for directly answering path queries in road networks with restrictions. This is because a query vehicle's height, width, and weight may be larger than the corresponding restrictions represented by the shortest path, leading to infeasible shortcut for the vehicle.

For example, as shown in Figure 1a, a typical shortcut from $v_7$ to $v_6$ that does not consider restrictions will point to the shortest path $v_7 \rightarrow v_4 \rightarrow v_6$. However, for a vehicle with a height of 2.5, a width of 2.0, and a weight of 3.0, this shortcut is infeasible because the edge $(v_4, v_6)$ has a height restriction of 1.8, as shown in Figure 1b, which is lower than the height of the vehicle.

To enable index-based methods to plan feasible paths for vehicles, shortcuts should be built under different road restrictions. A naive approach is to combine different height, width, and weight restrictions in the road network into distinct combinations, compute the shortest path for each combination, and store these paths into shortcuts. However, the vast majority of roads in the network feature various types of restrictions with differing restriction values. For example, approximately 73.8% of road segments in China have diverse types of restrictions with varying restriction values. Additionally, China's 1.0793 million bridges feature a wide range of weight restrictions, highlighting the diversity of restrictions across the network [28, 29, 31, 32]. Consequently, considering all restriction combinations when building shortcuts would lead to a



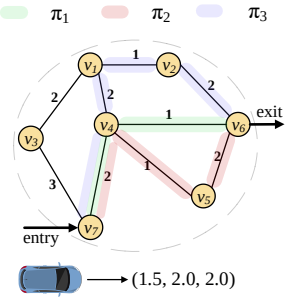

(a) A cell of a road network

| edge | he | wi | wt |
|---|---|---|---|
| ($v_1$,$v_2$) | \ | 2.4 | 10 |
| ($v_1$,$v_3$) | 3.5 | 2.5 | 20 |
| ($v_3$,$v_7$) | 2.5 | 2.0 | \ |
| ($v_4$,$v_5$) | 2.0 | \ | 15 |
| ($v_4$,$v_6$) | 1.8 | \ | \ |
| ($v_4$,$v_7$) | 4.0 | \ | 40 |

(b) Road restrictions

| shortcut path | combination (he, wi, wt) | shortest path | distance |
|---|---|---|---|
| $\pi_1$ | 1.8 , \ , 40 | $v_7 \rightarrow v_4 \rightarrow v_6$ | 3 |
| $\pi_2$ | 2.0 , 2.0 , 15 | $v_7 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6$ | 5 |
| $\pi_3$ | 2.5 , 2.4 , 10 | $v_7 \rightarrow v_4 \rightarrow v_1 \rightarrow v_2 \rightarrow v_6$ | 7 |
| ... | ... | ... | ... |
| $\pi_{120}$ | 3.5 , 2.5 , 20 | Unreachable | $\infty$ |

(c) A shortcut $\alpha_{v7,v6}$ built from vertex $v_7$ to exit vertex $v_6$, which contains shortest paths computed from entry vertex $v_7$ to exit vertex $v_6$ under all combinations of road restrictions.




(d) The paths of $\pi_1$, $\pi_2$, and $\pi_3$.

**Figure 1: An illustrative example to show the combinatorial explosion problem in shortcut construction for a road network cell with height (he), width (wi), and weight (wt) restrictions. For this road network cell, $v_7$ is the entry vertex and $v_6$ is the exit vertex. A vehicle with a height of 1.5, width of 2.0 and weight of 2.0 wants to travel from the entry vertex to the exit vertex.**

combinatorial explosion problem, which is intolerable in terms of both storage and computation. A naive way to reduce the number of shortcuts is to randomly build shortcuts for only a few of the restriction combinations. However, storing fewer shortest paths may result in vehicles obtaining sub-optimal paths. We illustrate the above with a detailed example.

**Example 1:** Figure 1 shows how shortcuts are built in a road network with different combinations of road restrictions. Figure 1a provides an example of a high-density cell of the road network, where $v_7$ is the entry vertex and $v_6$ is the exit vertex. Figure 1b records the restrictions on different road segments, due to which, we combine different type of road restrictions and computes shortest paths for each combination to ensure that each vehicle can get a feasible shortest path. Theoretically, there are 120 (*i.e.,* 6*4*5) shortcut paths in shortcut from $v_7$ to $v_6$, as shown in Figure 1c, which is 120 times greater than existing methods without considering road restrictions. Each shortcut corresponds to different shortest paths with different road restriction combinations.

If we randomly compute shortest paths for only a small subset of all combinations, it is highly likely that the shortcuts containing the optimal or acceptable paths for vehicles will be discarded. Figure 1d shows the shortest paths represented by shortcut paths $\pi_1$, $\pi_2$, and $\pi_3$ under restriction combinations $(1.8, \backslash, 40)$, $(2.0, 2.0, 15)$, and $(2.5, 2.4, 10)$. Assuming a vehicle with a height of 1.5, width of 2.0, and weight of 2.0 intends to travel from $v_7$ to $v_6$, $S_1$ represents its global shortest path. If we only store shortcuts $\pi_2$ and $\pi_3$, as shown in Figure 1d, the vehicle has to travel from $v_7$ to $v_6$ via the path represented by $\pi_2$, *i.e.,* $v_7 \rightarrow v_4 \rightarrow v_2 \rightarrow v_6$. Because $\pi_2$ is shorter than $\pi_3$. □

**Motivation**. Based on the above analysis, we find that the combinatorial explosion problem necessitates extensive pre-computation of shortcuts and results in significant memory consumption in existing methods. However, having fewer shortcuts between entry and exit vertices increases the likelihood of missing the optimal or an acceptable path, leading to potentially worse solutions. This motivates us to design a method to build fewer shortcuts in the road network and still allow most vehicles to find optimal or acceptable paths.

**Our solution**. To address the combinatorial explosion problem in shortcut construction caused by the restrictions of road segments, we propose a novel path planning method for road networks with restrictions, TRAPP. TRAPP utilizes vehicles' traffic flow data in the road network to filter out rarely used restriction combinations while preserving frequently used ones, which reduces the number of shortcuts to a reasonable range meanwhile ensuring the quality of path planning. Additionally, we introduce two optimizations: shortcuts merger to reduce memory usage and shortcuts pre-sorting to expedite shortcuts matching during path planning.

In summary, our contributions are as follows:

- **Studying and analyzing path planning in road networks with restrictions.** We formally study the problem of path planning in road networks with restrictions, providing a detailed definition of the problem. We critically analyze the limitations of existing methods when applied to restricted road networks and propose a novel approach to address these challenges.
- **Efficient path planning method.** We propose an efficient path planning method tailored for road networks with restrictions. This method significantly reduces the computational and memory overhead associated with building shortcut paths while enabling most vehicles to find optimal or acceptable paths in a short time.
- **Optimization of shortcut matching speed and shortcut storage.** We propose an optimized method for shortcut storage that significantly reduces memory usage by minimizing redundant path storage. Additionally, we have designed a method for quickly shortcut matching, which substantially decreases the matching time and thus accelerates path query processing.
- **Extensive experiments are conducted on real-world road networks.** We conducted extensive experiments on six real-world road networks. The experimental results demonstrate that our proposed path planning method TRAPP achieves excellent path planning performance while ensuring high efficiency.

**Organization**. The rest of this paper is organized as follows. We provide preliminaries in section 2. In section 3, we analyze the path planning problem in the road networks with restrictions. We provide a detailed introduction to our proposed method in section 4. Experimental results are reported in section 6 and the existing path planning methods are summarized in section 7. Finally, we conclude the paper in section 8.

## 2 PRELIMINARIES

In this section, we first formally introduce some basic concepts related to our work.

**Table 1: Frequently used notations**

| Notation | Description |
|---|---|
| $G = (V, E, R_E, L_E)$ | Road network with restrictions |
| $he, wi, wt$ | Height, width and weight |
| $R_E = \{R_E^{he}, R_E^{wi}, R_E^{wt}\}$ | Function to get restrictions |
| $rc = (he, wi, wt)$ | Road restriction combination |
| $c = (he, wi, wt)$ | Vehicle |
| $rv =< he, wi, wt >$ | Representation vector |
| $\alpha_{s,d}, S$ | Shortcut from $s$ to $d$ and a shortcut set |
| $\pi_{s,d}^{rc}$ | Shortcut path from $s$ to $d$ under $rc$ |
| $dist()$ | Function to get distance |
| $C = (V, E, R_E, L_E)$ | Cell of a road network |

**Road Network with Restrictions**. Let $G = (V, E, R_E, L_E)$ be a road network with restrictions where $V$ is a finite set of vertices that represents the intersections, $E = \{(u, v)\} \subseteq V \times V$ is a set of edges which represents road segments. $R_E = \{R_E^{he}, R_E^{wi}, R_E^{wt}\}$ contains three restriction functions such that each edge $(u, v) \in E$ carries the *height restriction* $R_{u,v}^{he}$, *width restriction* $R_{u,v}^{wi}$, and *weight restriction* $R_{u,v}^{wt}$. $L_E$ is a length function that each edge $(u, v) \in E$ carries a length value $l_{u,v}$.

**Vehicle**. A vehicle is defined as a triplet $c = (he, wi, wt)$ with three attributes, where $he$, $wi$, and $wt$ represent the vehicle's height, width, and weight, respectively.

**Feasible Vehicle Path and Its Distance**. Given a path from $v_0$ to $v_k$ that is denoted as a sequence of vertices $p_{v_0,v_k} = (v_0, v_1, \ldots, v_k)$ and a vehicle $c = (he, wi, wt)$, the path $p_{v_0,v_k}$ is a *feasible vehicle path* for the vehicle $c$ if and only if

$$c.he \leq R_{v_i,v_{i+1}}^{he},$$
$$c.wi \leq R_{v_i,v_{i+1}}^{wi},$$
$$c.wt \leq R_{v_i,v_{i+1}}^{wt},$$

for each edge $(v_i, v_{i+1}) \in p_{v_0,v_k}$. The distance of $p_{v_0,v_k}$ is the sum of the length of all road segments on $p_{v_0,v_k}$, *i.e.,* $dist(p) = \sum_{i=0}^{k-1} l_{i,i+1}$.

**Shortest Feasible Vehicle Path.** Given a set of feasible vehicle paths from vertex $v_0$ to $v_k$, $P_{v_0,v_k} = \{p_{v_0,v_k}^1, \cdots, p_{v_0,v_k}^N\}$ for the vehicle $c = (he, wi, wt)$, the *shortest feasible vehicle path* $p_{v_0,v_k}^*$ from $v_0$ to $v_k$ for $c$ is defined as

$$dist(p_{v_0,v_k}^*) \leq dist(p_{v_0,v_k}^i), 1 \leq i \leq N.$$

*Path Planning.* Given a road network with restrictions $G = (V, E, R_E, L_E)$ and a vehicle $c = (he, wi, wt)$, path planning aims to find a *shortest feasible vehicle path* $p_{s,d}^*$ from a given source vertice $s$ to a destination vertice $d$ for the vehicle $c$.

As we discussed previously, many methods [8, 13, 38, 49] accelerate path planning by building shortcuts in dense areas of road networks, *i.e.,* precomputing the shortest path from an entry to an exit of the dense area, thereby reducing the online search space. However, traditional shortcuts do not consider road restrictions when they are built, and each shortcut represents only one single shortest path. In a road network with restrictions, the shortcut should store multiple shortest paths for different road restrictions combinations, so that we can find the shortest *feasible vehicle path* for different vehicles with different heights, widths, and weights. Based on this, we formally define the shortcuts on road networks with restrictions. Before that, we first provide some basic concepts, *i.e.,* entry/exit vertices, feasible shortcut paths, and the shortest feasible shortcut path.

**Entry/Exit vertices**. Let $G_i = (V_i, E_i, R_{E_i}, L_{E_i})$ is a subgraph of a road network with restriction $G$, where $E_i \subseteq V_i \times V_i \cap E$. The vertex $v \in V_i$ is an entry (exit) vertex if it has an incoming edge $(u, v) \in E \setminus E_i$ (outgoing edge $(v, w) \in E \setminus E_i$) and $u \notin E_i$ ($w \notin E_i$).

**Feasible Shortcut Path**. Given a shortcut path $\pi_{v_0,v_k} = (v_0, \cdots, v_k)$ from the entry vertex $v_0$ to exit vertex $v_k$ of a subgraph and a restriction combination $rc = (he, wi, wt)$, the path $\pi_{v_0,v_k}$ is a *feasible shortcut path* under restriction $rc$ if and only if

$$rc.he \leq R_{v_i,v_{i+1}}^{he},$$
$$rc.wi \leq R_{v_i,v_{i+1}}^{wi},$$
$$rc.wt \leq R_{v_i,v_{i+1}}^{wt},$$

for each edge $(v_i, v_{i+1}) \in \pi_{v_0,v_k}$.

**Shortest Feasible Shortcut Path**. Given a set of feasible shortcut paths from $s$ to $d$, $\Pi_{s,d} = \{\pi_{s,d}^1, \cdots, \pi_{s,d}^N\}$ for $rc = (he, wi, wt)$, the *shortest feasible shortcut path* is defined as

$$dist(\pi_{v_0,v_k}^{rc}) \leq dist(\pi_{v_0,v_k}^i), 1 \leq i \leq N.$$

**Domination**. Given a vehicle $c = (he, wi, wt)$ and a restriction combination $rc = (he, wi, wt)$, $c$ is dominated by $rc$ if and only if

$$c.he \leq rc.he,$$
$$c.wi \leq rc.wi,$$
$$c.wt \leq rc.wt.$$

**Lemma 1:** *Given a vehicle $c = (he, wi, wt)$, a restriction combination $rc = (he, wi, wt)$, and a feasible shortcut path $\pi$ for $rc$, if $c$ is dominated by $rc$, then $\pi$ a feasible path for $c$.* □

**Proof:** Given a feasible shortcut path $\pi_{v_0,v_k} = (v_0, \cdots, v_k)$ for $rc = (he, wi, wt)$ and a vehicle $c = (he, wi, wt)$ dominated by $rc$. We have $rc.he \leq R_{v_i,v_{i+1}}^{he}$, $rc.wi \leq R_{v_i,v_{i+1}}^{wi}$ and $rc.wt \leq R_{v_i,v_{i+1}}^{wt}$ for

each edge $(v_i, v_{i+1}) \in \pi_{v_0,v_k}$. Since $c.he \leq rc.he$, $c.wi \leq rc.wi$ and $c.wt \leq rc.wt$. Then we have $c.he \leq R^{he}_{v_i,v_{i+1}}$, $c.wi \leq R^{wi}_{v_i,v_{i+1}}$ and $c.wt \leq R^{wt}_{v_i,v_{i+1}}$ for each edge $(v_i, v_{i+1}) \in \pi_{v_0,v_k}$, *i.e.,* $\pi_{v_0,v_k}$ is a feasible path for $c$. □

**Shortcut on Restriction Road Network**. Given a dense area of the road network with restrictions, where the dense area is a subgraph of the road network and is called a dense cell, the *shortcut* $\alpha_{s,d} = \{\pi^{rc_0}_{s,d}, \pi^{rc_1}_{s,d}, \ldots, \pi^{rc_k}_{s,d}\}$ stores a set of shortest paths from $s$ which is the entry vertex of $C$ to $d$ which is the entry vertex of $C$, each computed under different road restriction combinations $rc$. Here, $rc$ is represented as $rc = (he, wi, wt)$, where $he$, $wi$, and $wt$ denote height restriction, width restriction, and weight restriction, respectively.

**Example 2:** As illustrated in Figure 1c, a shortcut $\alpha_{v_7,v_6} = \{\pi_0, \pi_1, \ldots, \pi_{120}\}$ from vertex $v_7$ to vertex $v_6$ is built.

As illustrated in Figure 1c, $\{\pi_1, \pi_2, \ldots \pi_{120}\}$ are the shortcut paths computed from $v_7$ to $v_6$, where $v_7$ is the source vertex, and $v_6$ is the destination vertex. Taking $\pi_3$ as an example, $\pi_3$ is the shortest path computed under the restriction combination $(2.5, 2.4, 10)$, meaning that the height restriction of $\pi_3$ (*i.e.,* $R^{hi}(\pi_3)$) is 2.5, the width restriction (*i.e.,* $R^{wi}(\pi_3)$) is 2.4, and the weight restriction (*i.e.,* $R^{wt}(\pi_3)$) is 10.0. The shortcut path $\pi_3$ consists of the road segments $(v_7, v_4)$, $(v_4, v_1)$, $(v_1, v_2)$, and $(v_2, v_6)$, with a total distance of 4. □

## 3 PROBLEM STATEMENT

In this section, we first introduce the state-of-the-art two-stage framework for path planning. Then we analyze the problem of the combinatorial explosion of this framework on the road networks with restrictions, which leads to a surge in memory and computation. Based on that, we introduce the goal of this paper.

### 3.1 Two-stage path planning framework

The existing state-of-the-art path planning algorithms [8, 13, 38] employ a two-stage framework to search the shortest path for the given source and destination vertices. This framework contains an offline preprocessing stage and an online query stage.

In the offline preprocessing phase, it first identifies the dense areas of the road network, called dense cells of the road network, which is done by the community detection algorithms, such as PUNCH [9], METIS [21], KaPPa[17] and SCOTCH [34]. Then it builds shortcuts from the entry vertex to the exit vertex of the dense cell. In the online querying phase, it employs an existing smart path planning algorithm, such as Dijkstra [10] and A* search [15], to find the shortest path. When entering a dense cell, the path planning algorithm directly visits the shortest path represented by the shortcuts to avoid extensive searches within the cell, thereby significantly enhancing the efficiency of path planning.

As shown in Figure 2, in the preprocessing stage, it first partitions the road network $G$ shown in Figure 2a in into multiple cells, where different background colors represent different cells. Next, it computes the shortest paths between the entry and exit vertices within each cell and stores these paths in shortcuts, as illustrated in Figure 2c. In this figure, red dashed lines represent the shortcuts built between the entry and exit vertices, and red solid lines

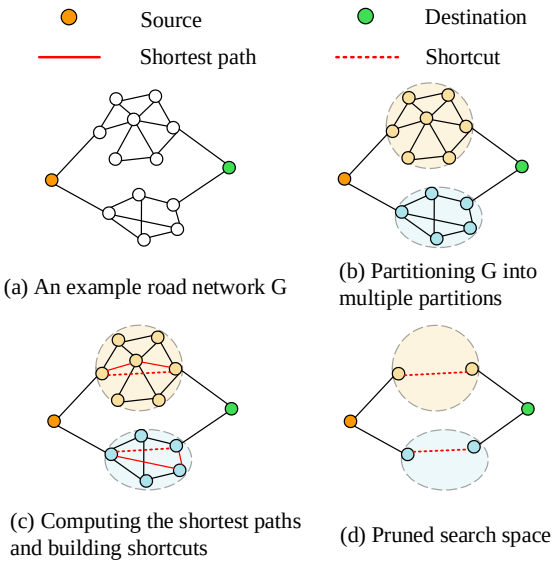



**Figure 2: An example to illustrate the process of shortcut building by existing methods. The vertices of the road network represent the intersections, the edges represent roads, and the shortcuts represent the record of pre-computed shortest paths.**

indicate the actual shortest paths recorded by these shortcuts. As shown in Figure 2d, with the help of shortcuts, the search space is significantly reduced.

**Drawbacks**. The traditional two-phase framework introduced above does not consider the restrictions within the road network, and each shortcut only stores a single shortest path that is treated as feasible for all vehicles. However, in road networks with restrictions, some edges may have restrictions, as introduced in Section 2. This necessitates the pre-computation of multiple *feasible shortcut paths* under different restriction combinations, allowing vehicles with different height, width and weight to find their *feasible shortest paths*. Unfortunately, dense cells contain many edges and, consequently, a variety of road restrictions with differing values. This results in an exponential increase in the number of restriction combinations, *i.e.,* combinatorial explosion problem. To illustrate, consider a cell with $N_{he}$ different height restrictions, $N_{wi}$ different width restrictions, and $N_{wt}$ different weight restrictions. The number of all restriction combinations in this cell is $N_{he} * N_{wi} * N_{wt}$. For example, as shown in Figure 1, the cell depicted in Figure Figure 1a has 120 restriction combinations (*i.e.,* 6*4*5).

Therefore, computing feasible shortcut paths for all road restrictions incurs significant computational and storage overhead. Additionally, this approach greatly increases the time required for online queries. This is because, for a vehicle $c$, if the restriction combination $rc$ of the shortest feasible shortcut path in the shortcut dominates $c$, then that path is a feasible path for $c$ (as demonstrated in Lemma 1). The more restriction combinations there are, the more feasible paths are stored in the shortcut for $c$, leading to an increased search effort to find the shortest feasible vehicle path for $c$. In other words, the combinatorial explosion problem not only

results in significant computational and storage overhead but also increases the online planning time.

**Naive Solution**. To avoid the calculation and storage of a large number of shortest feasible shortcut paths, a naive method is to randomly select a part of the restriction combinations and calculate their shortest feasible shortcut paths. Since some shortest feasible shortcut paths for some restriction combinations are discarded, the planned path may not be the shortest. This is because if the shortest feasible shortcut path, for a given vehicle, is discarded on a shortcut, this vehicle will search for another shortest feasible shortcut path for wider restriction so that the vehicle can find a feasible path. However, the shortest feasible shortcut path of wider restriction always is longer. Therefore, the path planned by the shortest feasible shortcut paths that are randomly saved in the shortcut is not the shortest feasible vehicle path, but longer than the actual shortest path. The fewer the shortest feasible shortcut paths saved in the shortcut, the longer the planned path will be compared to the shortest feasible vehicle path.

Only calculating and saving a small number of shortest feasible shortcut paths will lead to a longer shortest feasible vehicle path. For a given number of shortest feasible shortcut paths, is there any shortest feasible shortcut paths selection method that minimizes the shortest feasible vehicle path for all vehicles?

**Our Goal**. We aim to design a shortest feasible shortcut path selection method $\mathcal{S}$, such that for a vehicle set $C = \{c_i\}$, the sum of the length of the shortest feasible vehicle path is minimized, *i.e.,*

$$minimize \sum_{i=1}^{N} dist(\mathcal{P}_{c_i}(\mathcal{S}(\Pi))) \tag{1}$$

where $\mathcal{P}_{c_i}$ is the shortest path planning method, *e.g., A**, that return the shortest feasible path.

## 4 TRAPP ALGORITHM

This section provides a detailed introduction to our proposed path planning method, TRAPP. The core idea of TRAPP is to filter road restriction combinations using traffic flow data from the road network. We retain the combinations used by the majority of vehicles and discard those used by only a few. Building shortcuts solely for the retained combinations significantly reduces the number of shortcuts, thereby greatly decreasing memory usage and computational overhead.

**Representative Vehicle Dimension Mining**. The height, width, and weight of vehicles in real-world road networks exhibit distinct clustering trends within localized regions [30]. Based on this observation, we aim to identify the primary distribution ranges of these attributes to derive representative vehicle dimensions. These representative dimensions facilitate efficient filtering of road restriction combinations in subsequent tasks.

To achieve this, we first partition the road network $G = (V, E, R_E, L_E)$ into dense cells $C_i = (V_i, E_i, R_E, L_E)$ using the PUNCH algorithm [9]. This partitioning confines the analysis of traffic flow data to individual cells, allowing the extraction of vehicle characteristics—such as height, width, and weight—that accurately reflect the traffic patterns within each region. Within each cell, each vehicle is abstracted as a three-dimensional vector $v = \langle he, wi, wt \rangle$, where $he$, $wi$, and $wt$ represent the vehicle's height, width, and weight, respectively.

To extract representative vehicle dimensions, we apply the $K$-means clustering algorithm to group vehicles with similar dimensions. The traffic flow data is partitioned into $K$ clusters $\{\mathcal{VG}_1, \mathcal{VG}_2, \ldots, \mathcal{VG}_K\}$, where $K$ is empirically set to 30 based on observed trade-offs between computational efficiency and capturing representative dimension ranges. For each cluster $\mathcal{VG}_i$, we compute a representation vector $rv_i = \langle he_{\max}, wi_{\max}, wt_{\max} \rangle$, where $he_{\max}$, $wi_{\max}$, and $wt_{\max}$ represent the maximum height, width, and weight of vehicles within the cluster, respectively. The underlying idea is to leverage the height, width, and weight attributes of the vector to construct a *restriction combination* and build a corresponding *shortcut path*. This ensures that every vehicle within the cluster can efficiently identify a *feasible shortcut path* through the built *shortcut path*, thereby reducing both computational and storage overhead due to the limited number of *shortcut paths* maintained. Subsequently, we will formally define the representation vector and provide a rigorous theoretical foundation and proof for the proposed idea.

Definition 1 (representation vector). *Given a vehicle cluster $\mathcal{VG}$, the representation vector is defined as a three-dimensional vector $rv = \langle he_{\max}, wi_{\max}, wt_{\max} \rangle$, where $he_{\max}$, $wi_{\max}$, and $wt_{\max}$ represent the maximum height, width, and weight of vehicles within the cluster $\mathcal{VG}$, respectively.*

**Lemma 2:** *For a given vehicle cluster $\mathcal{VG}$, the shortcut path built under the height $he$, width $wi$, and weight $wt$ of the cluster's representation vector $rv =< he, wi, wt >$ is guaranteed to be a feasible vehicle path for all vehicles $c \in \mathcal{VG}$.* □

**Proof:** Consider a vehicle cluster $\mathcal{VG}$ with its representation vector $rv = \langle he, wi, wt \rangle$. Let the shortcut path $\pi_{v_0,v_k} = (v_0, \ldots, v_k)$ be built based on the *restriction combination* $rc = (rv.he, rv.wi, rv.wt)$. For each vehicle $c = (he, wi, wt) \in \mathcal{VG}$, by definition of the representation vector, we have $c.he \leq rv.he$, $c.wi \leq rv.wi$, and $c.wt \leq rv.wt$. This implies that the vehicle $c$ is **dominated by** $rc$. By Lemma 1, $\pi_{v_0,v_k}$ is a *feasible vehicle path* for every vehicle $c \in \mathcal{VG}$. □

The process for computing the *representation vectors*, as detailed in Algorithm 1, takes the cells of the road network $\{C_1, C_2, \ldots, C_n\}$ and the vehicle groups $\{\mathcal{VG}_1, \mathcal{VG}_2, \ldots, \mathcal{VG}_K\}$ as input. In lines 3–9, the algorithm iteratively processes each cell $C_i$ in the road network. For each vehicle group $\mathcal{VG}_k$ within a cell, lines 6–7 compute the maximum values of height, width, and weight across all vehicles in the group to form a representative vector $rv_k$, which summarizes the key attributes of the group. Finally, in line 10, the algorithm aggregates all representation vectors into the global set $RV$, providing a compact summary of vehicle characteristics to facilitate further filtering and optimization.

**Cell-aware Refinement** In this stage, we integrate the previously computed *representation vectors* with the actual road restriction combinations specific to each cell in the road network, tailoring this integration to the unique restrictions of each cell. By mapping the aggregated ranges of vehicle height, width, and weight to the

**Algorithm 1:** Computing representation vectors

**Input:** $\mathcal{VG}$:set of vehicle groups$\{\mathcal{VG}_1, \mathcal{VG}_2, \ldots, \mathcal{VG}_n\}$
$C$: set of road network cells$\{C_1, C_2, \ldots, C_n\}$
**Output:** RV: set of representation vectors
$\{RV_1, RV_2, \ldots, RV_n\}$

```
1  RV ← ∅
2  for each C_i ∈ C do
3      RV_i ← ∅
4      for each VG_i ∈ VG do
5          rv_i ← ∅
6          for T ∈ {he, wi, wt} do
7              rv_i.T ← max{c_j.T | c_j ∈ C_i}
8          RV_i ← RV_i ∪ {rv_i}
9      RV ← RV ∪ RV_i
10 return RV
```

corresponding road restrictions, our method enables efficient filtering of road restriction combinations, significantly reducing the number of stored restriction combinations and the corresponding index size. This process not only optimizes storage requirements but also simplifies subsequent queries, thereby enhancing the overall system performance.

Before delving into the specific methodology, we first introduce the concepts related to the mapping of *representation vectors* and the objectives of filtering road restriction combinations.

Definition 2 (representation vector mapping). *Consider a cell $C$ in a road network, with the set of restriction combinations $RC = \{rc_1, rc_2, \ldots, rc_n\}$ and the set of representation vectors $RV = \{rv_1, rv_2, \ldots, rv_k\}$. A representation vector mapping is defined as the process of mapping each $rv_i \in RV$ to a unique $rc_j \in RC$, forming a one-to-one correspondence $f : RV \rightarrow RC$ such that for every $rv_i \in RV$, there exists a unique $rc_j \in RC$.*

In this paper, we define the mapping method $f$ for the representation vector as mapping each $rv$ to the $rc$ in the set of *road restrictions RC* with the smallest Euclidean norm with respect to the current $rv$. This mapping method is to get the most similar restriction combination for representation vector. Formally, the mapping method is defined as:

$$f(rv) = \arg\min_{rc \in RC} \|rv - rc\|_2 \tag{2}$$

where $\|rv - rc\|_2$ represents the Euclidean norm between $rv$ and $rc$. Thus, in a given cell $C$, our objection for the mapping function is defined as:

$$minimize \sum_{i=1}^{K} \|rv_i - f(rv_i)\|_2 \tag{3}$$

A naive approach to solving the mapping problem involves constructing all possible road restriction combinations within a given cell and iterating through them to determine the corresponding $rc$ for each representation vector $rv$. While conceptually straightforward, this approach suffers from significant computational inefficiency, as its time complexity scales as $O(K \cdot N_{he} \cdot N_{wi} \cdot N_{wt})$ for a single cell, where $K$ denotes the number of representation vectors, and $N_{he}$, $N_{wi}$, and $N_{wt}$ represent the number of height, width, and weight restrictions, respectively. Such a prohibitive cost renders this approach impractical for large-scale road networks. To overcome these limitations, we introduce a novel and computationally efficient method that eliminates the need to exhaustively generate and evaluate all road restriction combinations. By reconsidering and systematically decomposing Equation 2, we identify that the mapping condition for each attribute $\mathcal{T} \in \{he, wi, wt\}$ can be independently optimized. This reformulation enables the mapping process to be expressed as the minimization of the $L_2$-norm for each attribute:

$$f(rv).\mathcal{T} = \arg\min_{rc \in RC} \|rv.\mathcal{T} - rc.\mathcal{T}\|_2 \tag{4}$$

, where $rv.\mathcal{T}$ and $rc.\mathcal{T}$ denote the values of the attribute $\mathcal{T}$ in the representation vector and road restriction combination, respectively. This decomposition fundamentally reduces the computational overhead by allowing independent processing of each road restriction type, obviating the need for pre-computing all possible combinations.

Based on the above analysis, we propose an efficient algorithm for computing the mapping process, as shown in Algorithm 2. The algorithm begins by iterating through each road network cell $C_i \in C$, aggregating road restrictions from all edges within the cell (Lines 1–6). For each edge $(u, v) \in E_i$ and each attribute $\mathcal{T} \in \{he, wi, wt\}$ (representing height, width, and weight), the algorithm collects non-empty road restrictions $R^{\mathcal{T}}_{u,v}$ into the corresponding restriction set $\mathcal{R}^{\mathcal{T}}_i$ for the cell. Once all restrictions are aggregated, the algorithm sorts each restriction set $\mathcal{R}^{\mathcal{T}}_i$ in ascending order to enable efficient mapping queries (Lines 7–8). Next, the algorithm processes the representation vectors $RV_i$ associated with $C_i$. For each vector $rv \in RV_i$, it iteratively maps each attribute $\mathcal{T}$ to the closest road restriction in $\mathcal{R}^{\mathcal{T}}_i$ based on Equation 4 (Lines 9–13). It is evident that the time complexity of the proposed algorithm is $O(K \cdot (N_{he} + N_{wi} + N_{wt}))$, which represents a significant reduction compared to the naive approach.

**Combination Rematch.** In real-world scenarios, most vehicles exhibit a consistent proportional relationship among height, width, and weight, with these ratios typically falling within a certain range. However, we observe that some vehicles deviate significantly from this standard pattern. For example, certain trucks may have similar height and width but vary greatly in weight, while some Special Purpose Vehicles show disproportionate ratios of height, width, and weight compared to typical passenger vehicles. These vehicles tend to result in longer paths in path planning. To substantiate this, we will first introduce key concepts and provide the corresponding theoretical framework and proofs.

Definition 3 (Feasible Edge Set). *In a given cell of a road network $C = (V, E, R_E, L_E)$, the set of feasible edges $E_{rc}$ for a restriction combination $rc = (he, wi, wt)$ is defined as:*

$$E_{rc} = \{e \in E \mid rc.he \leq R^{he}_e,\ rc.wi \leq R^{wi}_e,\ rc.wt \leq R^{wt}_e\}.$$

**Algorithm 2:** Efficient Mapping Processing for Representation Vectors

**Input:** $C$: set of road network cells $\{C_1, C_2, \ldots, C_n\}$
RV: set of representation vectors $\{RV_1, RV_2, \ldots, RV_n\}$

**for** *each* $C_i \in C$ **do**
  **for** *each* $(u, v) \in E_i$ **do**
    **for** $\mathcal{T} \in \{he, wi, wt\}$ **do**
      **if** $R^{\mathcal{T}}_{u,v} \neq \emptyset$ **then**
        $\mathcal{R}^{\mathcal{T}}_i \leftarrow \mathcal{R}^{\mathcal{T}}_i \cup R^{\mathcal{T}}_{u,v}$
        //store restrictions in corresponding sets
  **for** $\mathcal{T} \in \{he, wi, wt\}$ **do**
    sort $\mathcal{R}^{\mathcal{T}}_i$ in ascending order
  **for** *each* $RV_i \in RV$ **do**
    **for** *each* $rv \in RV_i$ **do**
      **for** $\mathcal{T} \in \{he, wi, wt\}$ **do**
        $rv.\mathcal{T} \leftarrow \arg\min_{r \in \mathcal{R}^{\mathcal{T}}_i} \|rv.\mathcal{T} - r\|_2$
        //map $rv$ attributes based on Equation 4

**Lemma 3:** *In a cell $C$ of a road network, consider two shortcut paths $\pi^{rc_i}_{v_0,v_k} = (v_0, \ldots, v_k)$ and $\pi^{rc_j}_{v_0,v_k} = (v_0, \ldots, v_k)$ with $i \neq j$, that connect the entry vertex $v_0$ to the exit vertex $v_k$. If $rc_i.he \leq rc_j.he$, $rc_i.wi \leq rc_j.wi$, and $rc_i.wt \leq rc_j.wt$, then:*

$$dist(\pi^{rc_i}_{v_0,v_k}) \leq dist(\pi^{rc_j}_{v_0,v_k}).$$

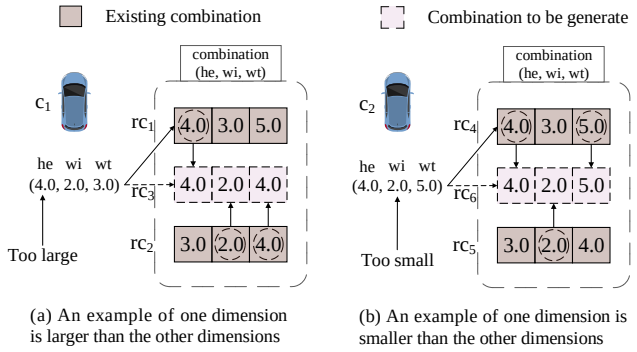
□

**Proof:** Given that $rc_i.he \leq rc_j.he, rc_i.wi \leq rc_j.wi$, and $rc_i.wt \leq rc_j.wt$, by Definition 3, we have $E_{rc_j} \subseteq E_{rc_i}$. Since $\pi^{rc_i}_{v_0,v_k}$ and $\pi^{rc_j}_{v_0,v_k}$ are the shortest paths from $v_0$ to $v_k$ under $rc_i$ and $rc_j$, respectively, and $E_{rc_j} \subseteq E_{rc_i}$, the path $\pi^{rc_i}_{v_0,v_k}$ can only use the same or more feasible edges than $\pi^{rc_j}_{v_0,v_k}$. Therefore, $dist(\pi^{rc_i}_{v_0,v_k}) \leq dist(\pi^{rc_j}_{v_0,v_k})$. □

Building on Lemma 3, it is clear that these vehicles encounter significant challenges in obtaining *shortest feasible vehicle paths* during shortcut matching. This often leads to suboptimal or unattainable paths. Consequently, targeted optimization strategies for these vehicles can effectively enhance the path planning system.

These vehicles can be broadly classified into two categories: The first category includes vehicles where one dimension (e.g., height, width, or weight) is significantly larger than the others. The second category consists of vehicles where one dimension is significantly smaller than the others, such as vehicles with weight much lower than their height and width. This imbalance reduces the effectiveness of path planning.

For the two cases discussed above, the core idea of our optimization is to recombine the components of the different existing restriction combinations to create new combinations. Before rematch, we first sort the road restriction combinations in ascending order based on their restriction values. Then, we give two examples to illustrate our approach. For the first case, as shown in Figure 4a, for vehicle $c_1 = (4.0, 2.0, 3.0)$ to find an acceptable path, we recombine the two combinations $rc_1 = (4.0, 3.0, 5.0)$ and




(a) An example of one dimension is larger than the other dimensions

(b) An example of one dimension is smaller than the other dimensions

**Figure 3: An example to illustrate the principle of combination rematch. The dashed boxes contain the currently existing restriction combinations, where he, wi, and wt denote height, width, and weight, respectively.**

**Algorithm 3:** Combination Rematch

**Input:** $C$: cells of the road network $\{C_1, C_2, \ldots, C_n\}$
$RC$: restriction combinations $\{RC_1, RC_2, \ldots, RC_n\}$

**for** *each* $C_i$ **do**
  **for** *each* $rc_j \in RC_i$ **do**
    sort $RC_i$ in ascending order
    **for** $\mathcal{T} \in \{he, wi, wt\}$ **do**
      $R \leftarrow \{rc.\mathcal{T} \mid rc \in RC_i\}$
      sort $R$ in ascending order
      $RC_i \leftarrow RC_i \cup (\text{REMATCH}(R, RC_i, rc_j, \mathcal{T}))$
**Function** REMATCH$(R, RC, rc, \mathcal{T})$:
  $M \leftarrow \emptyset$
  $pos \leftarrow$ index$(rc.\mathcal{T}, R)$
  //get the position of $rc.\mathcal{T}$ in $R$
  **for** $j \leftarrow 1$ **to** 2 **do**
    $rc_{temp} \leftarrow rc, rc_{temp}.i \leftarrow R[pos + j]$
    **if** $\theta(rc_{temp}, RC) \geq f * Total\_vehicles$ **then**
      $M.add(rc_{temp})$
    $rc_{temp} \leftarrow rc, rc_{temp}.i \leftarrow R[pos - j]$
    **if** $\theta(rc_{temp}, RC) \geq f * Total\_vehicles$ **then**
      $M.add(rc_{temp})$
  **return** $M$

$rc_2 = (3.0, 2.0, 4.0)$ that are adjacent to each other after sorting. Since $c_1$ is larger in height than in width and weight, we recombine the height of the larger combination with the width and weight of the smaller combination, i.e., we recombine the height of $rc_1.he$ with the $rc_2.wi$ and $rc_2.wt$ to get a more appropriate combination $rc_3 = (rc_1.he, rc_2.wi, rc_2.wt)$,*i.e.,* $(4.0, 2.0, 4.0)$. For the second case, as shown in Fig. 4b, since the width of the vehicle $c_2 = (4.0, 2.0, 5.0)$ is smaller compared to the height and weight, we recombine the height and weight of $rc_4 = (4.0, 3.0, 5.0)$ and the width and weight of $rc_5 = (3.0, 2.0, 4.0)$ to get a more appropriate combination $rc_6 = (rc_4.he, rc_5.wi, rc_4.wt)$,*i.e.,* $(4.0, 2.0, 5.0)$.

The details are shown in Algorithm 3. This algorithm takes the cells of road network $\{C_1, C_2, \ldots, C_n\}$ and the restriction combination sets $\{RC_1, RC_2, \ldots, RC_n\}$ as input. In lines 2-7 of the algorithm, each cell $C_i$ is processed individually. In lines 5-6, different types of restrictions are extracted in $R$ and sorted in ascending order. Subsequently, in line 7, the restriction combinations $rc_j$ are rematched, and the rematched combinations are added to the restriction combination set $RC_i$. Lines 8-19 of the algorithm illustrate the process of the rematch function. Lines 12-18 handle the two cases discussed earlier, with lines 13-15 addressing the first case and lines 16-18 addressing the second case. In lines 14 and 17, $\theta(rc_{temp}, RC)$ is a heuristic function. Since rematching all road restriction combinations would significantly increase the number of restriction combinations, this function aims to filter out those combinations that do not contribute to improving path planning efficiency, thus controlling the number of combinations. The function estimates the number of vehicles that can plan their paths using the *shortest feasible shortcut path* through the rematched road restriction combinations, using this estimate as the basis for filtering. The parameter $f$ is set to 0.03 in this paper, and Total_vehicles denotes the total number of vehicles in the traffic flow within the cell being processed.

**Building Shortcuts** After completing the filtering of road restriction combinations, we build shortcuts for these retained road restriction combinations. In each partition, we build a shortcut for each remained road restriction combination between every two boundary vertices, *i.e.,* we run modified Dijkstra's algorithm for each restriction combination between every two boundary vertices of a partition, and finally we save the shortest path we calculated by Dijkstra's algorithm as a shortcut.

The details are shown in Algorithm 4. This algorithm takes the cells of road network $\{C_1, C_2, \ldots, C_n\}$ and the restriction combination sets $\{RC_1, RC_2, \ldots, RC_n\}$ as input. The algorithm performs a shortcut building operation for each cell $C_i$ in lines 3-9. Specifically, lines 4-8 compute the *shortest feasible shortcut path* for each pair of entry/exit vertices within cell $C_i$ based on the existing restriction combinations and store this path in the shortcut $s$.

**Algorithm 4:** Building Shortcuts

**Input:** $C$: cells of the road network $\{C_1, C_2, \ldots, C_n\}$
$RC$: restriction combinations $\{RC_1, RC_2, \ldots, RC_n\}$

```
1 S ← ∅
2 for each C_i ∈ C do
3     for u, v ∈ V^i_{entry/exit} do
4         s ← ∅
5         for each rc_j ∈ RC_i do
6             p ← shortest_path(u, v, rc_j)
7             //get shortest path from u to v under rc_i
8             s.add(p)
9         S ← S ∪ s
```

# 5 OPTIMIZATION

In this section, we will provide a detailed explanation of the two optimizations proposed in this paper.

## 5.1 Shortcuts Merger

Through observation and analysis, we found that shortcut paths with different restriction combinations may have the same actual roads. Figure 4a shows an example of two different shortcut paths stored by traditional storage method, these two different shortcut paths that share the same source vertex $v_7$ and destination vertex $v_6$, but have different road restriction combinations. One shortcut path has a restriction combination of (2.0, 2.4, 10), while the other has a combination of (2.5, 2.4, 10). They both follow the same actual path: $v_7 \rightarrow v_4 \rightarrow v_1 \rightarrow v_2 \rightarrow v_6$. Using this storage method results in redundant storage, as shortcut paths with different road restriction combinations but the same actual paths save the same actual road multiple times. In real-world road networks, the actual paths of shortcut paths often have long distances, leading to substantially higher storage requirements compared to other shortcut path information. Therefore, minimizing the redundant storage of identical actual paths across different shortcuts can markedly reduce memory consumption.

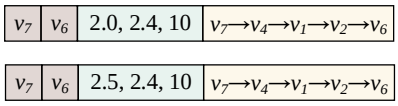
Based on the above analysis, we propose a novel shortcut storage method that reduces memory consumption by separating the actual paths from other information of the shortcut paths and using pointers to link other information to the actual paths. Figure 4b gives an illustration of our proposed shortcut path storage method. Our method stores the source vertex, destination vertex, restriction combinations, and actual paths of shortcut paths separately, and assigns pointers to the shortcut paths to link to their corresponding actual paths. Consequently, shortcut paths with different road restriction combinations but identical actual paths only need to store the actual path once. Given the need to establish a large number of shortcut paths in real-world road networks, our proposed storage method can significantly reduce the memory consumption associated with shortcuts compared to traditional methods.

## 5.2 Shortcuts Pre-sorting

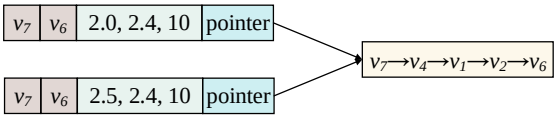
Given a shortcut $\alpha_{s,d} = \{\pi^{rc_0}_{s,d}, \pi^{rc_1}_{s,d}, \ldots, \pi^{rc_k}_{s,d}\}$ and a vehicle $c = (he, wi, wt)$. When a vehicle $c$ performs a path search in a shortcut, it must identify the path with the shortest distance among all *feasible paths* within the shortcuts to ensure that it finds the *shortest feasible vehicle path*, thereby achieving the optimal path. We refer to this operation as *shortcut matching*.

**Example 3:** For example, assuming that there is a vehicle $c = (2.0, 2.0, 10.0)$. Thus, as shown in Figure 1c, shortcut path $\pi_2$ and $\pi_3$ are the matching shortcut paths for the query. Since $\pi_2$ is the matching shortcut path with the minimum distance, $\pi_2$ is best matching shortcut path. □

**Traditional Shortcut Path Matching Method.** In the query stage, the traditional shortcut matching method fully traverses the set of shortcut paths to find the best matching shortcut path. Obviously, the time complexity of performing shortcut matching in this way is $O(n)$.




(a) An example of shortcut paths stored by traditional storage method.




(b) An example of shortcut paths stored by our proposed storage method.

**Figure 4: An illustration to show how different storage methods store shortcut paths.**

**Our Proposed Shortcut Path Matching Method.** To speed up shortcut matching, we propose a novel shortcut path matching method. This method pre-sorts the shortcut paths in ascending order based on their distances when building them. Therefore, during shortcut matching, we only need to traverse to the first shortcut path that is feasible for the vehicle to find the optimal matching shortcut path. Using this method for shortcut path matching, the best case occurs when the first shortcut is the optimal matching shortcut, with a time complexity of $O(1)$. The worst case occurs when the last shortcut path is the optimal matching shortcut path, with a time complexity of $O(n)$. On average, the optimal matching shortcut path has an equal chance of being at any position in the set, and the probability for each position is $\frac{1}{n}$. Therefore, the average number of searches is $\frac{1}{n}\sum_{i=1}^{n} i$, which is $\frac{n+1}{2}$ times. Therefore, the time complexity in this case is $O\left(\frac{n+1}{2}\right)$. Thus, it can be seen that our proposed method has the same time complexity as the traditional method even in the worst case, thereby significantly speeding up the shortcut path matching process.

## 6 EXPERIMENTS

All algorithms are implemented in C++ and all the experiments are conducted on a linux machine with Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz and 96 GB RAM.

### 6.1 Experimental Setup

**Datasets.** As shown in table 2, we use six real-world road networks which are all obtained from OpenStreetMap [1] in our experiment: Shenyang (SY), Tokyo (TYO), Paris (PAR), London (LON), Los Angeles (LA), and San Francisco (SF). Each road network is undirected, and consists of a set of vertices that represent intersections and a set of edges that represent a road segment. The weight on each edge denotes the distance of the road segment. Meanwhile, we randomly assigns some road restrictions to each edge in these road networks. As there are currently no publicly available datasets containing road restriction information, we conducted sampling in a selected area of Shenyang, China, to collect and analyze data on local road restrictions. Based on the statistical results of this sampling, we assigned realistic road restrictions to each edge within the road networks, ensuring that the experimental data closely reflects real-world conditions.

[1] https://www.openstreetmap.org/

**Table 2: Dataset description**

| Road Network | #Vertices | #Edges |
|---|---|---|
| Shenyang (SY) | 47,773 | 105,378 |
| Tokyo (TYO) | 94,016 | 212,754 |
| Paris (PAR) | 137,411 | 292,038 |
| London (LON) | 155,221 | 326,882 |
| Los Angeles (LA) | 161,384 | 346,764 |
| San Francisco (SF) | 300,617 | 633,958 |

For traffic flow data, we generated the data based on commonly seen vehicles (high-selling vehicles) on the roads.

**Comparison Methods.** As previously analyzed, existing index-based path planning algorithms are unable to generate feasible paths in road networks with restrictions, therefore, we compare TRAPP with the two naive methods mentioned in Section 3 and the modified Dijkstra algorithm.

- **Dijkstra**: The modified Dijkstra's algorithm, which searches only the edges that are feasible for a vehicle, can obtain the *shortest feasible vehicle path*.
- **Random**: The method which randomly stores *shortest feasible paths* under a certain proportion of road restriction combinations when building shortcuts.
- **All**: The method which stores *shortest feasible paths* of all road restriction combinations when building shortcuts.
- **TRAPP**: Our proposed path planning algorithm for road networks with restrictions.

**Query Set.** For each dataset, we randomly select two vertices that do not belong to the same cell as the source and destination vertices, respectively, and randomly select a vehicle from the traffic flow data as the generation rule for a path query. Based on this rule, we generate 300 path queries as the query set.

**Evaluation Metrics.** We use the following metrics to evaluate path planning performance: (1) *Efficiency*, measured by the average time per query in path planning. (2) *Space Utilization*, assessed by the number of stored paths in the shortcut structure, where lower storage usage indicates better efficiency. (3) *Effectiveness*, evaluated through the Planned Path Error Rate, the Path Query Failure Rate, and the Proportion of Optimal Paths.

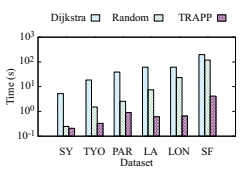



**Figure 5: Path planning time comparison.**

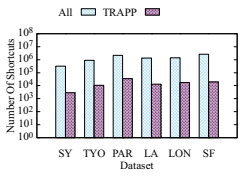



**Figure 6: Comparison of number of shortcut paths.**

## 6.2 Path Planning Time Evaluation

In this experiment, we evaluate the efficiency of path planning by comparing the runtime of the modified Dijkstra's algorithm, Random and our method TRAPP on the six datasets: SY, TYO, PAR, LON, LA, and SF, as shown in Table 2. For each dataset, we use the same query set for different methods to conduct comparative experiments, and take the average query time of a single query in the query set as the experimental result.

The experimental results are shown in Figure 5. From the results, we can see that for all datasets, the path planning time for TRAPP is significantly lower than that of the modified Dijkstra algorithm. In the best cases, the query speed of TRAPP is 426 times faster than that of Dijkstra, because Dijkstra's algorithm needs to search a large number of edges. Meanwhile, the query time for Random falls between Dijkstra and TRAPP. This is because the failure rate of the Random method is much higher than that of TRAPP, and when path planning is failed, Random uses the Dijkstra algorithm for searching, which leads to an increase in its query time.

In summary, our proposed method TRAPP effectively ensures the efficiency of path planning.

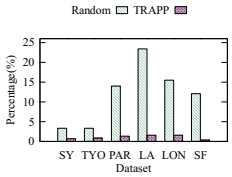



**Figure 7: Comparison of path error rate.**

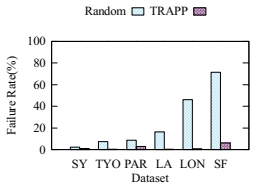



**Figure 8: Comparison of failure rate.**

## 6.3 Shortcut Storage Evaluation

To demonstrate that the shortcuts built by TRAPP can effectively control the number of actual shortest paths, we conduct experiments to test the number of shortcuts built by our proposed method TRAPP and compare it with All on six different datasets.

As shown in Figure 6, in all datasets, the number of actual shortest paths of shortcuts built by TRAPP is significantly less than that of All. In the best case, the number of actual shortest paths preserved by TRAPP is only 0.6% of those preserved by All. These experimental results show that TRAPP effectively reduces the number of actual shortest paths of built shortcuts, thereby decreasing the computational and memory overhead.

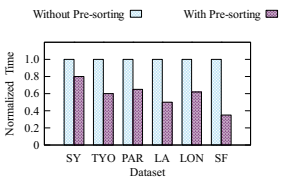



**Figure 9: Shortcut path matching time comparison.**

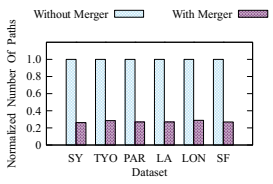



**Figure 10: Comparison of the number of actual paths.**

## 6.4 Planned Path Error Rate Evaluation

To demonstrate the effectiveness of TRAPP in path planning, we conduct experiments comparing the planned path error rate of TRAPP and Random. The planned path error rate is defined as the error rate between the path planned by the algorithm and the optimal path for a given path planning query $q$, *i.e.,*

$$\delta = (dist(p) - dist(p^*))/dist(p^*)$$

where $p$ is the planned path and $p^*$ is the optimal path for query $q$. In these experiments, Dijkstra's algorithm serves as the benchmark to obtain the feasible shortest path. For each dataset, we use the same query set across different methods and calculate the average planned path error rate of each individual query within the set, using this as the experimental result.

As shown in Figure 7, the planned path error rate of TRAPP is significantly lower than that of Random. In nearly all datasets, the average path distances planned by TRAPP are within 5% of the shortest path. This demonstrates the excellent path planning performance of TRAPP, allowing the majority of vehicles to get an acceptable path even when the optimal path is not found.

## 6.5 Comparison of Path Query Failure Rate

Path query failure occurs when a path query theoretically has at least one solution but cannot find one during practical execution. The path query failure rate is defined as the ratio of the number of failed queries to the total number of queries for a given query set $Q$. A higher path query failure rate indicates a less effective method. To validate the efficacy of our proposed approach, we conduct a comparative analysis of path query failure rates between TRAPP and Random in our experiment.

As shown in Figure 8, the failure rate of TRAPP is significantly lower than that of Random. In all the datasets, the failure rate of TRAPP is within 5%, and in the best cases, the failure rate of TRAPP can be below 1%. Our experimental results indicate that TRAPP can plan feasible paths for the vast majority of vehicles.

## 6.6 Shortcut Pre-sorting Evaluation

To verify the effectiveness of shortcut pre-sorting, we conduct an experiment to evaluate the matching time of shortcuts with setting $K = 30$. we compare the shortcuts matching time of the traditional matching method with that of our proposed matching method.

Figure 9 shows the normalized result of shortcuts matching time, our method reduces the time by 20% to 75% compared to the traditional method. This significant reduction demonstrates that the shortcut pre-sorting method can substantially improve the processing efficiency of path queries, thereby enhancing overall performance in practical applications.

## 6.7 Shortcut Merger Evaluation

We evaluated the efficiency of our proposed shortcut storage method by comparing its number of stored paths with that of the traditional shortcut storage method.

After normalization, as shown in Figure 10, across all datasets, our proposed method significantly reduces redundant actual path storage by 60% to 80% compared to the traditional method. This substantial reduction in memory consumption shows the effectiveness

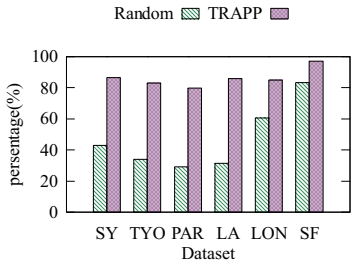



**Figure 11: Proportion of Optimal Paths.**

and efficiency of our proposed storage method, making it suitable for application in large-scale road networks.

## 6.8 Evaluation of the Proportion of Optimal Paths

To comprehensively evaluate our proposed method TRAPP, we conduct experiments on six different road networks to measure the optimal solution proportion of TRAPP. The optimal solution proportion is defined as the proportion of the number of queries that achieve the optimal solution to the total number of queries for a given query set $Q$. A higher proportion of optimal paths indicates more frequent planning of optimal solutions, thereby demonstrating superior path planning performance.

As shown in Figure 11, across different datasets, the optimal solution proportion of our proposed TRAPP method remains consistently high. In the best-case scenario, the optimal solution proportion reaches 96.8%, while even in the worst-case scenario, it remains at 79.6%. This demonstrates the excellent path planning effectiveness of TRAPP. Additionally, in all datasets, the optimal solution proportion of TRAPP is significantly higher than that of Random. For instance, in the SF dataset, the optimal solution proportion of TRAPP is nearly seven times higher than that of Random, despite both methods preserving a similar number of shortest paths. This clearly illustrates that our proposed method effectively filters out rarely used road restriction combinations and retains more frequently used ones, thereby ensuring excellent path planning performance with minimal computational and memory overhead.

## 6.9 Ablation Study

To verify the contribution of each component in TRAPP, we conduct experiments on the following variants of TRAPP.

- TRAPP **w/o PR**: This variant of TRAPP excludes the Partition-aware Refinement (PR) component.
- TRAPP **w/o CR+PR**: This variant of TRAPP excludes the Partition-aware Refinement (PR) and Combination Rematch (CR) components, essentially utilizing only the k-means algorithm.

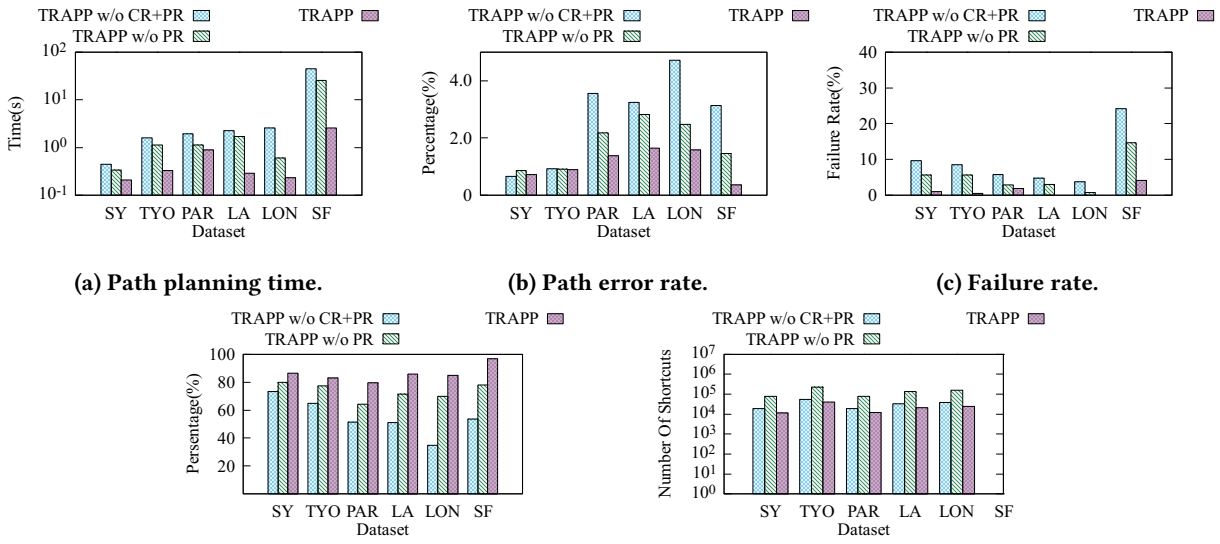
The ablation experiments evaluated the path planning time, path distance, failure rate, optimal solution ratio, and the number of preserved actual shortest paths for TRAPP, TRAPP w/o PR, and TRAPP w/o CR+PR across different datasets. Figure 12a shows that TRAPP consistently achieved the shortest path planning times across all datasets. This is because TRAPP has the lowest path planning failure rate among the three methods, resulting in fewer regressions to the Dijkstra algorithm. Figure 12b indicates that all three methods maintain low path planning errors when the optimal solution cannot be found. Figure 12c demonstrates that the complete TRAPP has the lowest failure rate, significantly lower than TRAPP w/o CR and TRAPP w/o CR+PR. Additionally, TRAPP w/o CR also has a significantly lower failure rate than TRAPP w/o CR+PR, highlighting the effectiveness of the CR and PR components. Figure 12d presents the experimental results of the optimal solution ratio for the three methods, showing that TRAPP has a significantly higher optimal solution ratio compared to TRAPP w/o CR and TRAPP w/o CR+PR, with the performance improving as more components are included. Figure 12e shows that TRAPP preserves the fewest actual shortest paths, while TRAPP w/o CR preserves the most, because the PR component retains more road restriction combinations. The experimental results also demonstrate that the PR component effectively controls the number of preserved road restriction combinations.

In summary, the experimental results of the ablation experiments demonstrate that the CR and PR components significantly enhance both the effectiveness and efficiency of TRAPP.

# 7 RELATED WORK

**Non-index-based Path Planning Algorithms**. Dijkstra's algorithm [10] is a classic shortest path algorithm that uses a greedy strategy to perform a full graph search to obtain the single-source shortest path on a non-negative weighted graph. Bellman-Ford algorithm [3, 12] is capable of solving the shortest path problem in graphs with negative edge weights. Johnson algorithm [19] is a combination of the previous two algorithms. Floyd-Warshall [11, 40] algorithm utilizes dynamic programming for path planning and A* search algorithm [15] incorporates heuristic methods to reduce unnecessary searches. [6] provided a theoretical summary of representative non-index-based algorithms and conducted an experimental evaluation.

**Index-based Path Planning Algorithms**. Graph indexing has been widely studied to enhance efficiency. Index-based methods typically involves pre-computing the shortest paths between some vertices and building indices to accelerate path planning. CH [13] and CRP [8] are the state-of-the-art path planning methods for road networks, where CH builds shortcuts for the neighbors of the contracted vertices in a specific order and CRP constructs a multi-level graph structure based on the given road network and builds shortcuts between the entry and exit vertices of each cell at each level. AH [49], which is similar to CH, leverages spatial information to construct a hierarchical structure and build multiple shortcuts within it. EDP [16] builds dynamic indices within the graph, enabling efficient edge-constrained shortest path queries on dynamic graphs. G*-tree [24] is an efficient hierarchical indexing structure that offers advantages such as good scalability. Xiao, Yanghua, et al. [43] design a novel path planning algorithm, which significantly reduces the indexing space consumption by exploiting the widespread symmetry in the graph, while ensuring the efficiency of path planning. H2H-index [33], which combines the advantages of 2-hop labeling and hierarchy, builds indices based on tree decomposition and maintains a hierarchical structure among all vertices in the graph. This method addresses the problem of large search spaces in hierarchical indexing structures for long-distance




**(a) Path planning time.** **(b) Path error rate.** **(c) Failure rate.**

**(d) Proportion of Optimal Paths.** **(e) Comparison of number of shortcut paths.**

**Figure 12: The efficiency of different optimizations.**

queries, while also mitigating the high computational overhead in hop-based indexing schemes for short-distance queries. Zhang Y, et al. [47] introduces the concept of relative subboundedness, studies the boundedness of CH and H2H, and proposes an incremental algorithm IncH2H, which can quickly update indices on dynamic road networks. Inspired by H2H, the LSD-index [46], which is based on tree decomposition, can perform label-constrained shortest path queries in a very short time while also reducing the memory consumption of the indices. Building upon the shortest-distance query processing methods PLL [1] and CTL [22], [45] extends their capabilities to efficiently handle shortest-path queries and introduces a novel path planning method, MLL, which achieves a significantly faster query speed. Some literature [23, 42] conduct evaluations on the performance of the representative path planning algorithms.

Real-world road networks have numerous different restrictions, yet existing algorithms do not take these into account. Therefore, it is crucial to study path planning algorithms that consider road network restrictions. TRAPP addresses this issue effectively, enabling the planning of passable, optimal, and acceptable paths for the majority of vehicles in road networks with restrictions, while significantly reducing the computational and storage overhead of the indices.

## 8 CONCLUSION

In this paper, we define the path planning problem under road restrictions and analyze the combinatorial explosion problem in existing methods applied to road networks with restrictions. Subsequently, we propose TRAPP, a path planning algorithm tailored for these networks. The core idea of TRAPP is to utilize traffic flow information within the road network to filter road restriction combinations, thereby addressing the substantial computational and memory overhead caused by the combinatorial explosion. Experimental results validate the effectiveness of TRAPP in path planning, as well as its low memory and computational overhead.

## REFERENCES


[1] 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 2013 ACM SIGMOD International Conference on Management of Data*. 349–360.

[2] Mohammed Al Eisaeia, Sara Moridpourb, and Richard Tay. 2017. Heavy vehicle management: restriction strategies. *Transportation Research Procedia* 21 (2017), 18–28.

[3] Richard Bellman. 1958. On a routing problem. *Quarterly of applied mathematics* 16, 1 (1958), 87–90.

[4] Franck Boizard, Bénédicte Buffin-Meyer, Julien Aligon, Olivier Teste, Joost P Schanstra, and Julie Klein. 2021. PRYNT: a tool for prioritization of disease candidates from proteomics data using a combination of shortest-path and random walk algorithms. *Scientific Reports* 11, 1 (2021), 5764.

[5] Zi Chen, Bo Feng, Long Yuan, Xuemin Lin, and Liping Wang. 2023. Fully Dynamic Contraction Hierarchies with Label Restrictions on Road Networks. *Data Science and Engineering* 8, 3 (2023), 263–278.

[6] Boris V Cherkassky, Andrew V Goldberg, and Tomasz Radzik. 1996. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming* 73, 2 (1996), 129–174.

[7] Spokane County. 2024. *Road Restrictions & Weight Enforcement Information.* Retrieved February 25, 2024 from https://www.spokanecounty.org/3745/Road-Restrictions-Weight-Enforcement-Inf

[8] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. 2017. Customizable Route Planning in Road Networks. *Transp. Sci.* 51, 2 (2017), 566–591. https://doi.org/10.1287/TRSC.2014.0579

[9] Daniel Delling, Andrew V Goldberg, Ilya Razenshteyn, and Renato F Werneck. 2011. Graph partitioning with natural cuts. In *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 1135–1146.

[10] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1 (1959), 269–271. https://doi.org/10.1007/BF01386390

[11] Robert W Floyd. 1962. Algorithm 97: shortest path. *Commun. ACM* 5, 6 (1962), 345–345.

[12] Lester Randolph Ford. 1956. Network flow theory. *Rand Corporation Paper, Santa Monica, 1956* (1956).

[13] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Experimental Algorithms, 7th International Workshop, WEA 2008,*

*Provincetown, MA, USA, May 30-June 1, 2008, Proceedings (Lecture Notes in Computer Science)*, Catherine C. McGeoch (Ed.), Vol. 5038. Springer, 319–333. https://doi.org/10.1007/978-3-540-68552-4_24

[14] Chengyang Han and Baoying Li. 2023. Mobile robot path planning based on improved A* algorithm. In *2023 IEEE 11th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Vol. 11. IEEE, 672–676.

[15] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[16] Mohamed S Hassan, Walid G Aref, and Ahmed M Aly. 2016. Graph indexing for shortest-path finding over dynamic sub-graphs. In *Proceedings of the 2016 International Conference on Management of Data*. 1183–1197.

[17] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. 2010. Engineering a scalable high quality graph partitioner. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 1–12.

[18] Wenbin Hou, Zhihua Xiong, Changsheng Wang, and Howard Chen. 2022. Enhanced ant colony algorithm with communication mechanism for mobile robot path planning. *Robotics and Autonomous Systems* 148 (2022), 103949.

[19] Donald B Johnson. 1977. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)* 24, 1 (1977), 1–13.

[20] Sungwon Jung. 2002. An efficient path computation model for hierarchically structured topographical road maps. *IEEE transactions on knowledge and data engineering* 14, 5 (2002), 1029–1046.

[21] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[22] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2020. Scaling up distance labeling on graphs with core-periphery properties. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1367–1381.

[23] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An experimental study on hub labeling based shortest path algorithms. *Proceedings of the VLDB Endowment* 11, 4 (2017), 445–457.

[24] Zijian Li, Lei Chen, and Yue Wang. 2019. G*-tree: An efficient spatial index on road networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 268–279.

[25] Moeen Meigooni and Emad Tajkhorshid. 2021. Waypoint Graph Generation with Growing Neural Gases for Pathfinding Applications in Molecular Dynamics Simulations. *Biophysical Journal* 120, 3 (2021), 78a.

[26] Changwei Miao, Guangzhu Chen, Chengliang Yan, and Yuanyuan Wu. 2021. Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm. *Computers & Industrial Engineering* 156 (2021), 107230.

[27] Government of Alberta. 2024. *Road restrictions and bans – Overview*. Retrieved February 25, 2024 from https://www.alberta.ca/road-restrictions-and-bans-overview

[28] The People's Government of Beijing municipality. 2024. *Requirements for Setting Load Limit Signs on Highway Bridges*. Retrieved September 26, 2024 from https://www.beijing.gov.cn/zhengce/zhengcefagui/qtwj/202204/t20220414_2677142.html

[29] Anhui Provincial Department of Housing and Urban-Rural Development. 2024. *Urban Bridge Load Limit Standards*. Retrieved September 26, 2024 from http://dohurd.ah.gov.cn/public/6991/53904301.html

[30] Ministry of Industry and Information Technology of the People's Republic of China. 2024. *GB1589-2016*. Retrieved October 24, 2024 from https://www.miit.gov.cn/xwdt/gxdt/sjdt/art/2020/art_eff1b0b6bdcd428d969d3bce5c102b19.html

[31] Ministry of Transport of the People's Republic of China. 2024. *Regulations on Road Safety and Protection*. Retrieved September 26, 2024 from https://www.mot.gov.cn/zhengcejiedu/gongluanquanbhtl/xiangguanzhengce/201510/t20151015_1905301.html

[32] Ministry of Transport of the People's Republic of China. 2024. *Traffic Overview*. Retrieved September 26, 2024 from https://www.mot.gov.cn/jiaotonggaikuang/201804/t20180404_3006639.html

[33] 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 2018 International Conference on Management of Data*. 709–724.

[34] François Pellegrini and Jean Roman. 1996. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking: International Conference and Exhibition HPCN EUROPE 1996 Brussels, Belgium, April 15–19, 1996 Proceedings 4*. Springer, 493–498.

[35] Yuanfang Ren, Ahmet Ay, and Tamer Kahveci. 2018. Shortest path counting in probabilistic biological networks. *BMC bioinformatics* 19 (2018), 1–19.

[36] Michael N. Rice and Vassilis J. Tsotras. 2010. Graph Indexing of Road Networks for Shortest Path Queries with Label Restrictions. *Proc. VLDB Endow.* 4, 2 (2010), 69–80. https://doi.org/10.14778/1921071.1921074

[37] Jose Ricardo Sanchez-Ibanez, Carlos J Pérez-del Pulgar, and Alfonso García-Cerezo. 2021. Path planning for autonomous mobile robots: A review. *Sensors* 21, 23 (2021), 7898.

[38] Peter Sanders and Dominik Schultes. 2005. Highway hierarchies hasten exact shortest path queries. In *European Symposium on Algorithms*. Springer, 568–579.

[39] Kun Tan, Qian Zhang, and Wenwu Zhu. 2003. Shortest path routing in partially connected ad hoc networks. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, Vol. 2. IEEE, 1038–1042.

[40] Stephen Warshall. 1962. A theorem on boolean matrices. *Journal of the ACM (JACM)* 9, 1 (1962), 11–12.

[41] Fang Wei. 2010. TEDI: efficient shortest path query answering on graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 99–110.

[42] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest path and distance queries on road networks: an experimental evaluation. *Proceedings of the VLDB Endowment* 5, 5 (2012), 406–417.

[43] Yanghua Xiao, Wentao Wu, Jian Pei, Wei Wang, and Zhenying He. 2009. Efficiently indexing shortest paths by exploiting symmetry in graphs. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. 493–504.

[44] Peng Xu, Qian Wu, Deyang Lu, Jian Yu, Yongsheng Rao, Zheng Kou, Gang Fang, Wenbin Liu, and Henry Han. 2020. A systematic study of critical miRNAs on cells proliferation and apoptosis by the shortest path. *BMC bioinformatics* 21 (2020), 1–14.

[45] Junhua Zhang, Wentao Li, Long Yuan, Lu Qin, Ying Zhang, and Lijun Chang. 2022. Shortest-path queries on complex networks: experiments, analyses, and improvement. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2640–2652.

[46] U Zhang, Long Yuan, Wentao Li, Lu Qin, and Ying Zhang. 2021. Efficient label-constrained shortest path queries on road networks: A tree decomposition approach. *Proceedings of the VLDB Endowment* (2021).

[47] 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.

[48] Xunyu Zhong, Jun Tian, Huosheng Hu, and Xiafu Peng. 2020. Hybrid path planning based on safe A* algorithm and adaptive window approach for mobile robot in large-scale dynamic environment. *Journal of Intelligent & Robotic Systems* 99, 1 (2020), 65–77.

[49] 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 2013 ACM SIGMOD International Conference on Management of Data*. 857–868.

[50] Xiaoyan Zhu, Alberto Garcia-Diaz, Mingzhou Jin, and Ying Zhang. 2014. Vehicle fuel consumption minimization in routing over-dimensioned and overweight trucks in capacitated transportation networks. *Journal of Cleaner Production* 85 (2014), 331–336.