Efficient Estimation of Time-Dependent Shortest Paths Based on Shortcuts

Linbo Liao[1], Shipeng Yang[1], Yongxuan Lai[1(✉)], Wenhua Zeng[1], Fan Yang[2], and Min Jiang[1]

[1] School of Informatics/Shenzhen Research Institute, Xiamen University, Xiamen 361005, China
{laiyx,whzeng,minjiang}@xmu.edu.cn
[2] School of Aerospace Engineering, Xiamen University, Xiamen 361005, China
yang@xmu.edu.cn

Abstract. The shortest path search in the road network in the road network is of great importance in various Intelligent Transportation Systems. However, the commonly used shortest path search algorithms, such as Dijkstra and A * algorithm, are time-consuming due to their complexity, which leads to their poor performance in large-scale road networks. Thus, a new optimization technology is required to solve the path search problem on large-scale road networks. In this paper, the temporal feature of the road network is considered for the shortest path search problem, which is closer to the road network of the real world. And, an algorithm called *Time-Dependent A* With Shortcuts (TDAWS)* is proposed to estimate the time-dependent shortest paths. Concretely, the road network is pre-processed offline and partitioned into several regions based on clustering, which captures the spatial pattern of the road network. Then we construct shortcuts contain the shortest paths information to reduce search time and propose two mechanisms called Hop On Directionally (HOD) and Hop-Off Early (HOE) to avoid unnecessary detours. We constructed an extensive experimental study on a road network with real-world taxi trajectory data and compared our approach with existing techniques. The results demonstrated that the time cost of our method is more stable and achieves up to 17 times faster than the precise shortest path searching algorithm with an acceptable extra ratio (about 20%) on the path length.

Keywords: Approximate shortest path · Graph partition · Path estimation · Time-dependent road network

This work was supported in part by the Natural Science Foundation of China under Grant 6187215 and Natural Science Foundation of Guandong under Grant 2021A1515011578, and Shenzhen Basic Research Program under Grant JCYJ20190809161603551.

Y. Lai et al. (Eds.): ICA3PP 2021, LNCS 13156, pp. 18–32, 2022.
https://doi.org/10.1007/978-3-030-95388-1_2

1 Introduction

With the development of intelligent transportation, many car-hailing systems and navigation systems have emerged in recent years [13]. Those systems process millions of path-planning or time-estimations tasks every day, within which the shortest path search is a fundamental block. The shortest path computation must be efficient to respond in a real-time way [16]. However, the shortest path searching algorithms are time-consuming due to the complexity of the task [22]. Besides, time-dependent shortest path search, which considers time-dependent factors like traffic jams, becomes a general search method and increases the computing time.

The well-known Dijkstra algorithm [5] performs well on static graphs and can be extended to the time-dependent case straightforwardly [6]. However, there are lots of repeated computations when processing shortest path search on the same road network. Moreover, the A^* algorithm [12] is a goal-directed search and is guaranteed to explore no more nodes than Dijkstra's algorithm. In the A^* algorithm, all the computed information is cached to accelerate the search, which requires a huge memory space and makes it impractical.

Road networks have some unique features that distinguish them from other common graph structures. First, the road network is usually a hierarchical network, in which the high-grade highway has greater capacity and higher driving speed than the low-grade highway. Second, high-grade highways can span and link different regions, while low-grade highways are scattered in each region. Therefore, when planning a trip from one area to another, people often choose the main road first and then the minor road leading to the main road. In this way, we can reduce the complexity of the path search process. Besides, some scenarios like the travel time estimation, in which only an estimated cost of the travel time needed, can avoid the detailed exact shortest path calculation [2]. Therefore, we need to optimize the classical shortest path algorithms such as the Dijkstra algorithm [5] to solve the time-dependent shortest path problem.

In this paper, we study the problem of travel time estimation for large-scale time-varying road networks. The challenges lie in several aspects: 1) the search time is positively correlated with the distance between origin and destination, which results in a big difference in the time costs between long-distance and short-distance path searching; 2) the speeds of the road segments change with time, which increases the uncertainty of path searching. In this paper, the idea *shortcut* [7] is applied to quickly guide the search process from the start region to the end region. And we propose an efficient algorithm that estimates the shortest path on time-dependent road networks based on the shortcut mechanism. The major contributions of this paper are as follows:

- We proposed a bidirectional partition method that combines with the K-Means clustering [17] to partition a road network into regions, where shortcuts are inserted at their best positions.
- We proposed a path searching algorithm to estimate the shortest paths efficiently. Different from existing research, we consider the shortest path search

problem within the scenario of large-scale time-dependent road networks. And our approach adopts bidirectional partitions and shortcuts to make the searching time-efficient and stable.
– We conducted extensive experiments on a real road network with real request datasets. The results demonstrated that our approach achieves up to 17 times faster than the precise shortest path searching algorithm with an acceptable deviation on the path length. Besides, the searching time is stable regardless of the distance from the origin to the destination.

The rest of this paper is organized as follows. Section 2 presents the related work of this paper. Section 3 gives some preliminaries and problem definitions. Section 4 presents the details of the graph partition algorithm that combines K-Means and the construction of shortcuts. Section 5 presents the detailed description of the approximating shortest path algorithm based on shortcut mechanism. Section 6 presents the implementation details of hop on directionally and hop off early. Section 7 presents the experimental studies and analysis. Finally, Sect. 8 concludes the paper and presents some future directions.

2 Related Work

There are lots of speed-up techniques proposed for path-finding algorithms during the past several decades [4,21]. This section mainly introduces two important algorithms related to road networks: 1) algorithm based on hierarchy in road network; 2) graph-based search space trimming algorithm.

A lot of works use the hierarchical structure of the road network to accelerate the search. Shapiro et al. [19] proposed a heuristic algorithm that uses the "level" of nodes and edges to generate approximate shortest paths rapidly. This algorithm can obtain the near-optimal path by searching a few nodes. Chou et al. [2] partitioned a large-scale network into a high-level subnetwork and a set of lower-level subnetworks. The partitioning permits a trade-off between pre-computation and query time processing. Then, they proposed a hierarchical algorithm to approximate the shortest paths in large-scale networks. A traffic mining approach had been proposed by Gonzalez et al. [10]. They used the road hierarchy and pre-computed areas to limit the search space. Delling and Nannicini [3] noted the timing of the road network and proposed a hierarchical approach called *core routing* that combined with bidirectional goal-directed search in time-dependent road networks. This algorithm is flexible and suitable for a dynamic scenario where the piecewise linear time-dependent cost functions on unfixed arcs. But the algorithm did not reduce the search time compared with the shortest path estimation algorithm while it has a complex preprocessing.

The road network contains a lot of road information, and a lot of work uses the pre-computed information of the road network to prune the search space. Wagner and Willhalm [20] pruned the Dijkstra's search by using geometric containers with precomputed information. Experiments show that the search space for online computation reduced significantly.Lauther, U. I. [14,15] proposed a

variant of Dijkstra's algorithm by using edge flags to prune the search space in static networks. They partitioned the road network into geometrically connected regions and introduced the concept of edge flags: the flag is set if there is a shortest path over it into the target region. Möhring et al. [18] also introduced an algorithm that uses edge flags and particularly examined different partitioning algorithms and compared their impact on the speed-up of the shortest-path algorithm. They found that the bidirectional search is the best partition-based speed-up method among the methods they tested. Gutman [11] defined the vertex reach and pruned a path search by using vertex reaches. It easily handles multiple origins and destinations. Goldberg et al. [8,9] improved the reach-based approach of Gutman [11] and introduced a practical algorithm that combines A^* search, landmark-based lower bounds, and reach-based pruning. Through the reach-aware landmarks and improved algorithm, the preprocessing and queries are faster while the overall space consumption is reduced.

Most existing works use a hierarchical structure or preprocess the road network to speed up the search or reduce the search scope. However, these algorithms consume a lot of time as the request distance increases. We consider the directed large-scale time-dependent road network and propose an efficient shortest path estimation algorithm based on road network partitions and shortcuts which has a stable search efficiency regardless of the distances of requests.

3 Problem Definition

3.1 Time-Dependent Road Network

Let a strongly connected directed graph $G = (V, E)$ be the road network with a set of nodes V and a set of edges $E \subseteq V \times V$. The edges of a graph are weighted by a function $w : (E, t_0) \rightarrow \mathbb{R}$ where t_0 is time.

The weight of an edge $e(u, v) \in E$ at time t_0 is the traveling time from u to v instead of distances; we denote the weight of edge $e(u, v)$ at time t_0 as $w(e(u, v), t_0)$. Given a path from o to d can be denote as $p(o, d, t_0) = (v_0 = o, ..., v_i, ...v_j, ..., v_k = o)$, where t_0 is the departure time. Then the weight of path p is defined as $w(p) = w_0(e(v_0, v_1), t_0) + w_1(e(v_1, v_2), t_1) + ... + w_{k-1}(e(v_{k-1}, v_k), t_{k-1})$, where $t_{i+1} = t_i + w_i(e(v_i, v_{i+1}), t_i)$. The path that has the minimum weight from o to d with a departure time t_0 is called the shortest path which is denoted by $p_s(o, d, t_0)$. We construct an edge $e(o, d)$ with weight $w(e(o, d), t_0) = w(p_s(o, d, t_0))$, which is called the *shortcut* from o to d. As shown in Fig. 1, the dotted line represents the shortcut from A to E, its weight is the minimum path weight from A to E. In the following, we represent the path as $p(o, d)$ when the departure time t_0 is known as context.

3.2 Partitions of Road Network

Given a strongly connected directed graph $G = (V, E)$, we denote the partition of G as $P_m (1 \leq m \leq k)$, where k is the total number of the partitions. The

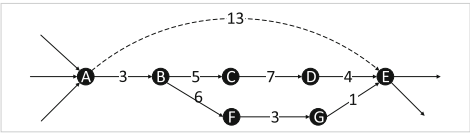


Fig. 1. A shortcut bypassing several nodes: the dotted line is the shortcut from node A to E, which represents the shortest path $p(A, E) = (A, B, F, G, E)$.

$V_{P_m}(1 \leq m \leq k)$ is the set of vertices in P_m and $V_{P_m} \cap V_{P_n} = \emptyset(m \neq n$ and $1 \leq m, n \leq k)$. Each partition has a representative node $C_m(1 \leq m \leq k)$, which is called *core node*.

For any node x in the graph G, the index of its partition can be expressed as i(x). So the partition that x belongs to can be denoted by $P_{i(x)}$ and the corresponding core node is $C_{i(x)}$. For simplicity, we replace them with P_x and C_x in the following.

4 Graph Partitioning and Shortcuts Construction

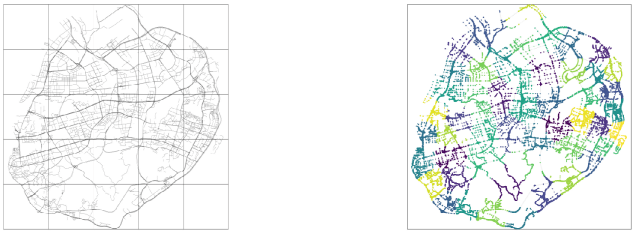
The preprocessing that creates additional information can reduce the search space. More precisely, our approach uses regions to constrain the search space and uses shortcuts to accelerate the path search algorithm. The weights of paths greatly depend on how the road network is partitioned and how core nodes are selected, as they severely affect the quality of shortcuts. In this section, we introduce the preprocessing of the time-dependent road network: graph partitioning and shortcut construction.

The main idea of our algorithm is to use shortcuts between regions to reduce the search space and accelerate the search. Considering the consumption of storage space, we only establish the shortcut between core nodes of each region. Partition and core node acquisition can be divided into two steps: 1) select the core nodes by the K-Means clustering-based algorithm; 2) generate the regions according to the core nodes. Then, a modified $A*$ algorithm is applied to accelerate the estimate of the time-dependent shortest path. The algorithm accelerates the searching by shortcuts when the origin and destination are not in the same region. Besides, two measures are adopted to avoid excessive detours. One can lead the algorithm to "hop on" a better shortcut and the other can help the algorithm "hop off" the shortcut more smoothly.

4.1 Bidirectional Partitioning

For a 2D layout of the graph, a common way is grid partition, where nodes in the same grid are divided into a partition. Figure 2(a) shows an example of grid partition, which can partition a graph into different regions fast in an easy way. However, this method ignores other properties like the density of nodes. Another partition method is based on the K-Means clustering algorithm, which divides

the sample set into k clusters according to the distance. The criterion of K-means is to reduce the distance between nodes within a cluster while increasing the distance between clusters. Figure 2(b) shows an example of the K-Means partition. However, the K-Means partition is not very suitable for road networks. First, longitude and latitude, used in distance calculations, can not represent the actual distance between two nodes in a road network. As the nodes are linked by several road segments, which have different speed limits and lengths. Second, the road network can be seen as a directed graph. For example, a car may go through different road segments when it goes from node a to node b and node b to node a.



(a) Grid partition (b) K-Means partition

Fig. 2. Two partition methods of Xiamen island

To generate the partitions considering the bidirectional distances between nodes, we process as follows:

1. First, we select core nodes based on K-means in two steps: 1) Obtain K partitions by the K-Means clustering algorithm; 2) the central node of each partition is identified as the core node, which has the shortest distance sum to other nodes in the same region. Then, we adopt the Distance-First Partitioning on graph G to assign nodes to suitable partitions in turn, whose pseudocode is shown in Algorithm 1. In the first part of the algorithm, the cluster centers(core nodes) are generated by K-Means algorithm (line 1). Then, the *candidateSet* is initialized to a set of all nodes except the core nodes (line 2). Queues are created for each core node to store the neighbor nodes in ascending order of distance to the core node (line 3–6). In the second part, we select partitions for each node in turn. For each partition, we extract a node from the queue and add it into the partition if it is in the candidate set. Then, we add the neighbor nodes of the node to the queue in order (line 14–18). Each partition performs the same steps in turn.
2. We also adapt the *Distance-First Partitioning* algorithm on the reversed graph[1] G^{-1} to represent the distance relationship between nodes and core nodes more comprehensively. The partition and core node in G^{-1} are denoted by P_m^{-1} and C_m^{-1} $(1 \leq m \leq k)$ respectively.

[1] G^{-1} has the same vertex set as G but all edges reversed.

Algorithm 1: DistanceFirstPartition(G, k)

input : A graph G, number of clusters k
output: A partitioned graph

```
Obtain the core nodes N_core of K partitions according to the K-Means clustering algorithm;
Initialize the candidate set S_c as the node in the graph except the core nodes;
Initialize empty queues Q for each partition;
for i in k do
    Otain the neighbor nodes of N_core[i];
    Add the sorted neighbor nodes into Q[i] by distance in ascending order;
while S_c is not empty do
    for i in k do
        while Q[i] is not empty do
            currentNode = Q[i].pop();
            if currentNode in S_c then
                Add the currentNode into the partition of core node N_core[i];
                Remove the currentNode from the candidate set S_c;
                Otain the neighbor nodes N_cur of the currentNode;
                for node in N_cur do
                    if node in S_c then
                        Calculate the distance from the node to the core node;
                        Add the node to Q[i] and sort Q[i];
                break;
return G;
```

We call this algorithm *Bidirectional Partitioning* because it considers the bidirectional distances between nodes and obtained two partitioning schemes according to the directions of edges.

4.2 Add Shortcuts

To reduce the complexity of searching the shortest path, we created a shortcut for each pair of core nodes: additional edges with time-dependent weight equal to the original shortest path between their endpoints. And we use $sc(u, v)$ to denote the shortcut from node u to node v. Then, we divide a day evenly into x time slices for that the time-dependent weight of the path is dynamic. Then we calculate the shortcut between each core node pair in each time slice and store them in the memory. It is obvious that the accuracy and storage space are increase with the time slices x.

5 Time-Dependent A^* with Shortcuts

In this section, we introduce a fast approach for estimating the shortest paths of a node pair. Our algorithm can reduce the search time significantly while

Algorithm 2: TDAWS(G, o, d, t_0)

input : A preprocessed graph G, the origin node o, the destination node d, the start time t_0
output: A approximate shortest path from o to d

```
if G.C_o^{-1} == G.C_d then
    p = TDRA(G, o, d, t_0);
    return p ;
else
    p_1 = TDRA(G, o, C_o^{-1}, t_0);
    p_2 = C_o^{-1}.shortcut(C_d, t_0 + w(p_1));
    p_3 = TDRA(G, C_d, d, t_0 + w(p_1) + w(p_2));
    return p_1 + p_2 + p_3;
```

the weight of the result is nearly optimal. The main idea is to use the precomputed information, partitions and shortcuts, to narrow the search and omit the calculation.

Given a graph $G = (V, E)$ that has been processed by *Bidirectional Partitioning* algorithm and source and destination vertices $o, d \in V$, the algorithm for estimating the shortest time-dependent path works as follows:

(a) If $C_o^{-1} = C_d$, then start a restrained time-dependent $A*$ search from o on G and the search scope of nodes is restrained in $V_{P_o^{-1}} \cup V_{P_d}$.
(b) If $C_o^{-1} \neq C_d$, the process of estimating shortest time-dependent path from o to d can be decomposed into four steps:
step 1. A Time-Dependent Restrained $A*$(TDRA) occurs on G from o to C_o^{-1} and the search scope of nodes is restrained in $V_{P_o^{-1}}$. The path obtained in this step is denoted by p_1. TDRA is a Time-Dependent A^*(TDA) with limited search scope.
step 2. The shortcut from C_o^{-1} to C_d provides the shortest time-dependent path p_2 from C_o^{-1} to C_d.
step 3. A TDRA occurs on G like step 1 while the source and target are C_d and d and the search scope of nodes is restrained in V_{P_d}. The path obtained in this step is denoted by p_3.
step 4. The final estimated shortest time-dependent path is the combination of p_1, p_2 and p_3.

We call this algorithm Time-Dependent A^* With Shortcuts (TDAWS). The Algorithm 2 presents the pseudocode of TDAWS.

6 Avoid Detours

Although TDAWS can effectively reduce the cost of computation time, it can also cause some unnecessary detours. This section describes two mechanisms to avoid detours: hops on the shortcut and hops off the shortcut.

6.1 Hop on Directionally

Given a graph $G = (V, E)$ that has been processed by *Bidirectional Partitioning* algorithm. The origin and destination nodes $(o, d \in V)$ are not in the same partition, i.e. $C_o^{-1} \neq C_d$. As shown in Fig. 3(a), m and n are the core nodes of o and d, respectively. According to TDAWS, the searching hops on the shortcut between m and n although there is a detour from o to m. Hence, we propose the Hop On Directionally (HOD) mechanism to avoid detours with the help of adjacent core node h of m. Hence, we propose the Hop On Directionally (HOD) mechanism to avoid detours with the help of adjacent core node k of m. Specifically, the HOD is described as follows:

1. Select X nodes from the adjacent core nodes of m by comparing the weight of shortcut to n as the candidate nodes.
2. Calculate the time cost from o to each candidate node. We chose the path with the lowest total time cost to replace the path obtained in Step 1 of TDAWS.
3. Then the TDAWS progresses as previously defined until the target is settled.

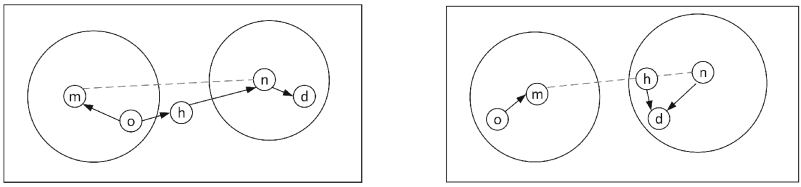
6.2 Hop Off Early

Given a graph $G = (V, E)$ that has been processed by *Bidirectional Partitioning* algorithm and source and destination vertices $o, d \in V$ and $C_o^{-1} \neq C_t$. As shown in Fig. 3(b), m is C_o^{-1}, n is C_d and h is a node on the shortcut. As we can see, the shortcut between m and n goes too "far" and the searching has to take a detour to get d. If the searching hops off the shortcut at k, the detour can be avoided. So we proposed a mechanism called *Hop Off Early(HOE)* to stop the TDAWS step 2 before it makes more detours. The HOE is as follows:

1. The TDAWS step 1 progresses as defined.
2. As the TDAWS progresses to step 2, the p_2 is obtained. Then traversal p_2 forward until it reaches the node that is in V_{P_d} and the node is denoted by h. The p_2 only keeps the path from the start point of p_2 to h and abandons the rest, which means the TDAWS hops off the shortcut early at node h.
3. A TDA occurs on G and the origin and destination are h and d. The path obtained in this step replaces the path obtained in TDAWS step 3 and we still denote it by p_3.
4. Then the TDAWS returns the combination of p_1, p_2 and p_3.

7 Experiments

We conduct experiments with real-world road network and taxi trajectory datasets to verify the performance of the proposed algorithm. The schemes are implemented in Java 1.8 and experiments are run on a desktop computer with AMD R9 3900XT CPU, 3.8 GHz, 64G RAM under Ubuntu 18.04.5 LTS.



(a) HOD (b) HOE

Fig. 3. Two mechanisms for avoiding detours, m is C_o^{-1}, n is C_d, h is a core node and the dotted lines are shortcuts

The road network of the Xiamen island is used for the simulation, which contains 24,750 road nodes and 3,234 road segments. The source file of map ([118.0660E,118.1980E] × [24.4240N,24.5600N]) is from OpenStreetMap[2] and the road network is based on the JGraphT[3] framework.

The default simulation settings are set as follows unless otherwise stated. The Xiamen Island is divided into 80 partitions by the K-Means clustering algorithm. And a day is divided into 24 time slices to represent the time-dependent properties of the road network.

7.1 Compared Algorithms

To study the performance, we compared the efficiency of algorithms that also use modified A^* algorithm on dynamic road network. We conduct the following algorithms:

- Time Dependent A*(TDA): the time-dependent case of $A*$, whose potential function uses Euclidean distance and an upper bound speed to estimate the weight between a node and the target. It serves as the baseline of the schemes.
- Time Dependent A* With Shortcuts (TDAWS): the modified time-dependent case of $A*$ using shortcuts to accelerate the searching process.
- Time Dependent A* With Shortcuts And Hop on Directionally (TDAWS+HOD): the TDAWS algorithm with Hop on Directionally mechanism.
- Time Dependent A* With Shortcuts And Hop off Early (TDAWS+HOE): the TDAWS algorithm with Hop off Early mechanism.
- Time Dependent A* With Shortcuts, Hop on Directionally And Hop off Early (TDAWS+HOD+HOE): the TDAWS algorithm with Hop on Directionally and Hop off Early mechanism.
- K Shortest Path Algorithm Based on Lifelong Planning A* Technique(KSP-LPA*): is adopted based on [1] , which formulates the deviation path calculation process as repeated one-to-one searches for the shortest path in a dynamic network.

[2] https://www.openstreetmap.org/.
[3] https://jgrapht.org/.

7.2 Result Analysis

Compared with the TDA, we do some offline preprocessing on the road network before the shortest path searching, including 1) clustering the graph with K-Means to obtain the partitions; 2) create shortcuts for each core node pair. The consumption of graph partition can be ignored as it only be executed once for a graph. The cost of building a shortcut is related to time slices and the number of partitions. It takes 24.562s to construct the shortcut for each time slice when the number of partitions is 80. And storing a day's shortcut consumes about 12MB when the number of time slices is 24.

Table 1. Overall performance of the schemes.

Schemes	Metrics			
	Mean search time (ms)	Standard deviation of search time	Shortcuts hit rate (%)	CR
TDA	4.272	32.235	–	1
TDAWS	0.123	0.503	87.4	1.44
TDAWS+HOD	0.150	0.450	83.7	1.33
TDAWS+HOE	0.263	0.726	87.4	1.29
TDAWS+HOD+ HOE	0.243	0.603	83.7	1.20
KSP-LPA*	2.44	1.681	–	3.28

Overall Performance. We evaluate the performances of TDA, TDAWS, TDAWS+HOD, TDAWS+HOE, TDAWS+HOD+HOE, and KSP-LPA* with 19,101 origin-destination pairs. We use a competitive ratio to evaluate the quality of an approximate time-dependent shortest path, which is defined as follows:

$$CR = \frac{w(p_a)}{w(p_{min})} \tag{1}$$

where p_a and p_{min} are the search path and the shortest path from the source node to the target node, respectively. When the $CR \in [1, \infty)$ is smaller, the search result is closer to the shortest path.

As shown in Table 1, the CR of TDA is the lowest (1.00) as its search result is the shortest path with the highest search time (4.272 ms) and standard deviation. The search time of TDAWS is the lowest (0.123 ms) with the cost of about 44% longer estimated path than TDA. As shown in the result of TDAWS+HOD and TDAWS+HOE, HOD and HOE mechanisms both can reduce the CR compared with TDAWS. However, the HOD mechanism will reduce the shortcut hit rate while the HOE mechanism will double the search time of TDAWS. When both HOD and HOE mechanisms are adopted, the estimated path quality (CR = 1.20) is further improved while it still costs more query time (0.243 ms) than TDAWS. The KSP-LPA* has about 43% of query timeless (2.44 ms) than TDA while its CR is 3.28, which means the estimated path weight is 3.28 times the shortest

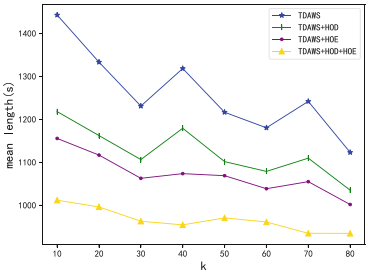


Fig. 4. Impact of the number of partitions on the mean path length

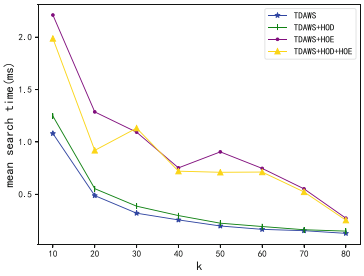


Fig. 5. Impact of the number of partitions on the mean search time

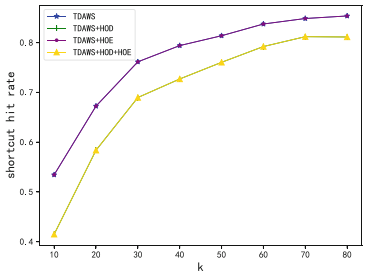


Fig. 6. Impact of the number of partitions on shortcut hit rate

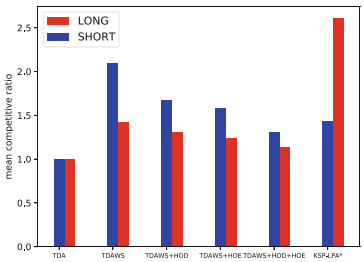


Fig. 7. Impact of the distances of requests on the mean competitive ratio

path weight. As for the stability of the algorithms, the algorithms based on shortcuts have very small standard deviations, which means for any shortest path request, they can respond in a stable time.

Impact of the Number of Partitions. Figure 4 shows the experimental results of the mean path length with different numbers of partitions, k, from 10 to 80. As shown in Fig. 4, when the number of partitions increases, the mean path length of the results achieved by all the approaches shows a trend of downward. The reason is that when the road network is divided into more partitions, the distances between core nodes and other nodes are decreasing, and the estimated path contains fewer detours. When k is 40 and 70, the values of mean length of TDAWS and TDAWS+HOD rise because the locations of some core nodes may cause more detours, while the HOE can effectively correct the detours. The influence of the number of partitions on TDAWS+HOD+HOE is small, which shows that TDAWS+HOD+HOE can effectively avoid detours caused by the locations of core nodes. Figure 5 illustrates the impact of the number of partitions on mean search time. The mean search times of all the algorithms show a downward trend as k increases. While the TDAWS has the best result, TDAWS+HOE and TDAWS+HOD+HOE have close results.

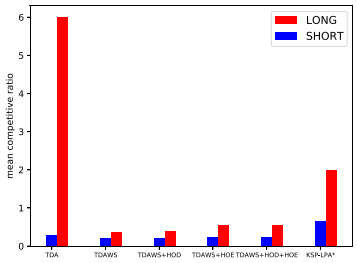


Fig. 8. Impact of the distances of requests on the mean search time

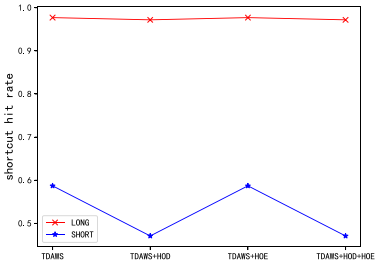


Fig. 9. Impact of the distances of requests on the shortcut hit rate

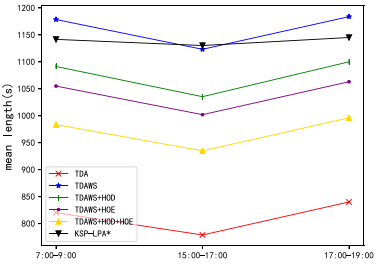


Fig. 10. Impact of traffic time on the mean path length

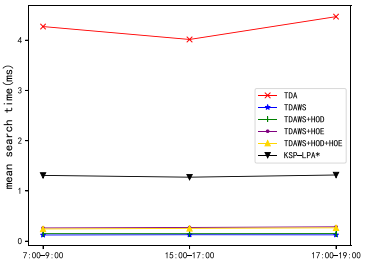


Fig. 11. Impact of traffic on the mean search time

From above, we can infer that the HOD in the AWS+HOD+HOE is not the the main factor for the time-consuming while the HOE is. As we can see from Fig. 6, the shortcut hit rates of all the algorithms decrease as k increases. The curves of TDAWS and TDAWS+HOE overlap, so as the TDAWS+HOD and TDAWS+HOD+HOE. This is because the HOD mechanism can affect the shortcut hit rate.

Impact of the Distances of Requests. To evaluate the performance of the proposed algorithms for different distance requests, We divided requests into two categories: the short-distance request and the long-distance request. Requests are classified according to the Euclidean distance between the origin and destination. If the distance is less than 3 km, it is called a *short-distance request*, otherwise, it is a *long-distance request*.

We conducted experiments with short-distance requests and long-distance requests, respectively. As shown in Fig. 7, The CR of KSP-LPA* increase sharply when the distance exceeds 3KM. In contrast, TDAWS and its deformations are not affected by the length of the distance, which takes advantage of the shortcut. Figure 8 illustrates the impact of the distances of requests on the mean search time. It can find that the search time of KSP-LPA* is significantly influenced while TDAWS and its deformations are stable when the distance increased.

This means that the algorithms based on shortcuts have stable search efficiency regardless of the distances of requests. This is because TDAWS has a higher shortcut hit rate in long-distance requests, which is also reflected in Fig. 9.

Impact of Traffic Time. We also verify the performance of the algorithms in different traffic periods, including morning peak time (7:00–9:00), evening peak time (17:00–19:00), and normal time (15:00–17:00). Figure 10 illustrates the impact of traffic time on the mean path length. It can be found that the estimated path weight in the peak period is higher, which is due to road congestion. As shown in Fig. 11, the traffic time has little impact on the algorithms that use shortcuts while the TDA and KSP-LPA* are more affected. This is because TDA and KSP-LPA require more resources to handle increased requests during peak times.

8 Conclusion

We have proposed an efficient algorithm for estimating the time-dependent shortest path based on the shortcut mechanism. The road network is preprocessed offline and partitioned into different regions. Each region has one core node and shortcuts between core nodes are constructed. We utilizes shortcuts to omit most of the searching process when searching a long-distance source-target pair. Besides, the HOD and HOE mechanisms are proposed to correct the detours of the approximated shortest paths. Experimental results show that the proposed algorithm can effectively reduce the searching time with an acceptable path length deviation.

References

1. Chen, B.Y., Chen, X.W., Chen, H.P., Lam, W.H.K., Talley, W.: Efficient algorithm for finding k shortest paths based on re-optimization technique. Transp. Res. Part E Logs Transp. Rev. **133**, 101819 (2020)
2. Chou, Y.L., Romeijn, H.E., Smith, R.L.: Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. Inf. J. Comput. **10**(2), 163–179 (1998)
3. Delling, D., Nannicini, G.: Core routing on dynamic time-dependent road networks. Informs J. Comput. **24**(2), 187–201 (2012)
4. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02094-0_7
5. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
6. Dreyfus, S.E.: An appraisal of some shortest-path algorithms. Oper. Res. **17**(3), 395–412 (1967)
7. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using contraction hierarchies. Transp. Sci. **46**(3), 388–404 (2012)

8. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: efficient point-to-point shortest path algorithms. In: Proceedings of the Meeting on Algorithm Engineering & Expermiments, pp. 129–143. Society for Industrial and Applied Mathematics, USA (2006)
9. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Better landmarks within reach. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 38–51. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72845-0_4
10. Gonzalez, H., Han, J., Li, X., Myslinska, M., Sondag, J.P.: Adaptive fastest path computation on a road network: a traffic mining approach. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, VLDB Endowment, pp. 794–805 (2007)
11. Gutman, R.: Reach-based routing: a new approach to shortest path algorithms optimized for road networks, pp. 100–111 (2004)
12. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
13. Lai, Y., Yang, S., Xiong, A., Yang, F., Li, L., Zhou, X.: Utility-based matching of vehicles and hybrid requests on rider demand responsive systems. IEEE Trans. Intell. Transp. Syst., 1–15 (2020)
14. Lauther, U.: An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In: Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung (2004)
15. Lauther, U.: An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edge-flags. In: DIMACS Book, vol. 74 (2009)
16. Li, L., Kim, J., Xu, J., Zhou, X.: Time-dependent route scheduling on road networks. SIGSPATIAL Spec. **10**(1), 10–14 (2018)
17. Macqueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability 1967 (1967)
18. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speed up Dijkstra's algorithm. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 189–202. Springer, Heidelberg (2005). https://doi.org/10.1007/11427186_18
19. Shapiro, J., Waxman, J., Nir, D.: Level graphs and approximate shortest path algorithms. Networks **22**(7), 691–717 (2010)
20. Wagner, D., Willhalm, T.: Geometric speed-up techniques for finding shortest paths in large sparse graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 776–787. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39658-1_69
21. Wang, Y., Li, G., Tang, N.: Querying shortest paths on time dependent road networks. Proc. VLDB Endowment **12**(11), 1249–1261 (2019)
22. Zhang, D., Yang, D., Wang, Y., Tan, K.-L., Cao, J., Shen, H.T.: Distributed shortest path query processing on dynamic road networks. VLDB J. **26**(3), 399–419 (2017). https://doi.org/10.1007/s00778-017-0457-6