Distrib Parallel Databases (2015) 33:227–252
DOI 10.1007/s10619-014-7146-x

Efficient route search on hierarchical dynamic road networks

Jiajie Xu · Yunjun Gao · Chengfei Liu · Lei Zhao · Zhiming Ding

Published online: 2 March 2014

Abstract The widespread use of GPS navigations and trip planning on web has aroused considerable interests in fast and scalable path query processing. Existing research has mostly focused on static route optimization where the traffic network is assumed to be stable. Nevertheless, in most cases, route planning is in the presence of frequent updates to the traffic graph due to the dynamic nature of traffic network, and such updates always greatly affect the performance of route planning. Most existing methods, however, cannot efficiently support traffic aware route planning. In this paper, two efficient strategies are proposed to handle this problem. We analyze the traffic condition on the road network and explore spatio-temporal knowledge to guide effective route planning. In particular, several effective techniques are employed to avoid both

A preliminary version of this work has been published in [28]. Substantial new technical materials have been added to this journal submission. In particular, we propose two novel route search strategies on the hierarchy of dynamic road network, as explicitly pointed out in Sect. 2 of this paper.

J. Xu · L. Zhao
School of Computer Science, Soochow University, Suzhou, China
e-mail: xujj@suda.edu.cn

L. Zhao
e-mail: zhaol@suda.edu.cn

Y. Gao (✉)
College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China
e-mail: gaoyj@zju.edu.cn

C. Liu
Faculty of ICT, Swinburne University of Technology, Melbourne, VIC, Australia
e-mail: cliu@swin.edu.au

Z. Ding
Institute of Software, Chinese Academy of Sciences, Beijing, China
e-mail: zhiming@iscas.ac.cn

unnecessary calculations on huge graph and excessive re-calculations caused by traffic condition updates. A comprehensive experiment is also conducted to evaluate the performance of our proposed strategies.

Keywords Route planning · Road networks · Road hierarchy · Dynamic road networks · Spatio-temporal analysis · Query processing

1 Introduction

With the decreasing manufacturing cost and increasing population, the number of automobiles in most big city surges significantly in recent years. In contrast, few of those cities have carried out significant improvement on the underlying road network facilities. As a result, the traffic problem is becoming greatly intensified, e.g., people in Beijing (China) spend almost two hours on traffic everyday in average, according to the latest report. This phenomenon has aroused wide interests in real time route search for various applications such as travel navigation and logistics. In this paper, we propose two solutions to find out the fastest route in dynamic road networks.

Most of existing route planning algorithms, for either travel distance or travel time optimization, are generally based on a static graph, meaning that actual road condition changes cannot be considered in the route planning process. However, a basic feature of road network is its high dynamics. As shown in Fig. 1, the best route planned (the green line in Fig. 1a could turn out to be slow 10 min later in the rush hours. Thus, it is important to achieve traffic aware route planning like Fig. 1b does, such that the planned route can be efficiently adjusted when necessary to guide users bypass new congestions. We aim to handle this problem in this paper in order to help users who tend to get stuck in the peak hour traffic congestions.

The main challenge of traffic aware route planning comes from the expensive computational overhead. For instance, on the Beijing road network that contains over 40,000 links, even a single calculation for the fastest path search would cost iCarTel several seconds. Since the road condition frequently changes in rush hour, it is obviously not realistic to re-plan the whole path for each road condition update, especially when the number of concurrent queries grows. Therefore, it is crucial to improve classical shortest path search algorithms, and to find ways for avoiding the time of

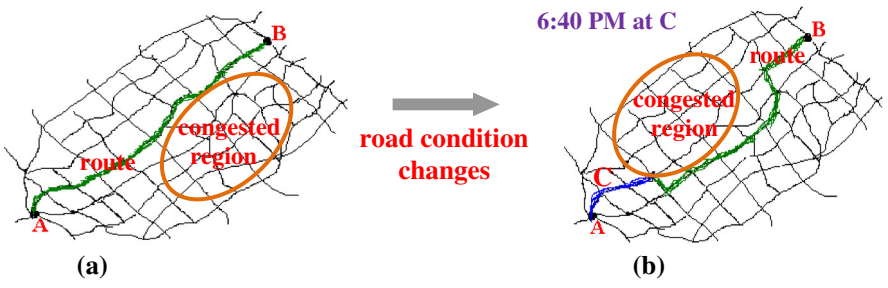

Fig. 1 Illustration of route planning. **a** Illustration of route planning, **b** revised earliest arrival route

re-computation caused by the dynamics of road condition. Due to the huge complexity of this problem, we aim to explore near-optimal solutions in an efficient and practical manner.

However, route planning on dynamic road networks needs to address two major challenges. The first challenge comes from real time response, which requires the query to be processed in a small search space; and the second challenge is that the road condition updates may cause excessive re-calculations. We would like to tackle these challenges by proposing two approaches in this paper. The first method is called the incremental route search (IRS) strategy. The main idea of IRS strategy is to filters out all faraway road segments, and to compute a partial path each time towards some selected intermediate destinations based on some criteria. The second approach is referred to as the hierarchical route search (HRS) strategy, which generates small search graph based on the hierarchical road network, and then computes the fastest path on it. To sum up, the key contributions of this paper are summarized as follows:

- We solve the traffic aware route planning. By adapting to the traffic condition changes, our methods can guide drivers to bypass the new congestions and to follow the best route in dynamic road networks continuously.
- We develop two novel approaches to process the fastest path queries in a small sub-graph every time. Computational cost is significantly reduced to support real time feedback of route queries, especially for the long distance queries.
- We take into account several features such as road hierarchy and driving flexibility of selected path to find out a reliable route in the presence of high dynamics of traffic condition in peak hours.
- We conduct extensive experimental evaluation to demonstrate the performance of our proposed algorithms in different traffic settings.

It is worth noting that, a continuous route planning method was proposed in [20], with a basic idea to store k best paths between any two vertexes offline based on some speed patterns, and to find the online best path by ranking the k best from current position to destination. It is a very practical solution from the system point of view, but it requires the speed pattern to be very accurate, because good paths cannot be found otherwise as it is not covered in the k best paths. Instead, our solutions are not based on the assumption that road conditions always follow some particular speed patterns as [20], and the IRS and HRS strategies are thus more robust to the dynamics of traffic for finding reliable paths.

The rest of this paper is organized as follows. We review the related work in Sect. 2. Section 3 describes the model and problem studied in the paper. Section 4 presents the preliminaries. Sections 5 and 6 elaborate the IRS and HRS strategies respectively for the route planning over dynamic road networks. Extensive experimental results and our findings are reported in Sect. 7. Section 8 concludes the paper with some directions for the future work.

2 Related work

Shortest path search is a classical problem with several typical solutions like Dijkstra, A* and its variation [16,19], which traverses the road network nodes in ascending order

of their distance from query position, and runs in $O(nlogn+m)$ time by using Fibonacci heap. Recently, many literatures like *highway hierarchies* (HH) [9,25], *contraction hierarchies* (CH) [2,9], *transit-node routing* (TNR) [1], *network indexing* [3,23,24,27], *landmark* [10], and *estimation* [22] algorithms try to exploit the hierarchical structure of road network in a pre-processing step, and then properly use it to speed up the search. HH and CH are based on shortcut techniques, i.e., some paths in the original graph are represented by some shortcut edges. To answer a query, a bi-directional search is executed on the overlay graph that constitute of the shortcuts and some edges in the original graph. As the shortcuts are the only extra structure stored in the index, the construction is relatively fast as compared to other index approaches. TNR is a method that makes use of the observation that a driving path usually passes one of a few important transit nodes. Though HH, CH and TRN also run in $O(nlogn+m)$ time, the search process can be significantly accelerated because of their much less search space.

More recently, some other major efforts related to this topic mainly include *probabilistic* path queries [13], dynamic k-NN [21], path oracles and efficient processing [7,8,26], *skyline* queries [5,17], trip planning with *multiple destinations* [18] and *complex road network structure* [14,29]. In particular, in [26], part of the shortest path distances called *path oracles* is pre-computed to answer approximated shortest path query in $O(log|V|)$ time and $O(\frac{|v|}{\varepsilon^2})$ space, where ε is the error bound of approximation. Nevertheless, they rely on heavy pre-processing, and are thus not suitable for dynamic scenarios where the road network topology or edge weights (e.g., when denoting the time to pass through a road segment) are greatly changed.

However, the above distance based approaches do not consider the actual road condition in the route planning, and as a result, they cannot provide satisfactory planning results in most cases. Recently, increasing attentions have been put on *time-dependent* shortest path search problem, where the shortest path search is based on a dynamic graph due to the ever changing of edge weight (i.e., the time required to pass this road). Classical algorithms like $D*$ can be used to find the shortest path continuously on dynamic graph, but they are essentially low-efficient solutions for traffic planning scenarios because of the frequent speed updates. Some *time-dependent* route search mechanisms were proposed in recent years [6,11,12,15], and most of them are pattern based approach, which assumes that some knowledge or speed patterns about traffic are available, and then makes use of such speed patterns to find the fast paths to destination.

Gonzalez et al. [11] proposed a traffic mining based fastest path search approach. It uses mining techniques to derive frequent driving patterns, and then computes the adaptive fastest path based on the match pattern. Route update is made when the matched pattern switches. Ding et al. [6] studied the time-dependent shortest path problem, with a Dijkastra based algorithm proposed to find the best departure time and fastest path over the network, according to the speed pattern based on statistically average road condition in different time intervals. Also, a critical-time-point approach was presented in [12] to generate best routes and their corresponding time intervals, and the issue of finding the fastest paths on a road network with speed patterns was discussed in [15]. Nevertheless, these methods only rely on the speed patterns derived from traffic data, which indicates, the searched path is unlikely to be optimal since the real time information on road condition is not considered.

Table 1 Symbols and descriptions

Notation	Description
$ind(v)$, $outd(v)$	The in-degree and out-degree of vertex v
L	A road network hierarchy set
$lv(e)$	The hierarchical level of a road segment e
$\pi(i)$	A set of road segments of the hierarchical level i
C	Road condition
S	$s_e \in S$ is the speed on a road segment e under a road condition
T	$t_e \in T$ is the required time for passing a road segment e under a road condition
$traffCond(t)$	The traffic condition of time t
src, $dist$	Source vertex and destination vertex of a route query

More recently, some novel route planning approaches considering real-time road condition are proposed. Malviya et al. [20] targeted to answer continuous route planning queries over a road network in the presence of speed updates on road segments. Its basic idea is to calculate k fastest paths (based on speed patterns) between any two vertexes with variance guarantee at the build time, and then to keep ranking the k fastest paths from current position to destination at the run time. Also, a heuristic based bidirectional route planning algorithm was proposed in [4] to speed up the search process. However, the aforementioned approaches require huge computational overhead, and hence, they are not feasible for the applications where the road conditions are frequently updated.

Note that, we conducted preliminary work on the route planning in a dynamic traffic network [28]. Based on this work, in this paper, we present two novel route search strategies on the hierarchy of dynamic road networks.

3 Problem statement

In this section, we give a formal description of traffic aware route planning. Table 1 lists the symbols used frequently in the rest of this paper. Several definitions based on the traffic network are presented to specify the problem we intend to address in this paper:

Definition 3.1 *(Road Network)* A road network is defined as a directed graph $G =< V, E >$, in which $V = \{v\}$ is the set of vertices representing road ends or **intersections**, and $E = \{(v_i, v_j | v_i, v_j \in V\}$ is the set of (directed) edges representing **road segments**. For each vertex v, $ind(v)$ and $outd(v)$ indicate the in-degree and out-degree of v according to the topology of G respectively. Given a route r with the set $r.V$ of all vertices it passes, the in-degree of r is defined as $IND(r) = \sum_{v \in r.V} ind(v)$ and out-degree is defined as $OUTD(r) = \sum_{v \in r.V} outd(v)$. For each edge e, we use $name(e)$ to denote its road name (e.g., 'Elizabeth Street'), and $lanes(e)$ to denote the number of lanes on e.

The in-degree and out-degree of vertices are important attributes for route planning. Given a vertex v, congestions are more likely to occur on v if $ind(v)$ is high due to

its influx of traffic flow. In contrast, vertices with high $out(v)$ are preferred because drivers have more (outgoing) paths to choose from. Such flexibility is very useful to bypass the new congestion that may occur ahead.

Definition 3.2 *(Hierarchy of Road Segments)* The road network may have hierarchical structure, where the road segments are classified into different *hierarchical levels* $L = \{1, 2, \ldots\}$, in which level 1 is the lowest. Given a road segment e with name $n = name(e)$, the average lane number of all road segments having name n can be calculated as

$$AvgLanes_(n) = \sum_{e' \in RD(e)} lanes(e')/|RD(e)|$$

where $RD(e)$ is the set of road segments $\{e' | name(e') = name(e)\}$ having the same name (e.g. 'Queen Street') with e. We can further measure its hierarchical level *lv (e)* of road segment e according to the *AvgLanes(n)* based on the name of e as:

$$lv(e) = round(AvgLanes(name(e)))$$

where function *round* returns the closest integer value of average number of lanes.

In this model, all road segments belonging to the same road are considered as at the same level. We reckon that the major roads (i.e., important and reliable roads) are those at higher hierarchical level. Function $\pi(i)$ returns the set of edges belonging to level i.

Definition 3.3 *(Road Condition)* The road condition of a road network can be expressed by $C =< S, T, D >$, where S and T are the speed and required travelling time for road segments, and D denotes the time duration when this road condition is valid. Given a road segment e, the speed on e is represented as $s_e \in S$, and the time required for passing e is $t_e \in T$. Given a time t, we use function *traffCond*(t) to return the road condition C satisfying $t \in C.D$.

Definition 3.4 *(Route Query)* In a dynamic road network G, a route query is defined as $qry = <src, dst>$, where *src* is the source vertex and *dst* is the destination vertex specified by users. We use this format as standard query type in this paper. In our system, users can also input $<src, dst,$ "no"$>$ to answer the fastest route query without continuous monitoring.

Problem Definition We target to solve the problem of traffic aware route planning which is formally defined as: Given a route query $qry = <src, dst>$ on a road network $G =< V, E >$, with possible road condition updates $C_0, C_1, C_2, \ldots$, we process the query for a continuous optimal path (route) $pth = (v_s, v, v', \ldots, v_t)$ on this dynamic road network that satisfies the following spatio-temporal optimisation goals and constraints:

(1) Spatial constraints: source vertex $v_s = src$ and destination vertex $v_t = dst$;
(2) Traffic condition constraints: route planning is subject to the actual road condition and its continuous updates;
(3) Optimisation goal: The total travel time should be minimized. That means, any route $pth' = (src, \ldots, dst)$: total travel time $\sum_{e \in pth} time(e) \leq \sum_{e \in pth'} time(e)$.
(4) Continuous optimisation: the result should be updated as optimum when the traffic condition environment changes.

This problem is computationally hard due to the huge scale of graph (i.e., a road network) and the continuous re-calculation caused by the excessive traffic condition updates on this graph. However, GPS navigation requires efficient query processing for immediate response. Therefore, an efficient approach is highly sought after to tackle this problem.

4 Preliminaries

In this section, we review the preliminaries of shortest path search, graph reduction, and road condition monitoring in Sects. 4.1, 4.2 and 4.3, respectively. Note that effective shortest path search and graph reduction are essential issues of route planning.

4.1 Shortest path search (A* algorithm)

The shortest path search on graph is a classical topic and Dijkstra algorithm is a fundamental algorithm commonly used to solve it. Many approaches were designed to improve Dijkstra algorithm, and the most famous one is A* algorithm. It adopts the hill climbing method, and the ordering of search is based on the function $f(x) = g(x) + h(x)$, where $g(x)$ denotes the exact distance from source vertex to current vertex x and $h(x)$ is an estimated distance (also the strict upper bound of the distance) to destination vertex.

A* algorithm is usually adopted to find the shortest path on a static (current) graph. To avoid over-estimation, the distance and time used for computing heuristic estimation $h(x)$ are 'as-the-crow-flies' distance and maximum speed limit on the road segments respectively. This approach costs $O(m + n\log n)$ time, where m is the number of edges and n is the number of vertices. However, A* algorithm is not effective for time-dependent shortest path cases because the maximal speed limit is used in estimation (to guarantee $h(x)$ is not over-estimated). As A* heuristic is not sensitive to the actual road condition, earliest arrival query processing on a large graph turns to be very inefficient, particularly when the traffic is heavy. To ensure that time-dependent queries can be efficiently responded, our approach seeks to reduce the size of n, so that a route query can be processed in a small sub-graph each time to improve efficiency.

4.2 Graph reduction

Although A* algorithm is very effective for the shortest distance query, its performance is usually poor for earliest arriving route queries on a large graph. This is caused by the fact that traffic conditions cannot be used in the A* heuristics to guide route search. Moreover, the continuous monitoring on the dynamic traffic network must be restricted in a relative small sub-graph. Therefore, graph reduction techniques can be used to ensure that route queries can be efficiently processed.

Graph reduction is applied in [20] to facilitate the shortest path algorithm for dynamic transportation networks. It reduces the whole space to an ellipse region, and then monitors the road condition change in this region. Only road segments in this region are admissible for route planning. Selected route is updated when the change

in this region reaches a threshold because it may not be optimal since then. However, these approaches are too mechanical since they only consider the locations of source vertex and destination vertex. In Sect. 5, we propose a set of spatio-temporal feature based techniques to shrink the search space for each computation.

4.3 Road condition monitoring

As the dynamics of traffic network is a core issue for location based service optimisation, making use of information about updates of road condition in a close area is essential for achieving traffic-aware route planning. Many previous efforts have been endeavoured to address this problem. Among the existing approaches, the most practical solution is the technique used in [20], which sets up a relative region first, and then checks if the number of segments with delay updates that locate inside the region for a route query exceeds the average number of segments locating inside this region. If it is true, the re-run of the shortest path computation (by A* algorithm) is invoked then. In this paper, we adopt this basic idea, and propose an efficient monitoring approach to guarantee that the selected route can be properly updated.

5 Incremental route search (IRS) strategy

In this section, we present an IRS strategy that keeps finding efficient partial paths, so that excessive re-calculations on faraway road segments caused by the frequent road condition updates can be avoided. In Sect. 5.1, we conduct a basic graph reduction to improve planning efficiency. Then top-k intermediate destinations are selected according to several spatio-temporal criteria in Sect. 5.2. Afterwards in Sect. 5.3, route planning is conducted towards the top-k intermediate destinations. In this way, we can avoid re-calculation on faraway road segments due to high possibility of speed change. Finally, the monitoring technique for the adaptation of route planning to dynamic road conditions is introduced in Sect. 5.4.

5.1 Initialization

In the initialization phase, we conduct a basic graph reduction to settle a small subgraph as the region relative to route query processing: an ellipse region G' shown in Fig. 2 is efficiently derived from the whole space G in the same way as [20]. Only the road segments in this region are considered as relevant to route planning. However, as only positions of source vertex and destination vertex are considered, this ellipse region is very likely to be over-sized, and we only utilize it to set a base for route planning operations used in remaining sections.

5.2 Top-k intermediate destinations

In traditional approaches, a path strictly from source to destination is usually planned in each time. Nevertheless, it is not effective for earliest arriving route search because the frequent updates on road condition are likely to cause excessive re-calculation,

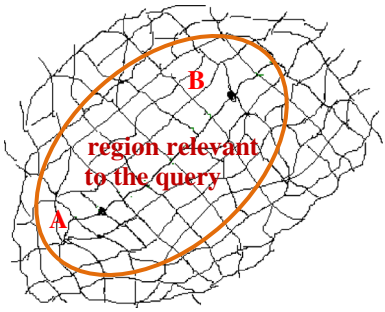

Fig. 2 Basic graph reduction (default ellipse region)

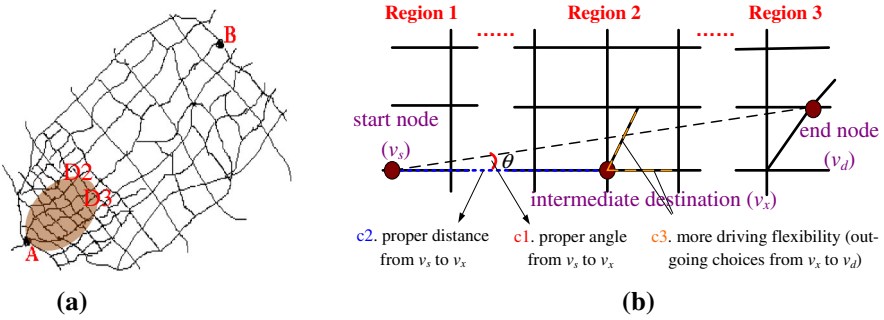

Fig. 3 Illustration of intermediate destination selection. **a** Intermediate destination, **b** evaluation criteria

especially on faraway road segments. For example, if congestion occurs on a road segment that is part of planned route, re-planning is needed to guarantee service quality. For efficiency purpose, it is thus reasonable to plan partial path in a limited scope, rather than plan the whole route. Re-calculations caused by dynamic road condition can be significantly reduced accordingly. To set the boundary of route search properly, we must select some intermediate destinations as shown in Fig. 3a, toward which effective route planning is conducted afterwards.

The selection of intermediate destinations must follow a set of spatio-temporal standards. First of all, the direction from source vertex to intermediate destination has great evaluating merit. The direction to intermediate destination is expected to be consistent with that to final destination. Assume that $ang\ ((v_s, v_x), (v_s, v_d))$ is the angle of two straight lines $(v_s \rightarrow v_x)$ and $(v_s \rightarrow v_d)$, e.g., 'c1' in Fig. 3b, it can be seen as the difference between direction to the intermediate destination and direction to the final destination. We definitely prefer small angle due to less difference. To ensure the direction to be consistent, only angles in [0, 90°] are accepted. Given that $cos(\theta)$ is the cosine value of an angle θ, which is in reverse proportion to the degree of ang. To evaluate the direction preference of selecting v_x as intermediate destination, we use $dw(v_x)$ to measure the direction weight of v_x as:

$$dw(v_x) = \cos(ang((v_s, v_x), (v_x, v_d)))$$

Meanwhile, the position of a vertex regarding to source and destination is an important criterion according to 'c2' of Fig. 3b. A faraway intermediate destination may cause

excessive re-calculation due to the frequent updates on traffic condition. Furthermore, the distance should not be too close as the global view is neglected: it is hard to satisfy global optimisation when partial path search is made toward an intersection 200 m away. To achieve a good balance between reducing re-calculation (not too faraway) and achieving global optimisation (not too close), we use position weight $pw(v_x)$ to measure intersection v_x as intermediate destination as:

$$pw(v_x) = t_{\max} - |h(v_s, v_x)) - t_{best}|$$

where $h(v_s, v_x)$ denotes the average travel time for the 'as-the-crow-flies' distance (e.g., 10 miles) from v_s to v_x on the current issue time (e.g., 7PM). Statistical data is used here. t_{max} is the upper bound of $h(v_s, v_x)$ for pruning faraway vertices, and t_{best} is the time of favoured distance based on statistics.

Another important criterion for evaluating intermediate destinations is the flexibility of future driving. According to 'c3' of Fig. 3b, high flexibility means better capability for exception handling, e.g., to bypass new occurred congestions. Driving flexibility of an intermediate destination is determined by a set of spatio-temporal features. It is obvious that more out-going paths from an intersection give us more flexibility to choose. Among out-going edges, those in the same direction to final destination are definitely preferred. Wrapping up these issues, the flexibility weight $fw(v_x)$ for selecting vertex v_x as intermediate destination is calculated as:

$$fw(v_x) = \sum_{v \in FV(v_x)} \cos\left(\frac{\text{ang}((\text{v}_\text{x}, \text{v}), (\text{v}_\text{x}, \text{v}_\text{d}))}{2}\right)$$

where $FV(v_x) = \{v \mid \text{edge } e = (v_x, \text{v}) \in E\}$ is the set of forward vertices of v_x, $\theta = ang((v_x, v), (v_x, v_d))$ is the angle of two straight lines $v_x \rightarrow v$ and $v_x \rightarrow v_d$. More out-going edges give drivers greater flexibility of route selection. For each out-going edge to v from v_x, we prefer θ to be small since its direction matches the required one. The cosine value is in reverse proportion to the angle size θ in range of $\theta \in [0, 90°]$. We thus use it to evaluate the preference of out-going paths regarding to moving direction.

To select proper intermediate destinations, issues mentioned above like spatial features and driving flexibility must be considered, as they benefit us to reduce re-calculations and to be more reliable under dynamics. In particular, the selectivity of intermediate destinations follows the following criterion:

$$w(v_x) = dw(v_x)/cd + pw(v_x)/cp + fw(v_x)/cf$$

Where $dw(v_x)$ is the direction weight, $pw(v_x)$ is the position weight, $fw(v_x)$ is the flexibility weight. Factors cd, cp and cf represent the average direction weight, position weight and flexibility weight of all vertexes in the relevant ellipse respectively. Thus, only vertices satisfying $w(v_x) \geq 3$ are qualified as intermediate destination. By ranking $w(v_x)$ on vertices that meet this requirement and do not belong to the congested region, k best vertices and the final destination are selected as the set of intermediate destination candidates, denoted as *IDC*. Route search is then conducted towards vertices in *IDC*. If no intermediate destination candidate can be detected, A* algorithm is used to search time-dependent shortest path to final destination v_d.

Algorithm 1. Partial Path Search (PPS) Algorithm
Input: *IDC*: intermediate destination candidates
v_s: source vertex
v_d: destination vertex
G': sub-graph of the road network relevant to query
Output: *pth*: a partial path

```
01. PV= {v_s};     /* processed vertices */
02. AV = {v' | v' is adjacent to v_s};
03. SP = null;     /* SP→v is the shortest path from v_s to vertex v in PV */
04. NumIntmDestReached = 0;
05. while NumIntmDestReached <|IDC|/2
06.       LowBound = 0;
07.       foreach v∈AV
08.             foreach v' ∈ PV having edge (v', v)  ∈ G'.E
09.                   LowBDTime(v, v') = time(SP→v) + dis_EU (v, v_d)/Speed_max;
10.             endfor;
11.             if LowBDTime (v, v')< LowBound then LowBound = LowBDTime (v, v') endif;
12.       endfor;
13.       select v_i∈PV, v_j∈AV: EstmTime(v_i, v_j) = MinTime;
14.       PV = PV +{v_j};
15.       AV = AdjustAdjctSet(AV);
16.       SP→v_j = SP→v_i  ∪ {edge(v_i, v_j)};
17.       if (v_j∈IDC)
18.             insert SP→v_j to PC;
19.             NumIntmDestReached= NumIntmDestReached+1;
20.       endif;
21. endwhile
22. pth = selectFromPaths(PC);
23. return pth;
```

5.3 Route search

Route search is made to find a path (partial route) to one of the top-k intermediate destination. We propose a novel algorithm to search the partial route efficiently. Given a partial path, we measure its cost as the sum of exact time from source vertex to intermediate destination and the lower bound of time required from intermediate destination to final destination (based on the Euclidean distance and maximal speed allowed). Lower bound is used here to compress the search space by filtering. Details of the (partial) route search toward intermediate destinations are given as follows.

Algorithm 1 is an efficient partial path search strategy. Compared with the conventional shortest path search, the optimization here is made for $k$rather than just one candidate, and the route returned is a partial one. Set *PV* contains all the processed vertices, in which the earliest arrival time from source vertex is known and recorded in *SP*. Set *AV* contains un-processed vertices adjacent to *PV*, and the search process is made on a vertex in *AV* each time.

For each vertex in adjacent vertex set *AV*, the minimal time required to travel through this vertex to destination is calculated in Lines 8–10, where $LowBDTime(v,\ v')$ is the lower bound of the time required to pass v and v' to destination, $time(SP\rightarrow v)$ is the shortest time from source vertex to v (refer shortest path record in *SP*), $dis_{EU}(v,\ v_d)$ is the Euclidean distance from v to destination, and $Speed_{max}$ is the possible maximal speed of the road network. Variable *LowBound* is the minimal time required to pass a vertex in *AV* to destination, and it is updated if we find v having $LowBDTime\ (v,\ v') < LowBound$ (Line 11). A vertex pair $v_i \in PV$ and $v_j \in AV$ that satisfies

EstmTime(v_i, v_j) = *MinTime* is selected in Line 13. As v_j has the minimal (estimated) time to destination among un-processed vertexes, we include the path shortest path to v_j in *SP*, and update the *PV* and *AV* set in Lines 14–16. If v_j is one of the intermediate destinations, the partial path $SP \rightarrow v_j$ is added to candidate path set *PC* (Line 18). This procedure continues until over half of intermediate destinations are reached.

Then we need to select the best path among several path candidates stored in set *PC*. The best path is selected by function *selectFromPaths*(*PC*) in Line 22. Road structure features and estimated travel time of the paths are considered by this function. Then we calculate the total number of outgoing paths $o = OUTD(pth)$, the total travel time t and the average lane number l of road segments on *pth*, and select the one with the minimal value of $\sqrt{o \times l}/t$ (more driving flexibility and less travel time) to return as the partial route to go.

5.4 Monitoring and update

Due to the high dynamics of road condition in rash hour, it is essential to monitor the road condition and react to the relevant updates on it. In the IRS strategy, we conduct road condition monitoring in the same way as [20]. On arrival of delay updates, we check if the number of road segments affected by the updates that lie inside the pre-computed ellipse for a routing query is larger than ε (a threshold) times the average number of segments lying inside an ellipse of this area. If so, we simply re-run the intermediate destination and route search and return a real-time optimal path to the end user. In this way, computational overhead for the continuous monitoring can be significantly reduced.

6 Hierarchical route search (HRS) strategy

The path derived from IRS strategy each time is a partial one, but this does not meet the expectation of users sometimes, and causes extra cost on monitoring. To tackle this problem, we further introduce a novel HRS strategy. To reduce computation overhead, the basic idea of HRS is to plan the route and monitor it on a compressed (rather than the whole) road network. The construction of compressed road network is query aware (i.e. subject to query point and destination). To ensure robust good route can be efficiently found in it, we tend to keep the useful road segments based on some measures, e.g. the fast roads in close region and the major roads in distant region. In addition, we collect and use the road condition change information subject to the compressed road network to adjust the planned route rationally.

Specifically, we introduce the pre-processing in Sect. 6.1, which includes hierarchical road network partition and indexing. Afterwards, we discuss how to generate a search graph with suitable size subject to a route query in Sect. 6.2, and efficiently process a route query on that search graph in Sect. 6.3. In this way, a proper whole path to final destination can be computed in real time. We then discuss how to achieve route monitoring and to conduct route updates properly in Sect. 6.4.

6.1 Pre-processing

In the pre-processing step, we partition the road network based on its hierarchical structure, and create index to facilitate efficient access on sub-graph information.

Towards the offline the road network partition, we start from the highest hierarchical level lv, and find the set of undirected edges in this level $E_{lv} = \{(v,\ v')|(v, v')\pi(lv)$ or $(v',\ v)\pi(lv)\}$ first. Then we partition the whole space based on E_{lv} as follows: (1) first, we detect all of the rings subject to the undirected edges in E_{lv} (each ring denotes a sub-graph); (2) for each ring, to retrieve road segments and find those spatially inside it to construct a sub-graph, and these graphs are spatially coherent and only share boundary edges belonged to E_{lv}. All edges not inside a ring are integrated as a sub-graph; (3) then for each sub-graph, the partition on it is further conducted as above based on road segments in level no less than $lv - 1$; This procedure continues until it reaches a level that should stop.

Along with the road network partitioning, we also construct a hierarchical index to preserve the hierarchical relation of different parts, so that efficient access can be achieved. The format of a hierarchical index node n is $n =<id, level, pred, succ, VE, mbr>$, where *id, level, pred, succ* are the ID, index level, predecessor and successors of node n (based on partitioning operation), and *VE* is the visible edge set may appear when this node is included. That is, assume we derive a sub-graph sg through partitioning based on road segments in level i, then only road segments in the next level (i.e. in level $i-1$) can be added to *VE*. Based on *VE*, we can find its *mbr*, i.e. the Minimum Bounded Rectangle (MBR) of *VE*.

For each hierarchical index node n, we can easily find its border intersections after partition, where the border intersections $BOR(n)$ of a node n is defined as its intersections that also belongs to one or more sibling nodes of n in the hierarchical index. In addition, we use $spd(n,\ C)$ to reflect the road condition in the region (subject to its visible edges *VE*) under a road condition C, and $dist(curr,\ G_i)$ to denote its distance to the current position *curr*, and they are calculated as:

$$spd(G_i, C) = \frac{\sum_{e \in E_i} C.s_e}{|E_i|}$$

$$dist(curr, n) = \frac{\sum_{v \in BOR(n)} dis_{EU}(v, curr)}{|BOR(n)|}$$

where $C.s_e$ is the speed of road segment e under the road condition C, and $dis_{EU}(v,$ *curr*) is the Euclidian distance from a given vertex v (border intersection) to the current position *curr*. Such index and features are used in the search graph generation for route search.

6.2 Search graph generation for route search

As route planning and monitoring on huge road network is usually inefficient while real time response is needed, we hope that all operations can be made on a small part of the road network, i.e. called search graph. The key problem is to find a search graph

with all useful road segments preserved in despite it is small enough. To achieve this, we try to explore a suitable search space *SG* that is formed by the major roads and fast road segments rationally.

The major challenge comes from the problem how to rank the importance of fast and major road segments regarding to a given route query. The spatial information is useful for balancing hierarchy and speed preferences. For the road segments close to the current position, current traffic status is definitely more important, so that we hope to keep fast road segments. In contrast, for the faraway road segments, road hierarchical information usually becomes more important. This is due to the fact that the speed on these road segments when actually passing it in future may be varied from the current speed, especially in peak hours. We use these important rules to guide search graph compression in HRS approach.

Search graph initialization Hence, a basic reduction is conducted as Sect. 5.1 first, through which a default ellipse region *DR* is derived (i.e. same as Fig. 2). It is a big region such that road segments outside are considered as irrelevant to query. Assume that *root* is the root node of hierarchical index I in pre-processing step, and $succ(n)$ is the function to return the set of all successor nodes of a given node n according to the index hierarchy. For each index node $n \in succ(root)$ that appears as child of root node of hierarchical index created in pre-processing step, we get its minimum bounding rectangle *n.mbr*. If $n.mbr \cap DR \neq \emptyset$, the visible edges *n.VE* (i.e. road segments in highest hierarchical level) of this node and involved vertexes are added to the search graph *SG*. Otherwise, the node n is not relevant to query. Also, we continuously add the hierarchical index nodes for including *curr* and *dstn*, so that they can be reachable via the *SG*.

After the search graph initialization where only top level hierarchical index nodes are involved, only the arterial roads (in greatest hierarchy) are included in the search graph. Obviously that route search cannot be simply carried out on *SG* because it only contains major road segments, which are not sufficient to provide a good route search result. To guarantee the precision of route search, we further expand the search graph by adding road segments that will appear on the optimal path.

Search graph expansion A divide-and-concur approach is used in HRS strategy to expand *SG* for this purpose: we continuously select the hierarchical index node (i.e. a sub-graph essentially) that is most relevant to route query q, and then apply 'zoom-in' to include its visible edges as depicted in Fig. 4b, until a proper search graph such as Fig. 4c is available for route search. The key problem here is to select a suitable node in I to include, where some important sub-graph features (e.g., the road condition in this region and its distance to current position) must be taken into account.

To balance different regions in road segment inclusion, the density of road segments for the next visible edge inclusion becomes an important feature. If the region is far away from current location, a node with high density value can tend to be ignored as we only hope to get a sketched view using major roads. In this paper, the density $dens(n)$ of a road segment n is defined as

$$dens(n) = \frac{\sum_{e \in n.VE} Length(e)}{size(n)}$$

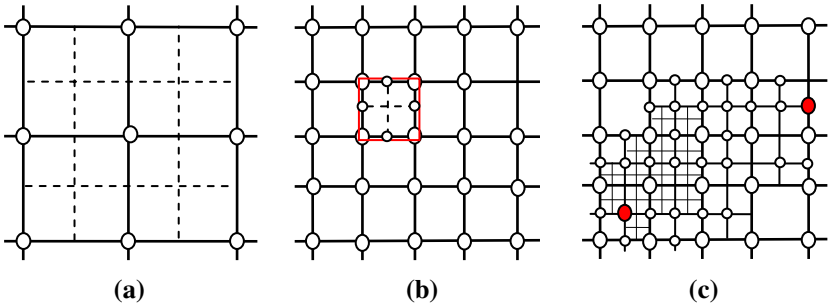

Fig. 4 Illustration of road network partition and search graph. **a** Default road network partition, **b** sub-graph 'zoom-in', **c** example of search graph

where $n.VE$ is the visible edges of node n (road segments that will appear in search graph if node n is included), $Length(e)$ is the length of a road segment e, and $size(n)$ represent the size of the region corresponding to a given node $n$based on its visible edges$n.VE$, and so far we use its mbr to compute an approximate value. Note that the density is a static value so we can calculate it offline, and it thus does not incur real time processing cost.

The core problem is to balance different factors (e.g., speed, position, direction, etc.) to select the most important node (i.e. edges) to include. Some global variables are employed to achieve this. The first variable is $SP_A = \frac{\sum_{e \cap DR \neq \phi} C.s_e}{|\{e|e \cap DR \neq \phi\}|}$, i.e. the average speed of edges in default ellipse region DR under current road condition C; the second variable is the average distance $D_A = \frac{\sum_{G_i \in SG} dist(curr, G_i)}{|\{G_i \in SG\}|}$ of all relevant hierarchical index nodes to location $curr$.

Since the fast roads are preferred in close regions while major roads are preferred in distant regions, we use D_A as the boundary indicator to separate them first. For the close regions corresponding to node n with $dist(n) < D_A$, we tend to filter out the slow regions that satisfies $spd(n, C) < SP_A$ if at lease k regions remain, or select top-k fastest close regions otherwise. For the faraway regions such as node$n$ with $dist(n) \geq D_A$, we tend to filter out the dense regions whose density is over D_A. The filtering can thus well balance different crucial factors in this way. Additional road segment inclusion is conducted on the most import region of remaining sub-graphs, the weight of the sub-graph of hierarchical index node n_i for 'zoom-in' selected can be calculated as,

$$W_i = \frac{spd(n_i, C) \times dens(n_i)}{dist(curr, n_i)}$$

The weight of node n_i for inclusion to search graph SG is in proportional to the average speed of the road segments in this region because users prefer the fast regions than others. In order to balance the close and faraway regions, W_i is in proportional to the road segments density in the region n_i belongs to. In contrast, W_i is in the reverse proportion to the Euclidian distance to current position. The reason is that the uncertainty of the speed that users actually pass a road segment increases with its distance to current position.

The generation of search graph for future route planning is processed as Algorithm 2. Initially, we find the set of hierarchical index nodes N that are children of root node (Line 1). Default ellipse region DR is then computed for basic graph reduction (Line 2). For each node n_i in N, we calculate the speed, density of this region and the weight for inclusion selection if its MBR intersects DR (Lines 4–11). Then we derive the total number of edges*numOfEdges*, average speed SP_A and average distance D_A of the current visible search graph. While the number of edge is less than allowed (Line 15), the set of candidate notes (candidates for 'zoom-in' inclusion) CN is derived based on hierarchy of index I (Line 17). By checking nodes in CN one by one, we filter out slow ones in close region, dense ones in far-away regions (Lines 18–22), and those would cause total edge number of VC to exceed expected value *ExpNum* if we apply zoom-in operation (Lines 23–24). Thereafter, we select the hierarchical index node n_k with maximal value of weight (Line 26) and update the visible search graph SG by adding n_k into it (Line 27). All the variables are updated afterwards (Line 28) to continue the visible region inclusion process, and SG is returned for route planning eventually.

Algorithm 2. Search Graph Generation
Input: *G*: road network
I: hierarchical road structure index
C: current road condition
*curr***:** current position
dstn: destination
Max: maximal number of visible edges
Output: *SG*: visible sub-graph for route planning

```
01. N = FirstHierarchy(I);
02. default ellipse region DR= getRelaventRegion(curr, dstn);
03. for each n_i ∈ N
04.       if n.mbr ∩ DR ≠ null
05.             add n_i to SG
06.             spd[i]= spd (n_i, C);      /* average speed of n_i */
07.             dist[i]= dist (curr, n_i);      /* average distance in n_i */
08.             dens[i]= dens(n_i);    /* average density in n_i */
09.             n_i→weight = computeWeight(spd[i], dist[i], dens[i]);
10.       endif;
11. end for;
12. numOfEdges = |{ edge | edge  ∈SG}|;      /* total number of edges in SG */
13. D_A = getAverageDist(dist);
14. SP_A = getAverageSpeed(spd);
15. while numOfEdge < ExpNum                /* conduct inclusion */
16.       CN = getCandidateNodes(SG);      /* the set of sub-graph for 'zoom-in' inclusion */
17.       foreach n_i ∈ CN
18.             if dist[i] <D_A
19.                   if spd[i] <SP_A    then CN = CN – { n_i } endif;
20.             else if i >1
21.                   if dens[i-1] > thrd or i >M then CN = CN – { n_i } endif;
22.             endif
23.             incrNum is the increased number of edges if we zoom in n_i ;
24.             if numOfEdge+ incrNum>ExpNum then CN = CN – { n_i } endif;
25.       endfor
26.       select n_k from SGS that has maximal value of n→weight;
27.       add n_k to SG;
28.       updateVariables();
29. endwhile;
30. return SG;
```

6.3 Route planning on search graph

Route planning is conducted on the search graph *SG* after it is generated in Sect. 6.1. As depicted in Algorithm 3, route search on *SG* is processed in a similar way as A*, where heuristic is employed to speed up the search process. As the distant part of the planned route may not be optimal, we select a vertex on the route afterwards as a reminder for future route improvement.

Starting from the *curr* vertex, we continuously include the vertex that has minimal time. *PV* is the set of processed vertexes whose minimal travel time is known, and *AV* is the set of vertexes that are adjacent to at least one vertex in *PV* but do not belong to *PV* (Lines 1–2). Each vertex inclusion is made in a while loop (Lines 5–14). Specifically, for each edge connecting vertex v in *PV* to vertex v' in *AV*, we compute the weight of v' for inclusion $f(v')$, which is the lower bound of the time via v' to destination. The weight $f(v')$ contains two parts, where $g(v')$ means the actual time from *curr* to v', and $h(v')$ is the lowest time required from v' to *dstn*. To guarantee $h(v')$ does not over-estimate the time to *dstn*, we use the Euclidian distance and the maximal speed allowed in all road segments in calculation. The search process is finished when the destination is reached, and the fastest route is finally returned to users. Like A* algorithm, the computational complexity of route search is $O(m+n \times \log n)$, where m is the number of edges and n is the number of vertexes. Nonetheless, the route search is efficiently processed because the scale of graph is restricted to a low level.

Algorithm 3. Route Search
Input: *SG*: search graph
C: current road condition
curr: current position
dstn: destination
Output: *route*: the optimal route

```
01. PV = {curr};      /* processed vertexes */
02. AV = {v | v belongs to SG and is adjacent to at least one vertex in PV but not belonged to it};
03. T = null;         /* T[i] denote the minimal time required from curr to v_i */
04. P = null;         /* P[i] denote the fastest path from curr to v_i */
05. while dstn is not in PV
06.       foreach e=(v∈PV, v'∈AV)      /* candidate edge for inclusion in route */
07.             g(v') = T[i] + C.t_e;
08.             h(v') = dis_EU (v', dstn)/MaxSpeed      /* MaxSpeed is the maximal speed allowed on
          all road segments */
09.             f(v') = g(v') + h(v');
10.       end for
11.       select j that e_j=(v∈PV, v'∈AV) that has maximal value of f(v');
12.       T[j] = T[i] + C.t_e;
13.       insert e_j to P[j];
14. endwhile
15. route is the path to dstn according to P;
16. return route;
```

However, as the distant sections of the planned route is not the optimal path because many road segments are filtered out from the visible graph, a verification point on *route* must be selected to indicate the validation of current plan. That means, current route should be revised before this point even though the road condition is unchanged. In our approach, we scan all vertexes on *route* in sequential order, and set the first intersection v as verification point if $\sum_{e \in SuccSubG(v)} lv(e) succSubG(v)$ is less than a threshold, where *succSubG(v)* returns the set of sub-graphs (on *SG*) to which the path of *route* from v to destination intersects. That means, for this partial path (from v to destination), we should conduct 'zoom-in' on the sub-graphs on the path and replan

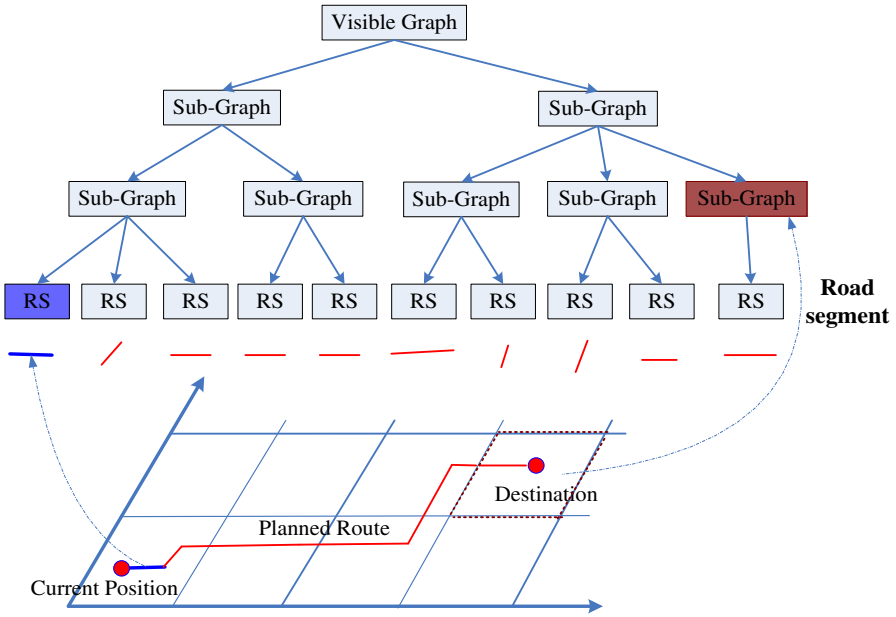

Fig. 5 The DM-Tree (for route monitoring)

the route to find the optimal path. Current route will be verified when the moving object is approaching point v.

6.4 Route monitoring and updating on search graph

Road condition monitoring starts immediately after a route is planned (by Algorithm 3). To update the planned route when necessary, we monitor both the current route and the search graph SG used in route search respectively. Delays on the current route are seen as reminder for route update (at least update on part of the route). By tackling road condition changes on SG, we understand the traffic status in close region in order to estimate if the current path is still optimal.

We frequently check the delays on the current route to detect new congestions, and the frequency of checking is higher in the close route section. To improve the effectiveness of route monitoring, we use road network spatial hierarchy to organise road segments of *route* to form a delay-monitor tree (DM-Tree), where we apply continuous monitoring until the current route is updated. As illustrated in Fig. 5, all road segments of the current route appear on the DM-Tree as leaf nodes, and non-leaf nodes (i.e., sub-graphs containing at least one road segment) are organised as a tree according to their spatial hierarchical relationship. For each non-leaf node n, we use $n \rightarrow SG$ to denote its corresponding sub-graph on visible graph, and $PT(n, C)$ to denote the travel time on the partial path inside $n \rightarrow SG$ based on road condition C. We update the partial path belonging to a non-leaf if its travel time is excessively increased: Assume the current route is planned based on road condition C_0, we continuously capture the non-leaf node n with $PT(n, C)/PT(n, C_0)$ greater

than a threshold, and update the partial path in $n \rightarrow SG$ using A* algorithm on the current visible graph.

To cope with the high dynamics of traffic, we also keep tackling the road condition changes on the visible graph, with a frequency lower than route monitoring. For each sub-graphs G_i in the visible graph, we continuously compute its average speed $spd(G_i,\ C)$ under latest road condition C, and measure the need of re-planning at this moment from its local perspective as $score(G_i,\ C)$

$$score(G_i, C) = \begin{cases} \frac{spd(G_i,C)-spd(G_i,C_0)}{spd(G_i,C_0)} & route \cap G_i = \phi \\ \frac{spd(G_i,C_0)-spd(G_i,C)}{spd(G_i,C_0)} & route \cap G_i \neq \phi \end{cases}$$

where C_0 is the road condition when route was planned in Sect. 6.2. The score equals to the ratio of speed increased (to original average speed) for sub-graphs intersecting with the current route, or the decreased speed ratio for other sub-graphs. The reason is that faster paths are likely to be found if the average speed on the former regions grows while that value on the latter regions decreases. Therefore, a higher value of $score(G_i,\ C)$ means the confidence of vote on route re-planning from the view of sub-graph G_i.

Then, we compute the average score $SC = \frac{\sum_{G \in VG} score(G,C)}{|\{G \in VG\}|}$ of all sub-graphs in the visible graph, where SC denote the need of route re-planning from the global perspective. If SC is larger than a threshold, we consider most sub-graphs have agreed with path adjustment, and thus simply re-plan the whole route from current position to destination using the HRS mechanism. Otherwise, we only focus on particular sub-graphs where road condition is greatly changed. In such a sub-graph SG_i, assume that v_j and v_k are the first and last vertex of part of route in SG_i respectively, we adjust the partial path from v_j to v_k using A* algorithm in the search space of SG_i to ensure local optimization.

7 Experimental evaluation

In this section, we experimentally evaluate the techniques that have been proposed in Sects. 5 and 6. Specifically, different approaches (including existing ones) are compared based on three criteria: (1) the number of examined edges in a single query processing, where we do not consider road condition changes; (2) search accuracy, i.e., the actual travel time of the users along the whole trip; (3) the total time of the continuous query processing along the whole trip (consider dynamics of road condition).

7.1 Experimental setup

Our experiments were conducted on a HP Compaq 8180 Elite (i5 650) computer with 2-core CPUs at 3.2 and 1.12 GHz, 4GB RAM and running Windows XP operating system. We use a road network that contains 226,238 directed edges (road segments) and 171,187 vertices (intersections). The network graph corresponds to part of the road network in the city of Beijing, China, and Fig. 6 describes the spatial distribution

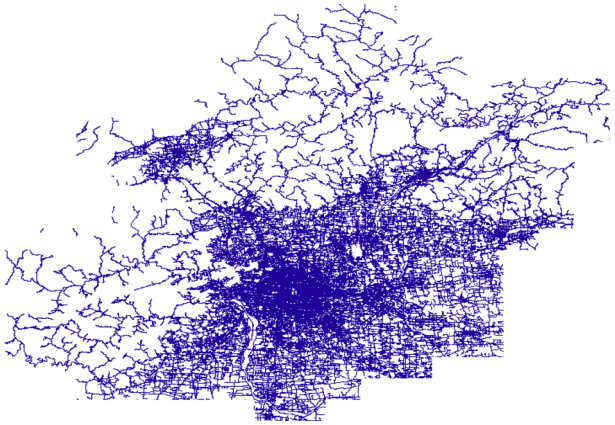

Fig. 6 Sketch of Beijing road network

of our map data. All edges and road segments they are stored with spatial grid indexes. As the actual speeds of all road segments are not available, we use both real (extracted from the Taxi trajectory data that can be downloaded at web, only densely distributed on major road segments) and simulation speed information in this experiment.

Experiments are based on 200 test cases, including 100 cases with constant road speed (Group 1) and 100 cases with dynamic road speed (Group 2). In the former cases, we evaluate the performance of single route search, i.e. the response time of different strategies (A*, IRS, HRS) and accuracy in different speed patterns, to tell their feedback efficiency to users; For the latter case, we focus on the performance of continuous route planning, including the number of recalculations, total processing times at server side, and total travel times of the customer under the planed route.

7.2 Performance study

7.2.1 Pre-processing cost

In the pre-processing step, as the map data we use have the hierarchical information of each road segment, we only need to partition it to sub-regions and construct a hierarchical index structure of the traffic network. In reality the two operations are process in a holistic way. By default, we create spatial grid index on the road network which consists over 0.2 million edges and around 0,17 vertexes in a parallel programing mode, and the total processing cost is as fast as no more than 1 min. Afterwards, we conduct the network partition and hierarchical indexing creation in a holistic way. Through our experiment, the road network can be partitioned based on the existing hierarchical information of road segments within 13 min, in which the time cost of creating hierarchical road index is around 4 min.

7.2.2 Performances on traditional query processing

We first test the performance of A*, IRS and HRS strategies on normal query processing, where continuous querying is not considered. That means, we only focus on the

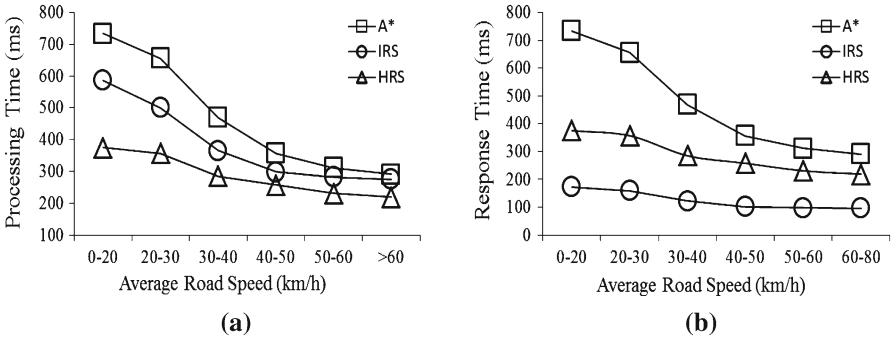

Fig. 7 Efficiency versus average road speed. **a** Total processing time, **b** response time

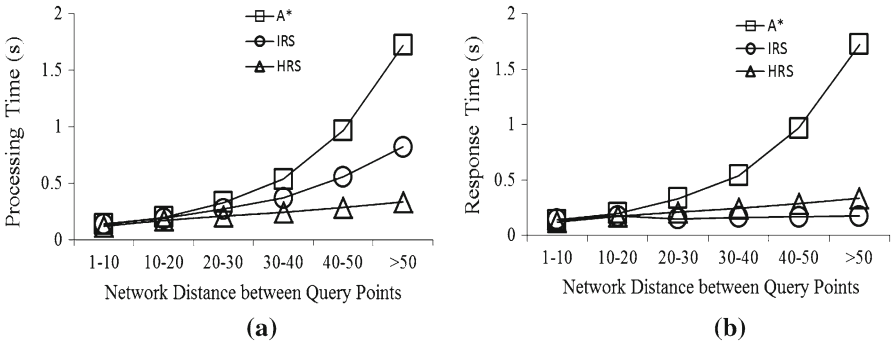

Fig. 8 Efficiency versus query length. **a** Total processing time, **b** response time

static environment here, without taking the dynamics of road condition into consideration. Therefore the test cases in Group 1 are used. The queries we use in experiment are long distance queries, i.e. strictly more than 20 kilometres.

Comparisons of the processing time of queries and response time to users are plotted as Fig. 7a, b respectively. According to Fig. 7a, we can observe that A* has the greatest computational overhead while the HRS algorithm has the least cost. This phenomenon confirms with the efforts made by IRS and HRS to reduce search space of query processing. This figure also shows that the variation of query processing time of the three strategies tend to be small when the road condition becomes good. It is due to the fact classical shortest path search algorithm like A* cannot make use of the speed information, which have less great impact for cases that road condition is good.

Figure 7b compares three algorithms regarding to the required response time to user, which is an important performance indicator of route planning services. From this figure we can see that the IRS algorithm has the minimum response time while A* is the worst. The reason here is that the result of query processing through IRS is a partial path, rather than a whole path.

We also evaluate the efficiency of three algorithms in different query lengths in Fig. 8. In general, the processing time and response time are similar for short distance queries, but they tend to vary significantly for long distance queries. In Fig. 8a, we can

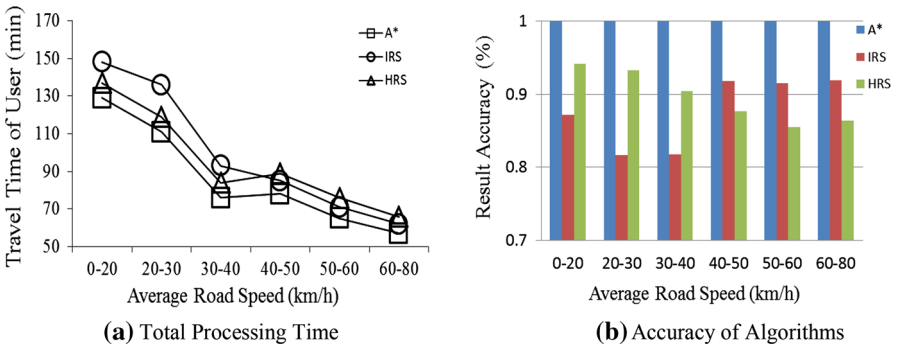

Fig. 9 Accuracy versus average road speed. **a** Total processing time. **b** accuracy of algorithms

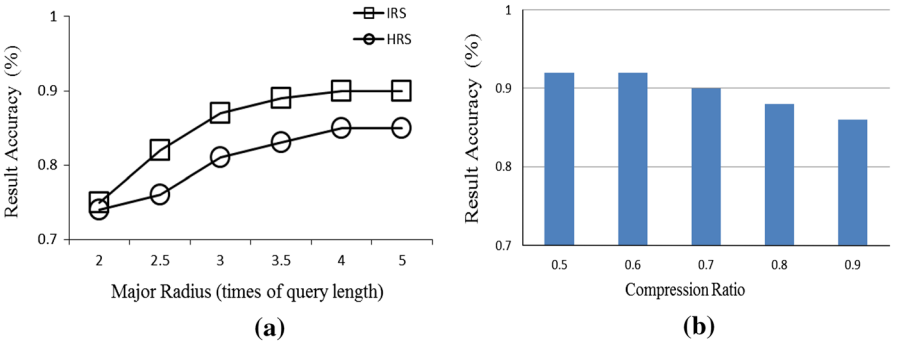

Fig. 10 Accuracy versus ellipse size and network compression ratio. **a** Accuracy versus ellipse size, **b** response time

observe that the growth of query length incurs as the processing time increases accordingly. Clearly, the A* algorithm has the greatest increase rate, while the efficiency of HRS algorithm is the most stable. The main reason is that HRS adopts planning method based on arterial roads, such that long distance queries can be processed on a compressed road network in HRS. In contrast, the efficiency of road network expansion based methods like A* and IRS are more sensitive to the query length. Figure 8b shows the response time of the proposed algorithms. It is observed that, IRS is the one having the minimum response time due to its partial result feedback nature.

Also, comparisons on the accuracy performance of different query processing algorithms are shown as Fig. 9a, b respectively. We know A* algorithm returns the optimal route to user, hence incurs the least travel time of user according to Fig. 9 a, and having 100 % accuracy as shown inFig. 9b. In general, IRS and HRS also has relatively good accuracy around 90 %. We can also observe that the IRS algorithm is superior to HRS algorithm when the road condition is good. In contrast, the HRS algorithm outperforms IRS in cases when the traffic becomes bad. This phenomenon can be explain by the fact major roads are more reliable in rush hours, but more efficient paths can be easily detected when the road condition becomes good because of their less distance. Therefore, we can select IRS and HRS properly in use based on the real time road condition for route planning.

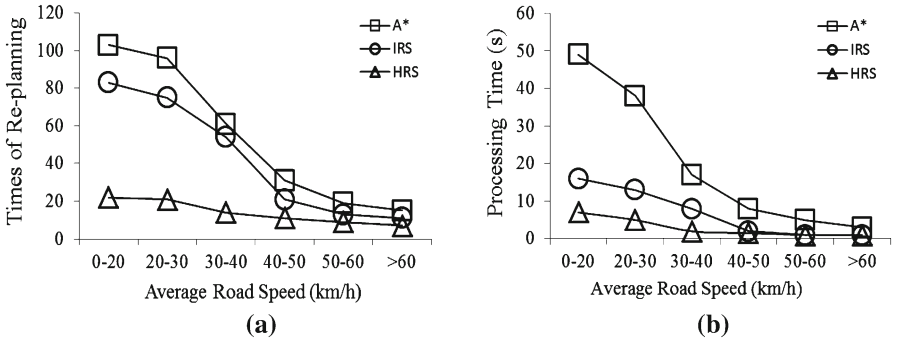

Fig. 11 Efficiency comparison versus average road speed. **a** Times of route updates, **b** total processing time of continuous querying

To better understand the parameters affecting the accuracy of planning, we further verify the accuracy of planned route in different settings of basic ellipse major radius and road network compression ratio. In Fig. 10a, we compare the accuracy of IRS and HRS algorithms with different settings of basic ellipse major radius (the value is the times of major radius to network distance between start and end query points), and it is obvious that the size of basic radius affects IRS greater than HRS, which can be explained as that IRS is a local optimum based solution. Figure 10b indicates the impact of HRS network compression ratio to route planning accuracy, and we can see HRS performs fairly well even when the compression ratio is as high as 0.9 (i.e., the ratio of search graph size to the network graph size after basic graph reduction).

7.2.3 Performances on continuous query processing

To support the traffic aware navigation, we further evaluate and compare the performance of the above three algorithms on continuous query processing. The test cases in Group 2 are used, where we update the road condition in 1–2 min, and note that the average road speed before update and that after update strictly fall in the same value range (e.g. 20–30 km).

Figure 11a shows the comparison results on the times of re-planning in continuous route search. It is an important factor because each route update has the tune-in cost (to mobile phone or GPS) and data transmission cost. From this figure we can see that the HRS algorithm incurs least route updates. It is also shown that both A* based solution and IRS algorithm are not robust to the dynamics of road condition as they require the planned route to be updated frequently when the traffic condition is poor.

Figure 11b is the experimental results of total processing time of the route queries. Specifically, for each time of the route updates in Fig. 11a, we have additional processing cost, and Fig. 11b shows the total time cost of each continuous query processing. We can easily observe that the HRS algorithm has the least total time cost of query processing. In contrast, the A* based solution has the maximal total time cost. Though

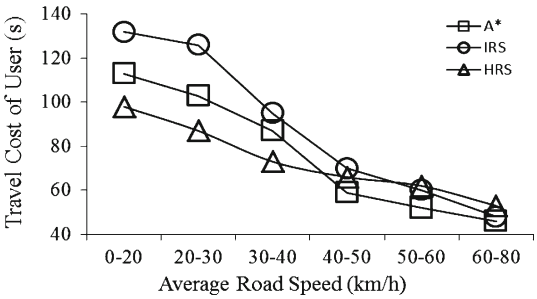

Fig. 12 Travel cost of users versus average road speed

IRS results in a large number of re-calculations, its total time cost is also small due to its cheap cost of each query processing (as it finds a partial path).

In addition, we use Figure 12 to illustrate the accuracy of continuous query processing (i.e. travel cost of user based on the planned route) of the above algorithms. From this figure we know that the planned routes returned by three solutions have similar travel cost when the traffic is good, and HRS is slightly worse than other two in such cases because it tend to stick on major roads, so may have longer distance to travel. However when the road condition is poor, the HRS provides the best continuous route to users. The A* is the worse than the HRS algorithm when the traffic is heavy, but it turns out to be the best when the road condition becomes good. The reason lies here is that the A* algorithm does not sensitive to the speed information. Meanwhile, the IRS leads to the greatest travel cost of users because it is a local optimization algorithm for finding a suitable partial path, and is thus not robust to the dynamics of road condition.

To sum up, experiments show that IRS and HRS returns near-optimal route for single route query (without considering road condition changes), and IRS is more suitable for scenarios when traffic is good, while HRS is more effective in the rush hours. As for the continuous route planning, we should select the HRS for most cases except when the road condition is extremely good, where we can simply use A* (only update when the travel time of current route increases sharply) because the speed information is not vital anymore when it becomes stable.

8 Conclusions

Traffic navigation is a basic service for people's travel nowadays. However, most studies focus on route planning on static road network, without considering the high dynamics of road network, which can significantly affect the performance of route search. In this paper, we present two traffic aware route planning strategies based on the hierarchical road network, with a guarantee that route planning is conducted in a small search space and that unnecessary re-calculations caused by the dynamics of road conditions can be avoided. Finally, experiments with real dataset demonstrate the effectiveness and efficiency of our proposed algorithms.

In the future, we will extract out and then use congestion evolution patterns from traffic data to further improve the performance of route planning, and construct a system based on the algorithms proposed in this paper.

Acknowledgments The research work reported in this paper was partly supported by the NSFC projects under grant numbers 61232006, 91124001, 61379033, 61003049, 61073061, and 61202064, the Fundamental Research Funds for the Central Universities under Grant 2013QNA5020, the Key Project of Zhejiang University Excellent Young Teacher Fund (Zijin Plan), and ARC discovery project under grant number DP120102627.

References

1. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: Proceedings of the Workshop on Algorithm Engineering and Experiments (2007)
2. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. In: Proceedings of the 7th International Workshop on Experimental Algorithms, pp. 303–318 (2008)
3. Chen, S., Tu, Y.C., Xia, Y.: Performance analysis of a dual-tree algorithm for computing spatial distance histograms. VLDB J. **20**(4), 471–494 (2011)
4. Demiyurek, U., Kashani, F.B., Shahabi, C., Ranganathan, A.: Online computation of fastest path in time-dependent spatial networks. In: Proceedings of the 12th International Symposium on Advances in Spatial and Temporal Databases, pp. 92–111 (2011)
5. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: Proceedings of the 23rd International Conference on Data, Engineering, pp. 796–805 (2007)
6. Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: Proceedings of the 11th International Conference on Extending Database Technology, pp. 205–216 (2008)
7. Gao, J., Jin, R., Zhou, J., Yu, J.X., Jiang, X., Wang, T.: Relational approach for shortest path discovery over large graphs. Proc. VLDB Endow. **5**(4), 358–369 (2011)
8. Gao, J., Qiu, H., Jiang, X., Wang, T., Yang, D.: Fast top-*k* simple shortest paths discovery in graphs. In: Proceedings of the 19th ACM Conference on Information and Knowledge Management, pp. 509–518 (2010)
9. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Faster and simpler hierarchical routing in road networks. In: Proceedings of the 7th International Workshop on Experimental Algorithms, pp. 319–333 (2008)
10. Goldberg A.V., Harrelson, C.: Computing the shortest path: a search meets graph theory. In: Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 156–165 (2005)
11. 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, pp. 794–805 (2007)
12. Gunturi, V.M.V., Nunes, E., Yang, K.S., Shekhar, S.: A critical-time-point approach to all-start-time lagrangian shortest paths: a summary of results. In: Proceedings of the 12th International Symposium on Advances in Spatial and Temporal Databases, pp. 74–91 (2011)
13. Hua M., Pei, J.: Probabilistic path queries in road networks: traffic uncertainty aware path selection. In: Proceedings of the 13th International Conference on Extending Database Technology, pp. 347–358 (2010)
14. Huang, B., Wu, Q., Zhan, F.B.: A shortest path algorithm with novel heuristics for dynamic transportation networks. Int. J. Geogr. Inf. Sci. **21**(6), 625–644 (2007)
15. Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on a road network with speed patterns. in Proceedings of the 22nd International Conference on Data, Engineering (2006)
16. Koenig, S., Likhachev, M., Furcy, D.: Lifelong planning A. Artif. Intell. **155**(1–2), 93–146 (2004)
17. Kriegel, H.-P., Renz, M., Schubert, M.: Route skyline queries: a multi-preference path planning approach. In: Proceedings of the 26th International Conference on Data, Engineering, pp. 261–272 (2010)
18. Li, F., Chen, D., Hadjieleftheriou, M., Kollios, G., Teng, S.-H.: On trip planning queries in spatial databases. In: Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases, pp. 273–290 (2005)
19. Likhachev, M., Ferguson, D.I., Gordon, G.J., Stentz, A., Thrun, S.: Anytime dynamic A*: an anytime, replanning algorithm. In: Proceedings of the 5th International Conference on Automated Planning and Scheduling, pp. 262–271 (2005)

20. Malviya, N., Madden, S., Bhattacharya, A.: A continuous query system for dynamic route planning. In: Proceedings of the 27th International Conference on Data, Engineering, pp. 792–803 (2011)
21. Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous nearest neighbor monitoring in road networks. In: Proceedings of the 32nd International Conference on Very Large Data, Bases, pp. 43–54 (2006)
22. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, pp. 867–876 (2009)
23. Rice, M.N., Tsotras, V.J.: Graph indexing of road networks for shortest path queries with label restrictions. Proc. VLDB Endow. **4**(2), 69–80 (2010)
24. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 43–54 (2008)
25. Sanders P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: Proceedings of the 13th Annual European Symposium on Algorithms, pp. 568–579 (2005)
26. Sankaranarayanan, J., Samet, H., Alborzi, H.: Path oracles for spatial networks. Proc. VLDB Endow. **2**(1), 1210–1221 (2009)
27. Xiao, Y., Wu, W., Pei, J., Wang, W., He, Z.: Efficiently indexing shortest paths by exploiting symmetry in graphs. In: Proceedings of the 12th International Conference on Extending Database Technology, pp. 493–504 (2009)
28. Xu, J., Guo, L., Ding, Z., Sun, X., Liu, C.: Traffic aware route planning in dynamic road networks. In: Proceedings of the 17th International Conference on Database Systems for Advanced Applications, pp. 576–591 (2012)
29. Zeng, W., Church, R.L.: Finding shortest paths on real road networks: the case for A*. Int. J. Geogr. Inf. Sci. **23**(4), 531–543 (2009)