2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress

General Time-Dependent Sequenced Route Queries in Road Networks

Mohammad H. Ahmadi
Department of Electrical Engineering
Allameh Dehkhoda University, Iran
Email: mohammad.hos.ahmadi@gmail.com

Vahid Haghighatdoost
Department of Computer Engineering
Shahed University, Iran
Email: haghighatdoost@shahed.ac.ir

***Abstract*—Spatial databases have been an active area of research over years. In this paper, we study how to answer the *General Time-Dependent Sequenced Route* queries. Given the origin and destination of a user over a time-dependent road network graph, an ordered list of categories of interests and a departure time interval, our goal is to find the minimum travel time path along with the best departure time that minimizes the total travel time from the source location to the given destination passing through a sequence of points of interests belonging to each of the specified categories of interest. The challenge of this problem is the added complexity to the optimal sequenced route queries, where we assume that first the road network is time-dependent, and secondly the user defines a departure time interval instead of one single departure time instance. For processing general time-dependent sequenced route queries, we propose an efficient solution as Continuous-Time Sequenced Route approach. Our proposed approach traverses the road network based on A^*-search paradigm equipped with an efficient heuristic function, for shrinking the search space. Extensive experiments are conducted to verify the efficiency of our proposed approach.**

I. Introduction

Nowadays, the location-based services allow users to search nearby facilities and find the shortest route between two locations. Given a source location and a destination, as well as a sequence of Categories of Interests (COIs) that the user is interested to visit (e.g., restaurants, hospitals), an OSR query [7] finds the optimal route from the source location to the destination passing through exactly one Point of Interest (POI) from each COI according to the pre-defined sequence. In this paper, we study the generalized form of OSR queries on time dependent road networks, calling it *General Time-Dependent Sequenced Route* (GTDSR) queries. Contrary to static road networks, where the weight of edges are constant, in this query, we assume the road network as a time-dependent road network. in the time-dependent (i.e., typical) road networks the weight/travel time of edges depends on the traffic congestion on the edges and hence is a function of the time of the day.

Let us consider a sample scenario, where a user who is currently at his work place,v_s, is willing to leave his work place in time interval [4pm,4:30pm], and visit a bank branch and a supermarket before going to home v_d. Assuming that bank and supermarket categories denoted by C_B and C_S, then the user issues a general time-dependent sequenced route query as $GTDR = (v_s, v_d, \{C_B, C_S\}$,[4pm,4:30pm]) to find the path with the minimum travel time along with the best departure time from his work place towards his home.

OTDSR queries [2] can be considered as a specific case of our problem. In terms of problem definition, contrary to our problem, the OTDSR query considers one single departing time from source location. In most situations, on the other hand, the users may look for the best departure time instance from a departure time interval in order to avoid being stuck in traffic. We take this characteristic as an intrinsic part of our problem and propose an efficient technique for solving GTDSR queries as Continuous-Time Sequenced Route (CTSR) approach. The basic idea behind the CTSR approach is that, we iteratively generate and examine different partial routes from source location, where in each step, a route incurring the minimum lower bound total travel time will be expanded. Furthermore, in CTSR approach, while generating different possible candidate partial routes, we also specify a specific departure time sub-interval for each new generated partial route, in which for that sub-interval time, that partial route is considered the most promising route from the given source location among all possible partial routes generated so far.

Contributions of this paper: (1) We propose and formalize for the first time the GTDSR queries in this paper. (2) We present CTSR approach for processing GTDSR queries. (3) We conducted extensive performance studies to evaluate the performance of proposed approaches in terms of processing time and number of expanded vertices.

II. Problem Definition

We model the road network as a time-dependent directed graph $G_T = (V, E, W)$, where V is the set of road network branch points (intersections), E is the set of road segments. W is the set of weight functions, where the weight of every edge $(v_i, v_j) \in E$ in time instance t is defined based on function $w_{i,j}(t) \in W$. To be more specific, $w_{i,j}(t)$ specifies the travel time from v_i to v_j via edge (v_i, v_j), departing v_i at time instance t.

Definition II.1. Given a time-dependent road network graph $G^T = (V, E, W)$, a source location $v_s \in V$, a destination $v_d \in V$, a sequence of COIs as $C = (C_1, C_2, ..., C_n)$ that the user is interested to visit. Assuming that the user

978-1-5386-1956-8/17 $31.00 © 2017 IEEE
DOI 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.158

departures the source location in time interval T, the goal of *General Time-Dependent Sequenced Route query* denoted by $GTDSR(v_s, v_d, C, T)$ is to find: 1) the best departure time instance t^* from source location v_s, and 2) a trip with the smallest travel time from v_s to v_d passing through exactly one POI from each COI.

III. Related Work

In [6], Li et al. proposed solutions for Trip Planning Queries (TPQ). In a TPQ, the user specifies a set of Category of Interests (COIs) and asks for the optimal route (with minimum distance) from her starting location to a specified destination which passes through exactly one POI of each COI. Sharifzadeh et al. [7], [8] presented OSR queries where the user asks for an optimal route from her starting location and passing through a number of POIs (each with a different type) in a particular order (sequence) imposed on all the types of POIs to be visited. The R-LORD algorithm of Sharifzadeh et al. [7] performs range queries to filter out the points that cannot possibly be part of the optimal route by utilizing an R-tree index structure [4], and subsequently builds the optimal route in reverse sequence(i.e., from ending to the starting point). In [8], Sharifzadeh et al. proposed a pre-computation approach for processing OSR query in both vector and metric spaces, in which firstly the Voronoi diagram [1] of solution space is constructed based on geometric properties of the solution space, and then the OSR route is obtained by recursively accessing the pre-constructed Voronoi diagram. In [9], Sung et al. investigated the problem of finding the shortest path from a given source location to a destination for a given specific departure time (not a starting-time interval). In the proposed approach in [9], the given query is transformed to a shortest path problem in a static graph with constant edge delays. In [5], kanoulas et al., proposed an A^* based algorithm for time-dependent shortest path (TDSP) problem. The proposed approach in [5] incrementally creates the partial paths and stores them in a priority queue Q. Meanwhile assuming p_k as a path from source location v_s to v_k, we assign a heuristic cost to each path p_k in Q as $f_{pk}(t) = g_{p_k}(t) + d_{k,e} - t$; where $g_{p_k}(t)$ is the arrival time to v_k, and $d_{k,e}$ is the lower bound estimate of the travel time from v_k to v_d. Totally $f_{pk}(t)$ gives us an estimation of travel time from v_s to v_d along path p_k, where the departure time from v_s is t. In each step, the algorithm picks the path p_i with the smallest $f_{pi}(t)$ from Q to be expanded by visiting all neighboring vertices of the last vertex in p_i. This proposes will be repeated until the picked up path from Q visits the destination v_d.

In [3], Ding et al. studied the generalized form of shortest paths over large graphs, where given a source location v_s and a destination v_d over a time-dependent graph G_T and a departure time interval, our goal is to find the fastest path with the best departure time instance. The proposed approach by Ding et al.[3] involves two steps. In the first step, we compute the earliest arrival time to different vertices from v_s regarding to different departing times. And then in the second step, among all candidates trips from v_s to v_d the optimal trip incurring the smallest travel time is returned.

In [2], Costa et al. studied the problem of OTDSR [2] queries as a specific case of our problem GTDSR queries. Contrary to GTDSR queries, in the OTDSR queries the user inputs only one single departure time instance from source location instead of departure time interval. The proposed approach by Costa et al. [2] for solving OTDSR queries is fundamentally an extension to Dijkstra algorithm which uses an A^* search to guide the network expansion so that vertices with more potential are examined first. For estimating the potential of a vertex v, a function $f(v) = g(v) + h(v)$ is used, where $g(v)$ is a lower bound estimate of the travel time to pass through all the COIs and to reach to the destination. The smaller the value given by the sum of the current cost to a vertex plus the heuristic function value for it, the greater its potential.

IV. Proposed Approaches

In order to reduce the execution time of the GTDSR queries, a pre-processing step is performed in CTSR approach. Our goal from pre-processing step, is to calculate bounds to guide the expansion of vertices, while applying the A*-based Dijkstra algorithm. The applied pre-computations in this paper are based on proposed pre-computations in [2].

In the pre-processing step, we need to calculate the optimistic distance estimates to reach the closest POI in each COI, from each $v \in V$ in $G^T = (V, E)$. In order to calculate these lower bounds, we construct a lower bound graph $\underline{G}$. Given a time-dependent graph G^T, $\underline{G}$ is a graph that has the same set of vertices V and edges E as G but the cost of an edge $\underline{G}$ is the minimum possible weight (i.e., the smallest required time to traverse the edge). Based on this definition, $\underline{G}$ can be considered an optimistic non-time dependent version of G. After computing $\underline{G}$, we pre-compute the shortest network path in $\underline{G}$ from each vertex to its nearest neighbor from different categories. In this paper by $L(v, C_i)$ we denote the smallest network distance from $v \in V$ to its nearest *point* from category C_i based on $\underline{G}$.

The basic idea behind of CTSR approach is that, we iteratively generate and store different partial routes from source location, ordered based on estimated lower bound travel time towards destination. As an iterative process, in each step, a route incurring the minimum lower bound total travel time will be retrieved and expanded. Furthermore, in

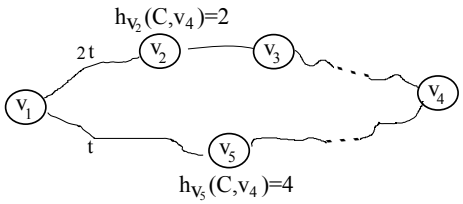


Figure 1: A sample road network.

Algorithm 1: *CTSR Approach*

Input: A starting point v_s, a destination v_d, a departure time interval $T = [t_s, t_d]$, and a sequence of COIs $C = (C_1, C_2, ..., C_m)$ that must be visited

Output: The optimal trip p^* from v_s to v_d, and the optimal departure time $t^* \in T$

$\{ancestor(.,.), T^*\} \leftarrow$ Time/Path Refinement(G_T, v_s, v_d, T)
$p^* \leftarrow$Path Selection$(v_s, v_d, ancestor(.,.), T^*)$
$t^* \leftarrow$ Time Selection(p^*, T^*)
return (p^*, t^*)

CTSR approach, while generating different possible candidate partial routes, we also specify a specific departure time sub-interval for each new generated partial route, in which for that sub-interval time, that partial route is considered the most promising route from the given source location among all possible partial routes generated so far. Let us assume p_i as a partial path with departure time T^s_{dep} from source location v_s, ending at v_i with total travel time TT_{v_i}. We use function $TT^{lb}_{v_i} = TT_{v_i} + h_{v_i}(C_{req}, v_d)$ to estimate the lower bound travel time of p_i, in which $h_i(C_{req}, v_d)$ indicates the lower bound travel time from v_i towards the destination v_d passing through at least one POI from each COI belonging to $C_{req} \subset C$. The smaller the value given by the $TT^{lb}_{v_i}$ (as the lower bound travel time for path p_i passing through vertex v_i and visiting at least one POI from required COIs), the greater its potential. We use the provided heuristic in [2] to estimate the value of $h_{v_i}(C_{req}, v_d)$, C_{req}=$(C_j, C_{j+1}, ..., C_m)$, as following:

$$h_{v_i}(C_{req}, v_d) = max\{L(v_i, C_j), ..., L(v_i, C_m), L(v_i, v_d)\} \quad (1)$$

where $L(v_i, C_j)$ is the lower bound travel time from v_i to its nearest POI belonging to C_j, pre-computed in pre-processing step. Similarly the minimum travel time from v_i to v_d can be computed as following: once v_d is given as a query input, we can compute the lower bound travel time from every vertex $v_i \in V$ to v_d, by running Dijkstra's algorithm (one-to-all shortest paths) from v_d in the reverse graph of $\underline{G}$, denoted by $\underline{G}^R$, only once.

We explain the basic idea of CTSR approach by using example of Figure 1. In this figure, we assume that the source and target locations are v_1 and v_4, respectively. The user is interested to visit one POI from COI C, where he/she defines the departure time interval from source location v_1 as T=[0,6]. In this example, the travel time from v_1 towards v_2 and v_5 is $2\,t$ and t, respectively. Furthermore, supposing that non of vertices v_1, v_2 and v_5 are POIs, we assume the lower bound travel time from v_2 and v_5 towards destination v_4 via visiting one POI from C as 2 and 4, respectively (i.e., $h_{v_2}(C, v_4) = 2$ and $h_{v_5}(C, v_4) = 4$).

We start traversing the road network from source location v_1. In the first step, two partial routes (v_1, v_2) and (v_1, v_5) are generated and examined, which incur the lower bound travel times $TT^{lb}_{v_2}(t)$=$2\,t + 2$ and $TT^{lb}_{v_5}(t)$=$t + 4$, respectively, for visiting one POI from C towards destination v_4. To be more specific, the lower bound travel time from v_1 towards v_4 via vertices v_2 and v_5 based on departure time interval [0,6] is defined as the ranges [2,14] and [4,10], respectively). By comparing $TT^{lb}_{v_2}(t)$ and $TT^{lb}_{v_5}(t)$, we observe that the minimum lower bound travel time of v_2 is smaller than the minimum lower bound travel time of v_5 as 4. Consequently, the partial route (v_1, v_2) is selected to be expanded in the next step by

Algorithm 2: *Time/Path Refinement in CTSR approach*

Input: A starting point v_s, a destination v_d, a departure time interval $T = [t_s, t_d]$, and a sequence of COIs $C = (C_1, C_2, ..., C_m)$ that must be visited

Output: The fastest possible trip p^* from v_s to v_d, and the optimal departure time $t^* \in T$

$\underline{G}^R \leftarrow Reverse(\underline{G})$
$dist[] \leftarrow Dijkstra(\underline{G}^R, v_d)$
$T^s_{dep} \leftarrow [t_s, t_d]$
$TT^{lb}_{v_s}(t) = h_{v_s}(C, v_d), \forall t \in T^s_{dep}$
$R_{v_s} \leftarrow \varnothing$;
$TT_{v_s}(t) \leftarrow 0, \forall t \in T^s_{dep}$
$Q \leftarrow \varnothing$
$AT_{v_s}(t) \leftarrow t, \forall t \in T^s_{dep}$
$T^{v_s}_{dep}(t) \leftarrow t, \forall t \in T^s_{dep}$
for $\forall v_n \in adjacency(v_s)$ **do**
  $TT_{v_n}(t) \leftarrow TT_{v_s}(t) + w(v_s, v_n)(T^{v_s}_{dep}(t)), \forall t \in T^s_{dep}$
  $AT_{v_n}(t) \leftarrow T^{v_s}_{dep}(t) + w(v_s, v_n)(T^{v_s}_{dep}(t)), \forall t \in T^s_{dep}$
  $TT^{lb}_{v_n}(t) \leftarrow TT_{v_n}(t) + h_{v_n}([C_{next}, ..., C_m], v_d), \forall t \in T^s_{dep}$
  $T^{v_n}_{dep}(t) \leftarrow AT_{v_n}(t), \forall t \in T^s_{dep}$
  if $v_n \in C[1]$ **then**
    $next \leftarrow 1$
    $R_{v_i}[next] \leftarrow v_n$
    $R_{v_n} \leftarrow R_{v_i}$
    $T^{v_n}_{dep}(t) \leftarrow T^{v_n}_{dep}(t) + spent[next], \forall t \in T^s_{dep}$
  $En(Q, [v_n, T^s_{dep}, AT_{v_n}(t), TT_{v_n}(t), TT^{lb}_{v_n}(t), R_{v_n}, T^{v_n}_{dep}(t)])$
  $ancestor(v_n, T^s_{dep}) \leftarrow v_s$
while $|Q| > 2$ **do**
  $[v_i, T_{v_i}, AT_{v_i}(t), TT_{v_i}(t), TT^{lb}_{v_i}(t), R_{v_i}, T^{v_i}_{dep}(t)] \leftarrow D(Q)$
  if $v_i = v_d$ *and* $|R_{v_i}| = |C|$ **then**
    return the set of ancestor functions $ancestor(.,.)$ accompanied with T^s_{dep}
  $[v_l, T_{v_l}, AT_{v_l}(t), TT_{v_l}(t), TT^{lb}_{v_l}(t), R_{v_l}, T^{v_l}_{dep}(t)] \leftarrow H(Q)$
  Determine $T_{v_n} \in T_{v_i}$, where $TT^{lb}_{v_i}(t) \leq \min_{\forall t' \in T_{v_l}} \{TT^{lb}_{v_j}(t')\}, \forall t \in T_{v_n}$
  for $\forall v_n \in adjacency(v_i)$ **do**
    $TT_{v_n}(t) \leftarrow TT_{v_i}(t) + w(v_i, v_n)(T^{v_i}_{dep}), \forall t \in T_{v_n}$
    $AT_{v_n}(t) \leftarrow AT_{v_i}(t) + w(v_i, v_n)(T^{v_i}_{dep}), \forall t \in T_{v_n}$
    $TT^{lb}_{v_n}(t) \leftarrow TT_{v_n}(t) + h_{v_n}([C_{next}, ..., C_m], v_d), \forall t \in T_{v_n}$
    $T^{v_n}_{dep}(t) \leftarrow AT_{v_n}(t), \forall t \in T_{v_n}$
    if $v_n \in C[|R_{v_i}| + 1]$ **then**
      $next \leftarrow |R_{v_i}| + 1$
      $R_{v_i}[next] \leftarrow v_n$
      $R_{v_n} \leftarrow R_{v_i}$
      $T^{v_n}_{dep}(t) \leftarrow T^{v_n}_{dep}(t) + spent[next], \forall t \in T_{v_n}$
    if $\exists [v_n, t, AT, TT, LB, R, T_{dep}] \in Q$, *where* $t \in T_{v_n}$ **then**
      if $TT_{v_n}(t) < TT$ *and* $|R| < |R_{v_n}|$ **then**
        remove $(v_n, t, AT, TT, LB, R, T_{dep})$ from Q
        $En(Q, [v_n, T_{v_n}, AT_{v_n}(t), TT_{v_n}(t), TT^{lb}_{v_n}(t), R_{v_n}, T^{v_n}_{dep}(t)])$
        $ancestor(v_n, T_{v_n}) \leftarrow v_i$
      else if $TT_{v_n}(t) > TT$ *and* $|R| > |R_{v_n}|$ **then**
        The new generated tuple is not eligible to be inserted to Q
      else
        $En(Q, [v_n, T_{v_n}, AT_{v_n}(t), TT_{v_n}(t), TT^{lb}_{v_n}(t), R_{v_n}, T^{v_n}_{dep}(t)])$
        $ancestor(v_n, T_{v_n}) \leftarrow v_i$
    else
      $En(Q, [v_n, T_{v_n}, AT_{v_n}(t), TT_{v_n}(t), TT^{lb}_{v_n}(t), R_{v_n}, T^{v_n}_{dep}(t)])$
      $ancestor(v_n, T_{v_n}) \leftarrow v_i$

Algorithm 3: *Path-Selection*

Input: A starting point v_s, a destination v_d, the optimal time interval $T^* \in T$, and the set of ancestor functions $ancestor(.,.)$
Output: The optimal trip p^*

```
1 v_j ← v_d
2 p* ← ∅
3 while v_j ≠ v_s do
4     v_i ← ansector(v_j, T*)
5     p* ← (v_i, v_j).p*
6     v_j ← v_i
7 return p*
```

Algorithm 4: *Time-Refinement*

Input: A path p^* from v_s towards v_d, and a departure time interval T^*
Output: The optimal departure time $t^* \in T^*$

```
1 t* ← argmin_{t∈T*}{TT_{v_d}(t)} // based on path p*
2 return t*
```

appending the vertex v_3 as a neighbor of v_2. Meanwhile, the main question is that the partial route (v_1, v_2, v_3) must be examined towards which departure time interval. Now we explain the answer of this question.

By comparing $TT^{lb}_{v_2}(t) = 2t + 2$ and $TT^{lb}_{v_5}(t) = t + 4$, it is not hard to see that the partial route (v_1, v_2) incurs smaller lower bound travel time in comparison to (v_1, v_5) in departure time sub-interval $[0, 2]$. In other words, assuming $[0, 2]$ as the departing time interval from source location v_1, then the partial route (v_1, v_2) is considered as the more promising route in comparison to partial route (v_1, v_5), due to requiring smaller lower bound travel time. Based on the aforementioned example, the basic idea behind of CTSR approach is that, for a given departure time interval T_{dep}, assuming v_i as a vertex which incurs the minimum lower bound travel time in comparison to other vertices, then we traverse the neighboring vertices of v_i towards a specific departure time sub-interval $T_i \in T_{dep}$, guaranteeing that in T_i the vertex v_i (i.e., the corresponding partial rout ending at v_i) incurs the smallest lower bound travel time in compassion to other vertices (i.e., the existing partial routes) in the road network.

Algorithm 1 illustrates the pseudo-code of CTSR approach. Processing GTDSR approach based on CTSR approach contains three phases: *Time/Path Refinement Path Selection* 3 and *Time Selection*, illustrated in Algorithms 2, 3 and 4, respectively. The output of *Time/Path Refinement* is: a set of ancestor functions and the optimal departure time sub-interval T^*. By using the output of *Time/Path Refinement* phase, then in the *Path Selection* phase, we discover the optimal path minimizing the travel time. Then, based on the obtained optimal departure time sub-interval from *Time/Path Refinement* phase, as well as, the resulted optimal path in the *Path Selection* phase, we discover the optimal departure time instance in *Time Selection* phase. In the following, we discuss CTSR approach with more details.

In Time/Path Refinement phase, we iteratively generate and examine different partial routes, where the generated partial routes are placed in a min-heap Q. Each generated partial route p_i is modeled by a tuple as

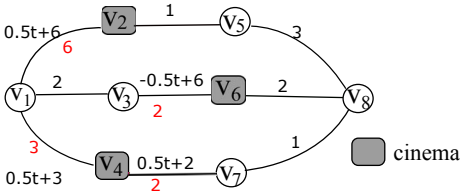


Figure 2: A sample road network.

$[v_i, T_{v_i}, AT_{v_i}(t), TT_{v_i}(t), TT^{lb}_{v_i}(t), R_{v_i}, T^{v_i}_{dep}(t)]$, illustrating a path starting from v_s ending at v_i, where the user departures the source location v_s in sub-interval $T_{v_i} \in T$. Furthermore, $AT_{v_i}(T_{v_i})$, $TT_{v_i}(T_{v_i})$ and $TT^{lb}_d(T_{v_i})$ indicate the arrival time interval, travel time interval and lower bound travel time interval from v_s towards v_i regarding to departure time sub-interval T_{v_i}. The generated partial routes are ordered in a min-heap Q based on the minimum lower bound travel time. To be more specific, the tuple $p_i = [v_i, T_{v_i}, AT_{v_i}(t), TT_{v_i}(t), TT^{lb}_{v_i}(t), R_{v_i}, T^{v_i}_{dep}(t)]$ is placed on the top of Q, if $\forall p_j = [v_j, T_{v_j}, AT_{v_j}(t), TT_{v_j}(t), TT^{lb}_{v_j}(t)] \in Q$, we have: $\min_{t \in T_{v_i}}\{TT^{lb}_{v_i}(t)\} \leq \min_{t' \in T_{v_j}}\{TT^{lb}_{v_j}(t')\}$.

In the Time/Path Refinement phase in CTSR approach, firstly the reverse graph of $\underline{G}$, denoted as $\underline{G}^R$, is loaded. Then, it computes the smallest distance of all vertices towards v_d by using $\underline{G}^R$, Line 2. Then in Line 4, according to Eq. 1, it computes the lower bound travel time from v_s to v_d through one POI from all COIs belonging to C. As the next step, through Lines 10-21, it generates a series of partial routes from v_s towards $\forall v_n \in adjacency(v_s)$, and pushes the corresponding tuples into Q. Let us assume, currently the corresponding tuples for partial paths p_i and p_l are the first and second top tuples in the Q, denoted by p_i=$[v_i, T_{v_i}, AT_{v_i}(t), TT_{v_i}(t), TT^{lb}_{v_i}(t), R_{v_i}, T^{v_i}_{dep}(t)]$ and p_l=$[v_l, T_{v_l}, AT_{v_l}(t), TT_{v_l}(t), TT^{lb}_{v_l}(t), R_{v_l}, T^{v_l}_{dep}(t)]$ (p_i and p_l incur the first and second smallest lower bound travel time among all existing generated partial routes). In CTSR approach, in the Time/Path Refinement phase in CTSR approach, in every iteration, it first dequeues the first top tuple from Q as p_i (Line 23). Subsequently, it uses the lower bound travel time of current top tuple in the top of Q, $TT^{lb}_{v_l}(t)$, as a basis for refining the departure time interval T_{v_i} (Lines 26-27). For that, we need to find a specific sub-interval $T_{v_n} \in T_{v_i}$, where $TT^{lb}_{v_i}(t) \leq \min_{\forall t' \in T_{v_l}}\{TT^{lb}_{v_l}(t')\}, \forall t \in T_{v_n}$. The reason is that, as aforementioned, we need to find the earliest and latest departure time from v_s, in which the partial route p_i outperforms p_l in terms of lower bound travel time. Then, as an iterative process, all neighboring vertices of v_i as $\forall v_n \in adjacency(v_i)$ are visited based on the refined departure time interval $T_{v_n} = [t'_1, t'_2]$ (Lines 29-37). Note that in CTSR approach as the algorithm proceeds we also need to update contents of Q such as tuple p_i assigned to vertex v_i, in the case that we find a more promising route to vertex v_i in terms of travel time and number of visited COIs (Lines 38-50). The Time/Path Refinement algorithm terminates, when

Table I: Time/Path Refinement in CTSR approach for example of Fig. 2

Step	v_i	T_{v_i}	$AT_{v_i}(t)$	$TT_{v_i}(t)$	$TT^{lb}_{v_i}(t)$	R_{v_i}	$T^{v_i}_{dep}(t)$	
1	v_4	$t \in [0,6]$	$[3,6]$ $y=1.32t+3$	$[3,6]$ $y=0.5t+3$	$[6,9]$ $y=0.5t+6$	v_4	$[4,13]$ $y=1.32t+4$	
	v_3	$t \in [0,6]$	[2,8]	2	6	0	$[2,8]$	
	v_2	$t \in [0,6]$	$[6,15]$ $y=1.32t+6$	$[6,9]$ $y=0.5t+6$	$[10,13]$ $y=0.5t+10$	v_2	$[7,16]$ $y=1.32t+7$	
2	v_3	$t \in [0,6]$	[2,8]	2	6	0	$[2,8]$	
	v_4	$t \in (0,6]$	$(3,6]$ $y=1.32t+3$	$(3,6]$ $y=0.5t+3$	$(6,9]$ $y=0.5t+6$	v_4	$(4,13]$ $y=1.32t+4$	
	v_7	0	8	7	8	v_4	8	
	v_2	$t \in [0,6]$	$[6,15]$ $y=1.32t+6$	$[6,9]$ $y=0.5t+6$	$[10,13]$ $y=0.5t+10$	v_2	$[7,16]$ $y=1.32t+7$	
3	v_6	$t \in [0,6]$	$[7,10]$ $y=-0.5t+5$	$[4,7]$ $y=-0.5t+7$	$[6,9]$ $y=-0.5t+9$	v_6	$[8,11]$ $y=-0.5t+6$	
	v_4	$t \in (0,6]$	$(3,6]$ $y=1.32t+3$	$(3,6]$ $y=0.5t+3$	$(6,9]$ $y=0.5t+6$	v_4	$(4,13]$ $y=1.32t+4$	
	v_7	0	8	7	8	v_4	8	
	v_2	$t \in [0,6]$	$[6,15]$ $y=1.32t+6$	$[6,9]$ $y=0.5t+6$	$[10,13]$ $y=0.5t+10$	v_2	$[7,16]$ $y=1.32t+7$	
4	v_8	6	13	6	6	v_6		✓
	v_4	$t \in (0,6]$	$(3,6]$ $y=1.32t+3$	$(3,6]$ $y=0.5t+3$	$(6,9]$ $y=0.5t+6$	v_4	$(4,13]$ $y=1.32t+4$	
	v_7	0	8	7	8	v_4	8	
	v_2	$t \in [0,6]$	$[6,15]$ $y=1.32t+6$	$[6,9]$ $y=0.5t+6$	$[10,13]$ $y=0.5t+10$	v_2	$[7,16]$ $y=1.32t+7$	
5	Return $\{ancestor(.,.)\}$ and $T^*=6$ as the output.							

Table II: Pre-computations for example of Fig. 2

v_i	$L(v_i, cinema)$	$L(v_i, v_8)$	$h_{v_i}(cinema, v_8)$
v_1	3	6	6
v_2	0	4	4
v_3	2	4	4
v_4	0	3	3
v_5	1	3	3
v_6	0	2	2
v_7	2	1	2
v_8	2	0	2

the current tuple in the top of Q meets the destination v_d and all required COIs.

The output of Time/Path Refinement phase in CTSR approach is an optimal sub-interval $T^* \subset [t_s, t_d]$, containing the optimal departure time instance, and a set of $ancestor$ functions. Then, in the second phase of CTSR approach, as Path-Selection, we start traversing the $ancestor$ functions from destination v_d towards source location based on T^*. In the Path-Selection phase in CTSR approach, we retrieve the optimal route by traversing the vertices starting from v_d towards source location v_s based on $ancestor$ functions and optimal sub-interval $T^* \subset [t_s, t_d]$, resulted from first phase. After discovering the optimal departure time interval $T^* \subset [t_s, t_d]$ in the first phase of CTSR approach, and retrieving the fastest possible path p^* in the second phase, subsequently in the third phase, we aim at discovering the optimal departure time instance from v_s. Algorithm 4 shows the pseudo-code for time-refinement step in CTSR approach, in which we find the optimal departure time sample $t^* \in T^*$, which minimizes the travel time from v_s to v_d based on path p^*.

Running Example for CTSR Approach: Let us assume Figure 2, indicating a sample road network, where some of its edges are time dependent. For example the travel time from from vertex v_3 to v_6, denoted by $-0.5t+6$, depends on the departure time instance from v_3 as t. Furthermore, in this figure the lower bound travel time of time-dependent edges have been illustrated by red color. Let us assume for a given GTDSR query, as $GTDSR(v_1, v_8, cinema, [0,6])$, the user is looking for a route for visiting a cinema while traveling from v_1 to v_8, where he is willing to leave the source location v_1 in time interval $[0,6]$.

Table I shows step by step execution of Time/Path Refinement in CTSR approach for example of Figure 2. In the first step, we visit all neighboring vertices of source location v_1 based on the given departure time interval $[0,6]$. For instance, assuming that we leave the source location v_1 in $[0,6]$, then the arrival time to v_4 would be as $AT_{v_4}(t)$=t+0.5t+3=1.32t+3, where the travel time from v_1 to v_4 equals to 0.5t+3, and the lower bound travel time from v_1 to v_8 via vertex v_4 equals to $TT^{lb}_{v_4}(t) = TT_{v_4}(t) + h_{v_4}(cinema, v_8) = (0.5t+3)+3 = 0.5t+6$. Assuming that the spent time in each *point of interest* is 1 unit, then the departure time from v_4 equals to $T^{v_4}_{dep}(t)$=1.32t+3+1=1.32t+4. Then in Step 2, we pop the tuple in the top of Q as [v_4,[0,6],$AT_{v_4}(t)$,$TT_{v_4}(t)$,$TT^{lb}_{v_4}(t)$,R_{v_4},$T^{v_4}_{dep}(t)$], which equals to route (v_1,v_4). Subsequently, we refine the corresponding departure time interval T_{v_4}=$[0,6]$ based on the lower bound travel time of second tuple in the top of Q as $[v_3, [0,6], AT_{v_3}(t), TT_{v_3}(t), TT^{lb}_{v_3}(t), R_{v_3}, T^{v_3}_{dep}(t)]$ which equals to 6, where we will have: $0.5t+6 \leq 6$, leads to t=0.

Subsequently, all neighboring vertices of v_4 will be visited

based on departure time instance t=0. The result of this step is generation of new partial route as (v_1, v_4, v_7) with total travel time 7. In the third step, the corresponding tuple for partial route (v_1, v_3) will be popped from Q, where there would be no refinement on corresponding departure time interval T_{v_3}=[0,6], considering that the lower bound travel time of the second tuple on the top of Q in Step 2, as (6,9], is always greater than 6. Consequently, the vertex v_6 will be visited for departure time interval $[0, 6]$, where the arrival time to v_6 will be computed as $AT_{v_6}(t) = -0.5(t+2)+6 = -0.5t+5$ with travel time as $TT_{v_6}(t) = -0.5(t+2)+6+2 = -0.5t+7$. In step 4, the corresponding tuple on the top of Q will be popped. Then the corresponding departure time interval for this tuple as $[0, 6]$ will be refined into t=6, based on the lower bound travel time of the second tuple on the top of Q, as $(6, 9]$. In step 5, the algorithm terminates, since the popped tuple from Q visits the destination v_8 and a POI as v_3, where, the set of $ancestor$ functions and the optimal sub-interval $T^* = 6$ will be returned as the output of Time/Path Refinement phase.

In the second phase on CTSR approach, by using the returned set of $ancestor$ functions in Time/Path Refinement phase and based on Algorithm 3, we will return the optimal path (v_1, v_3, v_6, v_8). In the third phase of CTSR approach, based on obtained optimal sub-interval $T^* = 6$ in Time/Path Refinement step and obtained optimal path p^*, then according to Algorithm 4, the optimal departure time instance $t^* = 6$ will be returned, as a time instance incurring the smallest travel time as 6 unit.

V. EXPERIMENTS

We compare the performance of our proposed approach with a base-line approach as Discrete-Time Sequenced Route (DTSR) approach. The DTSR approach can be considered as a straightforward approach for processing GTDSR queries. The basic idea behind of this approach is that, it initially extracts some departure time instances from the given departure time interval, and then aims at discovering the fastest trip from source location towards the given destination by traversing the road network vertices based on the initially extracted departure time instances. In DTSR approach, we use parameter δ as step size, introducing the difference between two consecutive departing time instances. The accuracy of DTSR approach depends on the number of extracted departure time instances. In our experiments we compare CTSR approach with DTSR techqniue based on different settings for parameter δ.

We conduct a comprehensive set of experiments on two road networks [1]: 1) San Joaquin road netwo,rk containing 18,263 vertices and 23,874 edges and 2) California road network which involves 21,047 vertices and 21,692 edges. In our experiments, we assume the aforementioned road networks as static road networks, where the weight of edges is constant. In our experiments, both of algorithms DTSR and CTSR were implemented in Java and tested on a windows platform with Intel Core4 CPU (2.4GHz) and 6GB memory. Furthermore, we uniformly distributed the POIs over the road network, where all COIs have the same number of POIs.

[1] www.cs.fsu.edu/lifeifei/SpatialDataset.htm

Table III: Experimental parameters and their values (bold defines default values).

Parameter	Range/Default
Density of POIs	50, 200, **400**, 800
Number of COIs	1, 2, **3** ,4
Query Locality	1%, 2%, **4%**, 8%

We evaluated the performance of DTSR and CTSR approaches with respect to different parameters shown in Table III: 1) the density of POIs, 2) the number of COIs, 3) the distance between the query source and target locations, which we refer to as "Query Locality". Note that in each experiment we run the DTSR approach based on different values for parameter δ indicating the step size as 1, 2, 4 and 8. In our experiments, we set the departure time interval as [0 24]. Furthermore, for each set of experiments, we executed 100 randomly generated queries and reported the average number of traversed road network vertices and processing time in Millisecond.

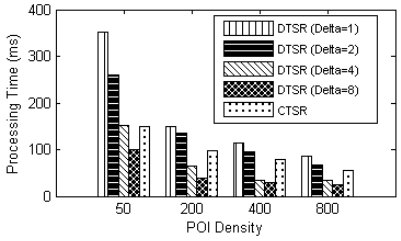
A. Effect of density of POIs

In this set of experiments, we varied the density of POIs in each COI from 50 to 800 based on Table III in San Joaquin and California road networks. Our experimental results, illustrated in Figures 3 and 4, indicate that the processing time and number of expanded verices decrease in both DTSR and CTSR approaches, when the density of POIs increases. The reason is that with the increase of POIs density, the candidate routes meet earlier the required condition for visiting at least one POI from different COIs. In other words, the length of optimal fastest possible route becomes shorter, when the cardinality of COIs increases.

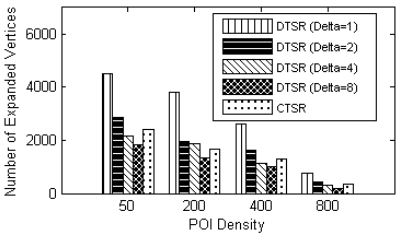
Based on our experimental results, the CTSR approach incurs smaller query processing cost in terms of response time and number of retrieved POIs in comparison to DTSR approach, when the step size in DTSR approach is set as $\delta = 1$ or $\delta = 2$. The reason is that, while generating the new candidate routes, CTSR approach also computes for which departure time interval from source location the new generated path can be considered as the most promising route in comparison to other candidate routes. So, we can say CTSR approach incurs smaller query processing cost due to traversing the road network vertices based on departure time intervals, where it avoids repeatedly examining the previously traversed vertices. On the other hand, our experimental results show that the DTSR approach with the step size 4 or 8 requires smaller query processing cost in comparison to CTSR approach due to examining a limited number of departure time instances for providing an approximate solution.

B. Effect of number of COIs

In this set of experiments, we varied the number of COIs from 1 to 4. Figures 5 and 6 illustrate the effect of the number of categories on query processing cost in San Joaquin and California road networks. Our experimental results show that

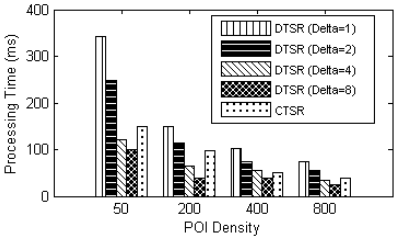


(a)

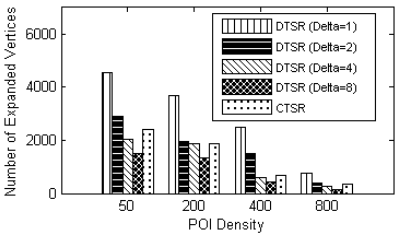


(b)

Figure 3: The effect of POIs density on processing time and number of expanded vertices in San Joaquin road network



(a)



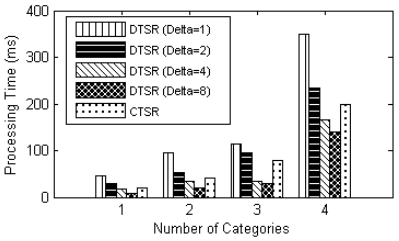
(b)

Figure 4: The effect of POIs density on processing time and number of expanded vertices in California road network

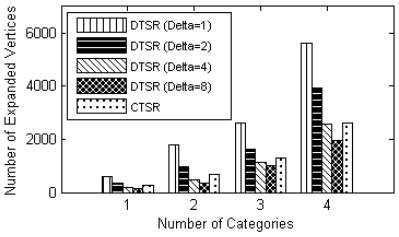
the query processing overhead of both approaches increases with the increase of number of categories that must be visited. The main reason is that with the increase of number of COIs the search space becomes larger, since the travel time of the optimal route increases for visiting at least one POI from required COIs. The results of this set of experiments illustrate this point that although DTSR approach resulted in approximation solution with step size 4 or 8 requires smaller query processing cost in comparison to CTSR approach, but CTSR approach requires smaller query processing cost in comparison to DTSR approach with step size 1 or 2, due to traversing the road network vertices based on departure time sub-intervals, where it avoids repeatedly examining the previously traversed vertices.

C. Effect of query locality

Figures 7 and 8 illustrate the effect of query locality , i.e., the distance between the source location and destination, on query processing cost of different approaches in San Joaquin and California road networks, respectively. As our experimental results show, the query processing cost of both DTSR and CTSR approaches increases with the increase of the distance between the source and target location. The larger the distance between the origin and target location, the greater number of expanded vertices and consequently the higher processing time. The main reason is that when the distance between the source and target location increases, the search space becomes larger due to an increase in the length of fastest possible route, and consequently the corresponding travel time. Thus higher number of candidate routes are supposed to be generated and examined. Our experimental results show that the CTSR approach incurs smaller query processing cost in



(a)



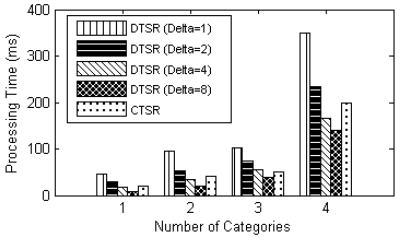
(b)

Figure 5: The effect of number of categories on processing time and number of expanded vertices in San Joaquin road network

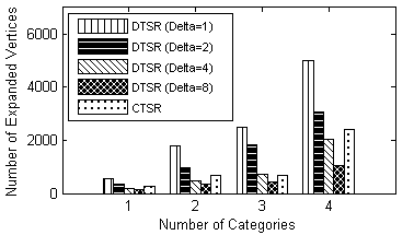
comparison to DTSR approach with step size 1 or 2, due to traversing the road network vertices based on departure time sub-intervals in comparison to single departure time instances.

VI. Conclusion

In this work, we addressed the extension of Optimal Sequenced Route (OSR) query in two directions. First, we considered the underlying network to be time-dependent, and

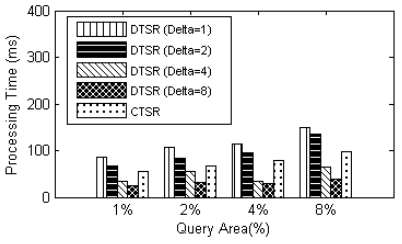


(a)

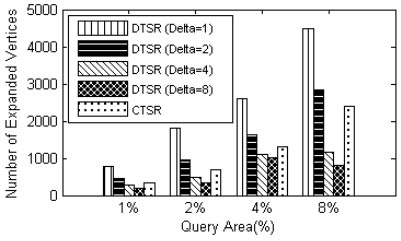


(b)

Figure 6: The effect of number of categories on processing time and number of expanded vertices in California road network



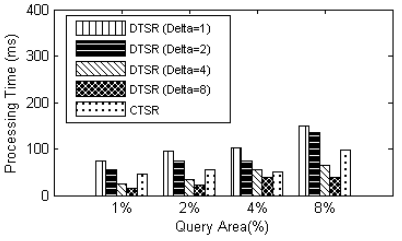
(a)



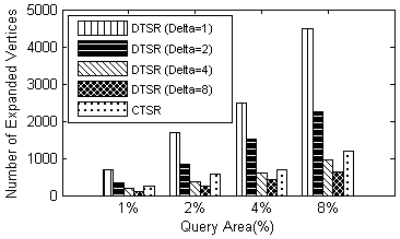
(b)

Figure 7: The effect of query locality on processing time and number of expanded vertices in San Joaquin road network

secondly the user inputs a departure time interval from source location. This extension, calling it General Time-Dependent Sequenced Route (GTDSR) query is a more realistic assumption with respect to, for instance, urban road network, where the users may look for the best departure time from a departure time interval in order to avoid being stuck in traffic. For solving GTDSR queries, we proposed CTSR technique based on the A^* paradigm with an admissible heuristic function



(a)



(b)

Figure 8: The effect of query locality on processing time and number of expanded vertices in California road network

for efficient traversing of the search space. We evaluated the efficiency of our proposed approaches via an extensive set of experiments.

REFERENCES

[1] Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.

[2] Camila F Costa, Mario A Nascimento, José AF Macêdo, Yannis Theodoridis, Nikos Pelekis, and Javam Machado. Optimal time-dependent sequenced route queries in road networks. In *SIGSPATIAL 2015*, page 56.

[3] Bolin Ding, Jeffrey Xu Yu, and Lu Qin. Finding time-dependent shortest paths over large graphs. In *EDBT 2008*, pages 205–216.

[4] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *Mathematical Programming*, 14(2):47–57, 1984.

[5] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. Finding fastest paths on a road network with speed patterns. In *ICDE 2006*, page 10.

[6] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In *SSTD 2005*, pages 273–290.

[7] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB journal*, 17(4):765–787, 2008.

[8] Mehdi Sharifzadeh and Cyrus Shahabi. Processing optimal sequenced route queries using voronoi diagrams. *GeoInformatica*, 12(4):411–433, 2008.

[9] Kiseok Sung, Michael GH Bell, Myeongki Seong, and Soondal Park. Shortest paths in a network with time-dependent flow speeds. *European Journal of Operational Research*, 121(1):32–39, 2000.