

# Taxi, Please ! A Nearest Neighbor Query in Time-Dependent Road Networks

Mirla Chucre<sup>1</sup> Samara Nascimento<sup>1</sup> Jose Antonio Macedo<sup>1</sup> Jose Maria Monteiro<sup>1</sup> Marco Antonio Casanova<sup>2</sup>

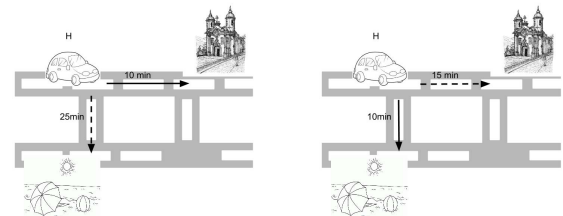
**Abstract**—In this paper we propose a new kind of  $k$ NN query on time-dependent network, which aims at finding  $k$  points of interest that are closest in time to a query point. This query is useful for many kind of applications where a user/customer should ask for a service provided by many moving providers (e.g. taxi drivers, ambulances, food delivers, etc). We described our solution and present experimental results comparing our proposed algorithm to a baseline approach. The experimental results show that our approach is efficient and effective.

## I. INTRODUCTION

The modern society faces many challenges in solving problems related with people mobility in large urban centers. Mobility is becoming an imperative issue because the road networks of big cities do not grow at the same rate as the number of vehicles. The high amount of vehicles creates congestion in the roads, which makes travel time forecasting extremely hard. Besides, travel time may radically change from rushing hours to normal hours. So, the assessment and consideration of traffic conditions is key for leveraging intelligent transportation systems. Intelligent systems based on models built through real data evaluations are critical to help the contemporary society in solving the mobility problem in big cities.

With the increasing interest in intelligent transportation, more complex and advanced query types were recently proposed, such as nearest neighbors ( $k$ NN) queries [1], [2] and route planning queries [3], [4]. Some recent studies included the temporal dependence to solve conventional spatial queries, such as  $k$ NN [5], [6] and shortest path [7] queries. In time-dependent networks, a  $k$ NN query, called TD- $k$ NN, returns the  $k$  points of interest with minimum travel-time from the query point. As a more concrete example, consider the following scenario. Imagine a tourist in Fortaleza who is interested in visiting the touristic attraction closest from him/her. Let us consider two points of interest in the city, the Fortaleza Cathedral and the Iracema Beach. He/she asks a query asking for the touristic attraction whose path

leading up to it is the fastest at that time; the answer depends on the departure time. For example, at 10:00h it takes 10 minutes to go to the Cathedral (see Figure 1a). It is the nearest attraction. Although, if he/she asks the same query at 22:00h, in the same spatial point, the nearest attraction is the Iracema Beach (see Figure 1b).



(a) The nearest neighbor at 10h.

(b) The nearest neighbor at 22h.

Fig. 1: An example of  $k$ NN query.

In this work, we identify a new variation of nearest neighbors queries in time-dependent road networks that has wide applications and requires novel algorithms for processing. Differently from TD- $k$ NN queries, we aim at minimizing the travel time from points of interest to the query point. As an example of the application of such query, a cab company can find the nearest taxi in time to a passenger requesting transportation. More specifically, we address the following query: find the  $k$  points of interest (e.g. taxi drivers) which can move to the query point (e.g. a taxi user) in the minimum amount of time.

We propose and discuss a solution to this type of query, which is based on the previously proposed incremental network expansion and use the  $A^*$  search algorithm equipped with suitable heuristic functions. We also discuss the design and correctness of our algorithm and present experimental results that show the efficiency and effectiveness of our solution. We note that, in some cases, the answer returned by a regular TD- $k$ NN query is coincidentally the same as returned by our query, for example, when all the roads allow bi-directional vehicle flow and all conversions are allowed, what is unreasonable in real situations.

This paper is structured as follows. In Section 2, we introduce some important definitions for understanding our proposed solutions and explain the road network

\*This work was not supported by any organization

<sup>1</sup>Dep. de Computacao, Federal University of Ceara, Brazil [mirla.chucre@dc.ufc.br](mailto:mirla.chucre@dc.ufc.br) [samara.nascimento@dc.ufc.br](mailto:samara.nascimento@dc.ufc.br) [monteiro@dc.ufc.br](mailto:monteiro@dc.ufc.br) [jose.macedo@lia.ufc.br](mailto:jose.macedo@lia.ufc.br)

<sup>2</sup>Dep. de Informatica, Pontificia Universidade Catolica do Rio de Janeiro, Brazil [casanova@inf.puc-rio.br](mailto:casanova@inf.puc-rio.br)

model used. In Section 3, we present a brief discussion of related works. In Section 4, we explain our approach and show the correctness of our solution. We show an experimental evaluation and results in Section 5. Finally, we present the conclusions of the paper in Section 6.

## II. RELATED WORK

The problem of  $k$ NN queries in road networks was introduced in [1]. Moreover, [8] introduced the problem of  $k$ NN queries in time-dependent networks. Time-expanded graphs allows us to exploit previous solutions in static networks to solve TD- $k$ NN queries. An improved solution is proposed by [6]. In this work, the authors proposed an algorithm that is based on INE expansion [1] and use an  $A^*$  [9] search to guide this expansion. Furthermore, [10], consider the problem of finding the closest point of interest in road networks where the travel time along each edge is a function of the departure time.

In addition to  $k$ NN queries, there are other types of proximity queries known as Reverse Nearest Neighbor (RNN) and Reverse Farthest Neighbor (RFN) queries [11]. A Reverse Nearest Neighbor (RNN) query retrieves interest points which consider the query point as their nearest neighbor. In contrast, a Reverse Farthest Neighbor (RFN) query retrieves interest points which consider the query point as their farthest neighbor.

In [12], the authors discussed various types of RNN queries and proposes several algorithms to process these queries in spatial network databases. Similar to NN, the basic RNN query can be generalized to find objects that consider the query point as one of the  $k$  nearest neighbors. This kind of query is called  $Rk$ NN, where  $k$  can be any number given at the query time. Finally, [11] introduced other types of reverse proximity queries, called RFN (Reverse Farthest Neighbor) and  $Rk$ FN.

## III. TIME DEPENDENT GRAPH MODEL

We assume that the structure of a time-dependent road network is modeled by a graph where the vertices represent starting and ending points of road segments or intersections. Those are connected by edges and the cost to traverse these edges vary with time. More formally, we formalize TDG as follows:

**Definition 1: (Time-Dependent Graph (TDG))** A **time-dependent graph** (TDG)  $G = (V, E, C)$  is a graph where: (i)  $V = \{v_1, \dots, v_n\}$  is a set of vertices; (ii)  $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$  is a set of edges; (iii)  $C = \{c_{(v_i, v_j)}(\cdot) \mid (v_i, v_j) \in E\}$ , where  $c_{(v_i, v_j)} : [0, T] \rightarrow R^+$  is a function which attributes a positive weight for  $(v_i, v_j)$  depending on a time instant  $t \in [0, T]$  and where  $T$  is a domain-dependent time length.

We assume that  $H(\cdot)$  is a set of function that are defined in the interval  $[0, T]$  where  $T$  is a domain-dependent time length. Particularly, in this work, we assume that  $T$  has the granularity of 15 minutes during a day. For each edge  $(u, v)$ , a function  $c_{(u, v)}(t)$  gives the cost of traversing  $(u, v)$  at the departure time  $t \in [0, T]$ . We also assume that the

travel times of the edges in the network follow the FIFO property, i.e., an object that starts traversing an edge first has to finish traversing this edge first as well. The general time-dependent shortest path problem in which the departure is immediate, i.e. the user departs exactly at the time  $t$ , and in which waiting is disallowed everywhere along the path through the network is NP-hard [13], but it has a polynomial time solution in FIFO networks. Since the travel times satisfy the FIFO property, waiting in a intermediary vertex in a path is not beneficial.

Note that the definition given above does not require the graph to be bidirected. More specifically, the existence of an edge  $(u, v)$  does not imply in the existence of the edge  $(v, u)$ . Furthermore, there may be opposing edges  $(u, v)$  and  $(v, u)$  such that  $c_{(u, v)}(t) \neq c_{(v, u)}(t)$ . As an example, consider the graph shown in Figure 2 which is a representation of a time-dependent road network. The travel times of its edges for each instant of a day are shown in the graphics in Figure 2b. The pairs of opposite edges  $(A, C)$  and  $(C, A)$  and  $(B, C)$  and  $(C, B)$  have the same cost. However,  $(A, B)$  and  $(B, A)$ , although opposite, have distinct costs.

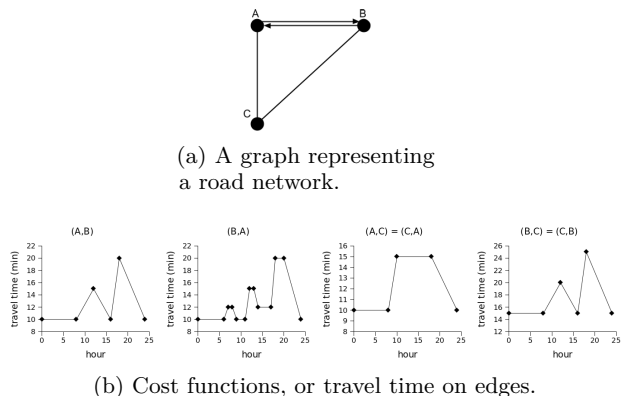


Fig. 2: A graph representing a road network and the costs of its edges for different times of a day.

The time cost to traverse a path from a specific starting time, or departure time, is called *travel-time*. The *travel-time* is calculated assuming that stops are not allowed because, as discussed before, we consider that the network is FIFO waiting in a vertex do not anticipate the arrival time of a vehicle. The *travel-time* of a path is calculated considering the *arrival time* at each vertex belonging to it. These concepts are formally defined below.

**Definition 2:** Given a TDG  $G = (V, E, C)$ , the *arrival time* at the vertex  $v_j$  of an edge  $(v_i, v_j) \in E$  at departure  $t \in [0, T]$  is given by  $AT(v_i, v_j, t) = t + c_{(v_i, v_j)}(t) \bmod T$ .

Given that a vehicle starts a path at a vertex of the graph and this path starts at a determined departure time, the *arrival-time* calculates the time instant when the vehicle arrives at the other end of the edge. Considering the cost functions shown in Figure 2b, when traversing the road represented by  $(B, A)$  at 10:00 am, a vehicle arrives at 10:10 am at  $A$ . Note that the operation of the

rest (mod) exists since the calculation of the arrival-time is circular. For instance, consider that one departs from  $B$  towards  $A$  at  $t = 24:00$ ,  $AT(v_i, v_j, 24:00)$  is 0:10, since the vehicle arrives at the other end of the edge at this time.

*Definition 3:* Given a TDG  $G = (V, E, C)$ , a path  $p = \langle v_{p_1}, \dots, v_{p_k} \rangle$  in  $G$  and a departure time  $t \in [0, T]$ , the *travel-time* of  $p$  is the time-dependent cost to traverse this path, given by  $TT(p, t) = \sum_{i=1}^{k-1} c_{(v_{p_i}, v_{p_{i+1}})}(t_i)$  where  $t_1 = t$  and  $t_{i+1} = AT(v_{p_i}, v_{p_{i+1}}, t_i)$ .

The above definition shows how the cost of a path, called *travel-time*, is calculated. Given the sequence of vertices that compose a path and the time instant when one starts to traverse this path, the *travel-time* is the sum of the costs to go from one vertex to the next one in the sequence. The cost to go from the first to the second vertex is calculated considering the departure time  $t$ . The cost to reach the next vertices depends on the *arrival time* at the previous vertex. It is important to notice that this definition does not take into consideration stops at the nodes of the graph, that is, the way to the next vertex in the sequence begins at the same moment when the previous vertex was reached.

*Definition 4: (Point of Interest (POI))* Let  $P = \{p_1, \dots, p_n\}$  be a set a point of interest (POI)  $p_i = (pid, loc, mf)$ ,  $1 \geq i \leq n$  where  $pid$  is the POI identification,  $loc$  is the location of  $p$  in geographical space (i.e. latitude, longitude) and  $mf : RxR \rightarrow V$  is a given mapping function that associates univocally the location of  $p$  to a vertex  $v \in V$ .

#### IV. REVERSE NEAREST NEIGHBOR QUERIES IN TIME-DEPENDENT ROAD NETWORKS

In this section, we describe the problem of finding nearest neighbor on a time-dependent network, hereinafter called NN-Reverse-TD query. The objective of this query is to find the nearest POI  $p$  to a query object  $qp$  in a time-dependent graph such that  $p$  has the minimum time-dependent travel-time to  $q$ .

The TDG is built from a road network model where each edge contains a non-negative weights, a linear function according to the time of day. Each edge connects two different nodes of the graph. We also assume that our TDG is populated with a set of point of interests. In our scenario, POIs may change their position along time, representing, for instance, taxi cabs, ambulances, food delivers, etc. However, during query initiation all POIs are mapped to network vertices and their positions do not change during query processing.

We present two solutions to process NN-Reverse-TD query, namely Naïve and our proposal. They are based on an algorithm that performs as incremental network expansion (INE) [1]. The first solution use a Dijkstra algorithm [14], and the second an A\* search [9] to guide this expansion, i.e., to determine the order in which vertices are expanded in TDG. It uses the current distance plus a heuristic function  $H(\cdot)$ .

The Algorithm 1 (NN-Reverse-TD-Dijkstra) presents the general structure of the algorithm to solve the Naïve Solution (i.e. the Baseline). Furthermore, the Algorithm 2 (NN-Reverse-TD) formalizes our solution to the RNN-TD problem.

##### A. Problem Statement

Now we are ready for stating the problem of finding the Nearest Neighbor Reverse in Time Dependent Network Query (NN-Reverse-TD Query). In order to formalize this problem, we resort to the Definitions 1, 2, 3 and 4 presented in previous section .

Let  $P = \{p_1, \dots, p_N\}$  be the set of POIs mapped on vertices of a TDG graph  $TDG = (V, E, C)$ . Each POI  $p$  is univocally mapped to a vertice of  $TDG$ . Let  $tt(s, t)$  be a travel function, which computes the travel time from source node  $s$  to target node  $t$  over  $TDG$ , where  $s, t \in V$ .

*Definition 5 (Query Point):* A query point  $qp \in V$  is a vertex of  $TDG$ . This query point  $qp$  is arbitrarily selected by a user.

We formalize the NN-Reverse-TD query problem as follows:

*Definition 6 (Reverse-TD Query Problem):* An instance of *Reverse Query Problem* is a tuple  $NNRQ = (TDG, P, qp, qt, wt)$ , where  $TDG$  is a TDG graph,  $P$  is a set of point of interests,  $qp$  is a query point such that  $qp \in V$ ,  $qt \in [0, T]$  is a non-negative integer representing the query time and  $mtt \in [0, T]$  is a non-negative integer defining the maximum waiting time. Given an instance of NRQP, the problem is to find a  $p \in P$  that satisfies the following conditions: (i)  $tt(p, qp) \leq wt$ ; (ii)  $\forall pi \in V \mid tt(p, qp) \leq tt(pi, qp)$  and  $p \neq pi$

Condition 1 establishes that the travel time should be less than the waiting time  $wt$  and condition 2 specifies that travel time from  $p$  to  $qp$  must be minimal taking into account all POIs.

##### B. Naïve Solution (Baseline)

The naïve approach is basically an extension of Dijkstra's algorithm [14], which addresses the problem of finding the shortest path from an origin to a destination node. Dijkstra's algorithm computes all the shortest path between the source node to every target nodes. However Dijkstra's algorithm does not take into consideration the travel time of the each edge, to this end we rely on A\* shortest path implementation described in our previous work [6], which adopts the following criteria: (i) The target is a point of interest; (ii) The non-negative weights are an heuristic function  $H(\cdot)$  that adds to each edge an estimate of the cost to reach any another edge.

The idea is to reach POIs quickly. Then, after they are reached, we calculate the travel time to *query object*, 5. This is clearly a sub-standard solution, but which serves nevertheless as a baseline. It is important to stress that our naïve solution, as well as Dijkstra's algorithm determine if any two 'shortest path' only one will be returned.

1) *Off-line Pre-processing*: The naïve solution has two pre-processing steps that are executed off-line, i.e., before processing the query. The first step calculates the value of the heuristic function to the vertices of  $G$ . For each vertex  $v \in V$ , the distance between it and the nearest point of interest in  $G$  is calculated. This distance is used as an estimate to the time to reach a POI from  $v$ .

---

**Algorithm 1: NN-Reverse-TD-Dijkstra (Baseline)**

---

```

input      : A query point  $q \in V$ , maximum travel time  $t \in [0, T]$ 
              and day time  $dt$ 
output    : The nearest neighbor to  $q$  considering the maximum
              travel time  $t$ 

1 begin
2    $poiIds \leftarrow V_{POI}[1..k]$ ;
3    $TT_{min} \leftarrow t$ ;
4    $POI \leftarrow -1$ ;
5    $VV \leftarrow \emptyset$ ;
6   while  $poiIds$  do
7      $Path \leftarrow Dijkstra(q, poiIds_u, dt)$ ,  $u$  is POI;
8     if  $Cost(Path) \leq t$  then
9       if  $Cost(Path) \leq TT_{min}$  then
10         $POI \leftarrow poiIds_u$ ;
11         $TT_{min} \leftarrow Cost(Path)$ ;
12         $VV \leftarrow Path_{verticesVisited}$ ;
13      end if
14    end if
15  end while
16  if  $POI > -1$  then
17     $NN(POI, TT_{min}, VV)$ ;
18  end if
19 end

```

---

2) *Query Processing*: Algorithm 1 (NN-Reverse-TD-Dijkstra) presents the algorithm to solve the NN-TD Query problem. Using polymorphism to put the problem in a more abstract structure, we have the input method a query point  $q \in V$ , maximum travel time  $t \in [0, T]$  and time of day  $dt$ .

As the two solutions differs in implementation, the baseline algorithm is similar to Backtracking Search. First, it returns all  $v \in V$  that represent a point of interest in graph (line 2). Then, for each point of interest returned, a shortest path check is performed from the point of interest to the query point (line 6 and 7). As previously discussed, this shortest path is an adaptation of Dijkstra’s algorithm, the weight of each edge is replaced by a heuristic function  $H(\cdot)$  whose input is a time of the day and whose output is the time required.

When there is a shortest path from point of interest to query point the next step is to carry out the following checks: (i) if the total time that was returned in the baseline query is less than or equal to the input value  $t$ , this is *maximum travel time* for moving from *point of interest to query point* (line 8); (ii) if the total time that was returned in the baseline query is less than or equal to a value already attributed (line 3) with the lowest value for moving from *point of interest to query point* (line 9).

### C. NN-Reverse-TD Algorithm (Proposed Solution)

In this section we present the proposed algorithm and show how to perform the pre-processing step. This algorithm is based on the Incremental Network Expansion (INE) algorithm, initially proposed in [1]. The INE is an

algorithm based on Dijkstra’s algorithm visited all the reachable vertices of  $q$  in order of their proximity, until nearest point of interest are located. However different from previous solutions, our approach does the expansion using a reverse graph,  $G^R$ , which allows finding and pruning candidates POIs.

The heuristic function of the naïve solution does not take into consideration reverse graph. Its goal is to reach closer POIs within a shorter time. Then, after these POIs are reached, the better POI is calculated. This solution is not very efficient in the sense that it spends time searching for POIs that are close to  $q$ , but that may take a long time to return a solution. Clearly, the use of reverse graph heuristic is the best option, because besides considering an estimate to reach POIs, can better guide the search of them.

Based on this, we propose a reverse nearest neighbor solution. This heuristic allows the search to be expanded capture the heuristic function  $H(\cdot)$  is not from the origin vertex (i.e.  $q$ ) to destination vertex (i.e. POI). The crossing follows the idea of having a graph going backwards. So, the search to be expanded is from the destination vertex to  $q$ .

Now we formalize the notion of reverse time dependent graph as follows:

*Definition 7 (Time-Dependent Graph Reverse)*: The *reverse graph*, defined by  $G^R$ , is a graph of  $G$  with the same set of vertices, but the edges are reversed, i.e., if  $G$  contains an edge  $(u, v)$  then the reverse of  $G$  contains an edge  $(v, u)$ .

1) *Offline Pre-processing*: The pre-processing has two steps. In the first step we created the graph  $G$  in memory with the values of the heuristic function. Differently from the naïve solution, we carry out the process to reverse graph,  $G^R$ , where all directed edges are reversed.

2) *Query Processing*: The two solutions differ in implementation, but they have the same input value: a query point  $q \in V$ , the maximum travel time  $t \in [0, T]$  and a query time  $qt$ . However, the builder objects for handling the query method, the parameter graph is an reverse graph. Algorithm 2 formalizes our solution to the RNN-TD problem.

The algorithm begins the expansion and inserts  $q$  in a priority queue  $Q$  (line 5) that stores the set of candidates for expansion in the next step. An entry in queue  $Q$  is a tuple  $(v_i, ATv_i, TTv_i)$ , where  $ATv_i = AT(q, v_i, dt)$  and  $TTv_i = TT(q, v_i, dt)$ . The priority of elements in  $Q$  is given by the increasing order of  $TTv_i$  values with the purpose of checking first the vertices that offer a greater chance to reach the  $POI$  in  $V$ .

Next, the vertices are dequeued from  $Q$  (line 6). When a vertex  $u$  is dequeued from  $Q$  it is marked as visited (line 7) in the *Visited* list corresponding to the identifier vertex visited until the current vertex. For example, the *Visited*[1] list represents that the vertice with identifier (id) equals 1 already been visited. The path from this vertex to  $q$  is defined in another list, *Parents* list.

---

**Algorithm 2:** NN-Reverse-TD

---

```
input : A query point  $q \in V$ , maximum travel time  $t \in [0, T]$ 
        and day time  $dt$ 
output : The nearest neighbor with the minimum route from
         $POI$  to  $q$  considering the maximum travel time  $t$ 

1 begin
2    $G^R \leftarrow G_{reverseGraph}$ ;
3    $AT_q \leftarrow t + dt$ ;
4    $TT_{max} \leftarrow 0$ ;
5   En-queue  $(q, AT_q, TT_{max})$  in  $Q$ ;
6   while  $Q \neq \emptyset$  do
7      $(q, AT_q, TT_{max}) \leftarrow$  De-queue  $Q$ ;
8     Mark  $q$  as visited;
9     if  $TT_{max} \leq t$  then
10      if  $q$  is  $POI$  then
11         $ReturnNN(POI, TT_{min}, VV)$ 
12      end if
13      for  $u \in adjacency(q)$  do
14        if  $u$  is visited then
15          continue;
16        end if
17         $TT_{current} \leftarrow q_{travelTime} + u_{travelTime}$ ;
18        if  $TT_{current} > t$  then
19          continue;
20        end if
21         $AT_{current} \leftarrow q_{ArrivalTime} - u_{travelTime}$ 
22        En-queue  $(u, AT_{current}, TT_{current})$  in  $Q$ ;
23      end for
24    end if
25  end while
end
```

---

For every element  $u$  pull of the  $Q$  check if the vertex was added to  $Visited$  list (line 13), i.e., if  $Visited$  contain vertex of the  $u$ . Another checking is realized, if maximum travel time  $t$  was reached, if this return *true*, an exception thrown path not found for maximum travel time and time of day, line 17 and 18.

For every  $v$  neighbor of  $u$  for an updated  $ATv$ , we check if it is in the  $Visited$  list. If this condition is satisfied (which is verified on line 16) "jumps over" one iteration in the loop (line tal), doing this we avoid re-expand vertices unnecessarily. Otherwise,  $v$  add to  $Visited$  (line tal).  $TT$  is updated, added travel time value to reach the current neighbor, line tal. Again to verify that the maximum service time has been exceeded, it has been exceeded, "jumps over" one iteration. The  $Parents$  list is updated with  $v$ . Already, as the network is expanded,  $ATv_i$  is decremented with  $t$  travel time value of the current neighbor.

The priority queue  $Q$  offer the entry with a tuple  $(v, ATv, TTv)$ . The algorithm stops when the vertex is  $POI$  and maximum travel time for service,  $t$ , it is not busted, as shown on lines 9 and 10.

## V. EXPERIMENTAL EVALUATION

This section describes the experimental evaluation conducted in order to evaluate the proposed algorithm and compare it to a naive solution used as a baseline. All experiments were conducted on a Intel Core 2 Quad CPU Q6600 server, with 8GB RAM and 2.40GHz, using Ubuntu 11.10 of 64 bits as operating system. Our performed the experiments without caching.

We defined a synthetic cost function, denoted by  $H(\cdot)$ , where for each edge, we chose a random speed between

3km/h and 60km/h for each interval of the day, so that the time cost given by the ratio between the edge length and this speed satisfies the FIFO property. The cost function  $H(\cdot)$ , for a given edge  $e$ , has a temporal resolution of 96 points in time, e. g., a value at every 15 minutes of a day. So, for a certain edge  $e$ , the cost in time to traverse  $e$  may vary every 15 minutes.

In order to generate time-dependent road networks, we have used two different data sets. The first data set, available from *Open Street Map* (OSM) [15]. The second data set, available from a Brazilian cab fleet monitoring company called Taxi Simples [16], provides data about the distribution and moving of cabs in Fortaleza city. Since, each taxi driver at Taxi Simples company has a mobile phone equipped with a GPS device, this data set stores information about the location of taxi drivers. So, this data set stores many geographic positions for the same cab. Once the size of this data set is too large, we have used only the data collected on November 16, 2015, from 11am to 12pm (local time).

We created a database, using *PostgreSQL 9.3.10* with the spatial database extension *Postgis, 2.1*, containing two tables: *ROADS* and *TAXI-CAB*. The table *ROADS* stores information about the streets of the Fortaleza city. This table has two columns: *gid*, the street identifier, and *geom*, whose data type is *MultiLineString* and stores a point, with latitude and longitude, obtained for each corner, which corresponds to a vertex. The *TAXI-CAB* table stores information about the taxi positions. This table has three columns: *id*, the cab identifier, *point\_geo*, whose data type is *Point* and stores a point, with latitude and longitude, representing the cab position, and *gid*, a foreign key used to match the cab position and the road network.

The taxi positions, stored in *TAXI-CAB* table follows the *GEOLOCATION API* specification [17] and have an average error of 13.60 meters. In this case, we performed the map matching between the taxi positions and the road network, that is necessary to place the cab over a correct street segment.

The time dependent network was analyzed from experimental tests performed on different partitioning. We changed the volume of vertices in 1k, 10k, 50k and 100k. After this, we generate, for each data set, 1% of points of interest. We executed 100 randomly selected queries with  $k = 1$  on each network. In order to select a snippet of the Fortaleza road network, we have used the following strategy: we choose the square region with the largest number of cabs. The query point was selected randomly and the chosen snippets of the Fortaleza road network are within 1k, 10k, 50k and 100k vertices.

The *Open Street Map* data set about the Fortaleza city has a road network disconnection, with certain kinds of "islands". This happens because Fortaleza is usually composed of islands, which do not allow the existence of street segments, similar to the Cocó Park, Airport, Universities, among others. Thus, since the query points

were chosen randomly, in some executions there was no way between the query point (passenger) and a POI (taxi driver). In this cases, the query point belonged to a disconnected sub-graph. As the Fortaleza road network provided by *Open Street Map* is disconnected, the average number of incident edges is very low (close to 1), as illustrated in Figure 3. Figure 4 shows the ratio of executions that found a POI. For example, in the experiment using road networks with size of 1k just 40% of the executions found a POI.

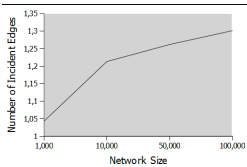


Fig. 3: Number of incident edges

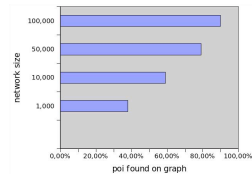


Fig. 4: Ration of executions that found a POI X Network size.

Figure 5, on logarithmic scale, illustrates the average behavior of both algorithms when the network size increases. This experiment indicates that the size of network does not affect significantly the average number of expanded vertices. Despite of the increases in the processing time and the number of expanded vertices, the pre-processing time of the Naïve solution increases faster, therefore, the proposed algorithm outperforms the baseline solution.

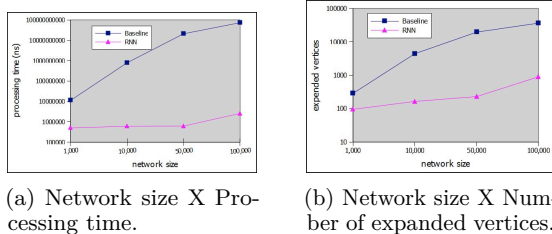


Fig. 5: Evaluation of network size influence in real road networks.

## VI. CONCLUSIONS AND FUTURE WORK

In this work, we proposed a new query that returns the the nearest taxi in time to a passenger requesting transportation. With this approach, we can find the k points of interest with low travel times to the query point. This solution find the nearest neighbor from a certain query point and a given departure time in TDGs where the service time of the facilities are taken into consideration. We discussed our proposed solution and presented experimental results comparing the quality of the response returned. The experimental results show the efficiency and effectiveness of our solution. We are currently investigating variations of this query, for example,

we want to consider a solution to dynamic networks, with dynamic weights, where we have frequent updates.

## REFERENCES

- [1] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, “Query processing in spatial network databases,” in *VLDB*, 2003, pp. 802–813. [Online]. Available: <http://www.vldb.org/conf/2003/papers/S24P02.pdf>
- [2] M. Kolahdouzan and C. Shahabi, “Alternative solutions for continuous k nearest neighbor queries in spatial network databases,” *GeoInformatica*, vol. 9, no. 4, pp. 321–341, 2005.
- [3] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, “The optimal sequenced route query,” *VLDB J.*, pp. 765–787, 2008.
- [4] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann, “The partial sequenced route query with traveling rules in road networks,” *GeoInformatica*, pp. 541–569, 2011.
- [5] U. Demiryurek, F. B. Kashani, and C. Shahabi, “Efficient k-nearest neighbor search in time-dependent spatial networks,” in *Database and Expert Systems Applications, 21st International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part I*, 2010, pp. 432–449. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-15364-8\\_36](http://dx.doi.org/10.1007/978-3-642-15364-8_36)
- [6] L. A. Cruz, M. A. Nascimento, and J. A. F. de Macêdo, “k-nearest neighbors queries in time-dependent road networks,” *JIDM*, vol. 3, no. 3, pp. 211–226, 2012. [Online]. Available: <http://seer.lcc.ufmg.br/index.php/jidm/article/view/187>
- [7] G. Nannicini, D. Delling, D. Schultes, and L. Liberti, “Bidirectional A\* search on time-dependent road networks,” *Networks*, vol. 59, no. 2, pp. 240–251, 2012. [Online]. Available: <http://dx.doi.org/10.1002/net.20438>
- [8] U. Demiryurek, F. Kashani, and C. Shahabi, “Towards k-nearest neighbor search in time-dependent spatial network databases,” in *Proc. of the 6th DNIS Workshop*, 2010, pp. 296–310.
- [9] A. V. Goldberg and C. Harrelson, “Computing the shortest path: A search meets graph theory,” in *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, 2005, pp. 156–165. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1070432.1070455>
- [10] C. F. Costa, M. A. Nascimento, J. A. F. de Macêdo, and J. C. Machado, “Nearest neighbor queries with service time constraints in time-dependent road networks,” in *Proceedings of the Second ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS 2013, November 5, 2013, Orlando, Florida, USA, 2013*, pp. 22–29. [Online]. Available: <http://doi.acm.org/10.1145/2534190.2534194>
- [11] Q. T. Tran, D. Taniar, and M. Safar, “Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks,” *T. Large-Scale Data- and Knowledge-Centered Systems*, vol. 1, pp. 353–372, 2009. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-03722-1\\_14](http://dx.doi.org/10.1007/978-3-642-03722-1_14)
- [12] M. Safar, D. Ebrahimi, and D. Taniar, “Voronoi-based reverse nearest neighbor query processing on spatial networks,” *Multimedia Syst.*, vol. 15, no. 5, pp. 295–308, 2009. [Online]. Available: <http://dx.doi.org/10.1007/s00530-009-0167-z>
- [13] O. Ariel and R. Raphael, “Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length,” *J. ACM*, vol. 37, no. 3, pp. 607–625, 1990. [Online]. Available: <http://doi.acm.org/10.1145/79147.214078>
- [14] E. W. Dijkstra, “Communication with an automatic computer,” Ph.D. dissertation, University of Amsterdam, 1959. [Online]. Available: <http://www.cs.utexas.edu/users/EWD/PhDthesis/PhDthesis.PDF>
- [15] O. Wiki, “Main page — openstreetmap wiki,” 2014, [Online; accessed 10-dezembro-2015]. [Online]. Available: [http://wiki.openstreetmap.org/w/index.php?title=Main\\_Page&oldid=1060762](http://wiki.openstreetmap.org/w/index.php?title=Main_Page&oldid=1060762)
- [16] T. Simples, “Tracking táxi simples,” 2015, [Online; accessed 5-novembro-2015]. [Online]. Available: [https://s3.amazonaws.com/txsimples\\_tracking/index.html](https://s3.amazonaws.com/txsimples_tracking/index.html)
- [17] “Geolocation api specification,” <http://dev.w3.org/geo/api/spec-source.html>, 2015, accessed: 2015-12-13.