Finding the Nearest Service Provider on Time-Dependent Road Networks

Lívia Almada Cruz, Francesco Lettich, Leopoldo Soares Júnior, Regis Pires Magalhães, and José Antônio Macedo

Federal University of Ceará, Computer Science Department, Brazil

livia.almada@ufc.br, francesco.lettich@gmail.com, leopoldosmj@lia.ufc.br, regismagalhaes@ufc.br, jose.macedo@dc.ufc.br

Abstract. In this paper we present a novel approach to solve the *Time-Dependent Nearest Server* problem, i.e., the problem of finding a *service provider* that, given a requesting user, can reach the user's geographical location as early as possible while taking into account traffic conditions. We also present a reduction of the above problem to the *fastest-path* problem to prove the correctness and bound the time complexity of our approach. In the experimental evaluation we compare our approach with respect to the state-of-art, and show its effectiveness in terms of accuracy and performance.

Keywords: Nearest Neighbor Queries, Time-dependent Networks, Location-based Services, Time Dependent Nearest Server

1 Introduction

Large amounts of vehicles in big urban agglomerations usually create serious mobility issues, such as congestion. Indeed, traffic conditions can change considerably over time, ranging from rush hours to regular hours, thus having relevant impacts on travel times. Consequently, taking into account traffic conditions is essential to improve the quality of location-based services, for example by providing better route plans.

In this context, we model a road-network as a *time-dependent* road-network – i.e., the costs associated with the edges of the network can vary over time – and attempt to solve a *variation* of the problem of computing nearest neighbor queries over time-dependent networks, i.e., the *Time-Dependent Nearest Server (TDNS) problem*[1]); here, the goal is to find out a service provider (e.g. taxis, ambulances, firefighters, etc.) that can reach some requesting user in the shortest amount of time while taking into account traffic conditions.

Figure 1 provides an example of the applicability of TDNS queries: let us suppose there is a user issuing an emergency call and a system has to find out which ambulance, among those available, can reach the user's location in the shortest amount of time. In this context, we model the network by means of a time-dependent network, where streets get associated with edges. Ambulances moving over the network may require to update frequently their positions. Also, to take into account traffic conditions each edge is associated with a time-dependent cost function that determines the travel time needed to cross the edge at the time instant an object starts traversing it (e.g., rush hours typically imply higher costs).

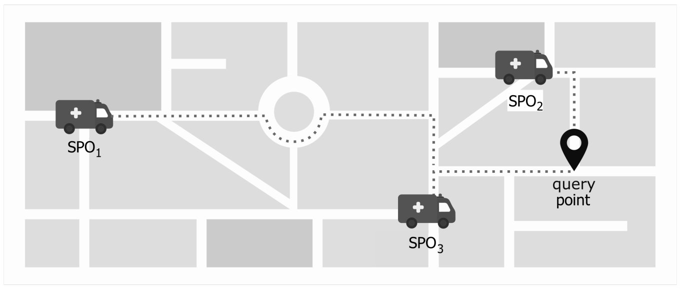

Fig. 1: Example of nearest service provider objects.

In this paper, we propose a new method to solve TDNS queries. Differently than the algorithm proposed in [1], our solution returns exact travel times. Furthermore, our solution can be easily extended to manage frequent position updates, thus allowing on-line query processing. Finally, we show how we reduce the TDNS problem to the *time-dependent fastest path* problem and prove the correctness of the reduction; we note that the reduction is conducted in linear time and thus has minor impacts on the performance of our approach.

The rest of the paper is structured as follows: Section 2 provides the definitions and the problem statement. Section 3 reviews the related works. Section 4 illustrates the approach we propose to compute TDNS queries. Finally, Section 5 presents the experimental evaluation, while Section 6 draws the final conclusions.

2 Preliminaries

A **time-dependent graph** (TDG) $G = (V, E, C)$ is a directed graph where $V = \{v_1, \ldots v_n\}$ represents a set of *vertices*, $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ represents a set of *edges*, and $C = \{c_{(v_i,v_j)}(\cdot) \mid (v_i, v_j) \in E\}$, with $c_{(v_i,v_j)} : [0, T] \to R^+$, represents a set of *cost functions*. Each cost function attributes a positive weight to the associated edge, depending on the *time instant* $t \in [0, T]$ provided to the function (T is a domain-dependent time length).

Since G is directed, bidirectional segments may have associated different edge costs, i.e., $c_{(u,v)}(t) \neq c_{(v,u)}(t)$. Furthermore, we assume that C represents a set of piecewise linear functions that satisfy the *FIFO property*, i.e., if an object

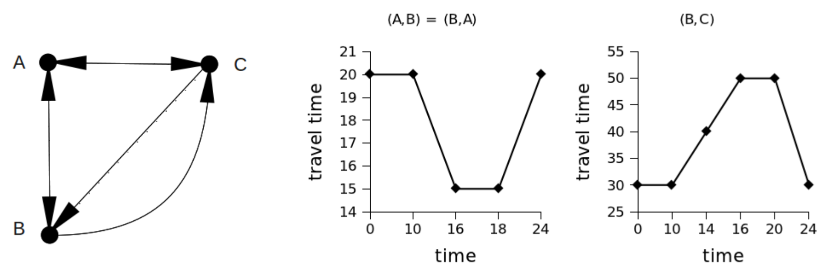

Fig. 2: A time-dependent graph (*left plot*) and examples of cost function for the edges (A, B) (*center plot*), (B, A) (*center plot*) and (B, C) (*right plot*).

A starts crossing an edge before an object B, then A has to finish crossing the edge before B. We remind that the problem of finding the fastest path in time-dependent networks that satisfy the FIFO property has a *polynomial* time solution, while it is NP-hard in non-FIFO networks [2]. Figure 2 presents a TDG and an example of travel time function that satisfies the FIFO property.

Given a TDG $G = (V, E, C)$, a *path* $P = \langle v_{p_1} \cdots, v_{p_{n-1}}, v_{p_n} \rangle$ in G and a *departure time* $t \in [0, T]$, we define the **travel time (TT)** of P to be the time-dependent cost to traverse P; this cost is defined as the sum

$$TT(P, t) = \sum_{i=1}^{n-1} c_{(v_{p_i}, v_{p_{i+1}})}(t_i),$$

where $t_1 = t$ and $t_{i+1} = (t_i + c_{(v_{p_i}, v_{p_{i+1}})}(t_i))$ *mod* T. The rationale behind the use of *modulo* is to deal with time cycles (e.g., a day implies $T = 24$ hours). We also define the **time-dependent fastest path (TDFP)** from a vertex v to a vertex u at time t to be the path with the minimum travel time; we denote it by $TDFP(v, u, t)$.

A **Point of Interest (POI)** represents an object within the underlying network that can be characterized by its geographic coordinates (latitude and longitude). In most of the works present in the literature a POI represents just a static entity, such as a hospital, a restaurant, and so on. In our case, however, POIs represent moving objects acting as service providers. For simplicity, from here on we use the term point of interest (POI) to refer to a service provider. At this point we can provide the *problem statement.*

Definition 1 (Problem statement). *Given a time-dependent graph $G = (V, E, C)$, a query point q, a time-instant t, and a set P of moving objects (POIs) in G, the* ***Time-Dependent Nearest Server (TDNS)*** *problem requires to retrieve the object $p \in P$ such that, $\forall p' \in P$, $p' \neq p$, we have that $TT(TDFP(p, q, t), t) \leq TT(TDFP(p', q, t), t)$.*

3 Related Work

The first algorithm that considers a time-dependent variant of the shortest path problem was proposed in [3]. Here, the authors present a modified form of the Bellman's iteration scheme to find the shortest route between any two vertices in a network. In [4] the authors propose an approach that is based on a bidirectional A^* search with landmarks to solve the time-dependent shortest path problem.

Other related works solve the problem of finding the k nearest neighbors in time-dependent networks, where the goal is to find the first k POIs that can be reached from the query point in the shortest amount of time (*time-dependent* k*-NN* query). The problem was first introduced in [5], where the authors propose and compare two different approaches: the first approach relies on time-expanded graphs, which allow to exploit previous solutions in the domain of static networks to solve the considered problem. The second approach is based on the INE algorithm [6], which expands the search starting from the query point until it finds the k nearest neighbors.

Another solution [7] attacks the above problem by adopting a schema based on an incremental expansion that incorporates an A^* search with a successive pruning phase; here, the authors leverage an heuristic function that works well with time-dependent networks. The intuition behind the approach is to discard vertices that are near the query point but far from any point of interest. Finally, upper-bounds on travel times are used to prune unpromising paths.

Others works propose solutions that deal with *variations* of the problem of computing k-NN queries over time-dependent networks. In [8], the authors propose A^*-based solutions to compute k-NN queries where the *operating times* of points of interest (e.g., opening/closure time of a commercial center) are taken into account. In time-dependent k-NN queries, we are interested to go from a given query point to a set of *fixed* POIs in the lowest possible time. The problem we consider in this work is thus different, in that POIs represent *moving* service providers.

Finally, in [1] the authors present the problem of finding the nearest service provider (POI) in time-dependent networks. They propose two solutions: the first one is a naïve approach that executes the fastest path algorithm between a query point q and each POI in P, and returns the solution having the minimum cost. The second solution is based on a breadth-first search conducted on a *reverse* graph: the reverse graph is obtained from the original graph by reverting the set of edges, i.e., an edge (b, a) exists in the reverse graph if and only if (a, b) exists in the original graph. The experimental evaluation shows that the second solution outperforms the naïve solution and exhibits good execution times; however, since the algorithm exploits reverse graphs, travel times associated with paths may be different than the travel times in the original graph.

Finally, we mention that this solution requires each POI to be associated with a node in the original graph, which in turn represents a potential bottleneck during the processing.

4 Competitive Network Expansion

The goal of this section is to introduce our proposal, i.e., the *competitive network expansion* (CNE), and the key ideas that brought to its conception. The *Competitive Network Expansion* algorithm considers the *original* graph to compute the cost of the paths from a set of service providers (POIs) to a query point q – as such, our algorithm is guaranteed to always return *exact* costs. A naïve solution would calculate the fastest paths between all the POIs and q, and pick up the fastest one. Instead, we base our iterative search on the idea of computing the paths from all the POIs in a *competitive* way, where at each step the POI having the highest chance to be q's nearest neighbor acquires the most preference.

4.1 Competitive Search

Given a set of POIs P, we apply a *search algorithm* to select the nearest POI among the candidates. Our search step is based on a reduction of the TDNS problem to the fastest path problem. Before delving into the inner workings of the search step, we need to show how the reduction is carried on. The idea behind the reduction is that, by including a virtual node vn in G connected to all the POIs in P, and by adding in E zero-cost edges that connect vn to all the POIs in P, we have that the fastest path between vn and q passes through q's nearest POI.

Theorem 1. *Let $G = (V, E, C)$ be a TDG and $PC = \{pc_1, \cdots pc_\lambda\}$, $PC \subseteq P$, be a set of POIs; let us obtain from G a transformed graph $G' = (V', E', C')$ by including a virtual node, vn, and a set of virtual edges, that is, $V' = V \cup \{vn\}$, $E' = E \cup \{(vn, pc_1), \cdots, (vn, pc_\lambda)\}$, and $C' = C \cup \{c_{(vn,pc_1)} = 0, \cdots, c_{(vn,pc_\lambda)} = 0\}$. If the time-dependent fastest path from vn to q pass through $pc \in PC$, and pc is the first POI in PC that appears in the path, then pc is the TDNS of q.*

Proof. Let pc be the first POI in the time-dependent fastest path from vn to q. If pc is the first POI, then pc is the second vertex in this path, because vn has edges only with POIs in PC. Furthermore, the cost of the fastest path from vn to q at time t is $\text{TDFP}(vn, q, t) = c_{(vn,pc)}(t) + \text{TDFP}(pc, q, t) = \text{TDFP}(pc, q, t)$, since $c_{(vn,pc)}(t) = 0$. Let us suppose that pc is not the TDNS of q: then, there must be another POI $pc' \in PC$ that is the TDNS of q, since $\text{TDFP}(pc', q, t) < \text{TDFP}(pc, q, t) = \text{TDFP}(vn, q, t)$; however this is impossible, since this would imply that there exists a faster path than the fastest one.□

At this point we can describe the search step of CNE. Our approach uses a variant of the A* algorithm [9] to find out the fastest path from the virtual node to the query point. In this context we use the travel time, in conjunction with a *heuristic* function, $h(.)$, to determine the order in which vertices are considered during the search. We note that the heuristic function must be *admissible*, i.e., it must not overestimate the distance from the vertex to the goal; as such, we recur to a lower bound distance, i.e., given any vertex u and the query point q, we define $h(u) = \frac{d_E(u,q)}{V_{max}}$, where $d_E(u,q)$ is the Euclidean distance from u to q, and V_{max} is the maximum velocity allowed on the network. The pseudo-code of CNE is provided in Algorithm 1.

Algorithm 1: Competitive Network Expansion

```
Input  : A TDG $G = (V, E, C)$, a query point $q$, a departure time $t$, a set of POIs $PC$, the
         number of candidates $\lambda$.
Output: The reverse nearest neighbor of $q$
begin
    $visited \leftarrow$ InitQueue$(G, PC, q, t)$
    while $visited \neq \emptyset$ do
        $(u, AT_u, TT_u, L_u) \leftarrow$ Dequeue$(visited)$
        Mark $u$ as DEQUEUED
        if $u = q$ then
            return $(u, AT_u, TT_u, L_u)$
        foreach $v \in Neighbors(u)$ do
            $TT_v \leftarrow TT_u + c_{(u,v)}(AT_u))$
            $AT_v \leftarrow (t + TT_v)$ mod $T$
            $L_v \leftarrow TT_v + \frac{d_E(v,q)}{V_{max}}$
            if $v$ is not marked as VISITED then
                Mark $v$ as VISITED
                $visited \leftarrow$ Enqueue$((v, AT_v, TT_v, L_v), visited)$
            else
                $(v, AT'_v, TT'_v, L'_v) \leftarrow$ getEntry$(visited, v)$
                if $L_v < L'_v$ then
                    UpdateEntry$(visited, (v, AT_v, TT_v, L_v))$
```

Algorithm 2: InitQueue(G, PC, q, t)

```
Input  : A TDG $G = (V, E, C)$, a set of POIs, $PC = \{pc_1, \cdots pc_\lambda\}$, a query point $q$, and a
         departure time $t$.
Output: The queue initialized with candidate POIs.
begin
    $queue \leftarrow \emptyset$
    foreach $poi \in PC$ do
        $TT_{poi} \leftarrow 0$
        $AT_{poi} \leftarrow t$
        $L_{poi} \leftarrow \frac{d_E(poi,q)}{V_{max}}$
        $queue \leftarrow$ Enqueue$((poi, AT_{poi}, TT_{poi}, L_{poi}), queue)$
        Mark $poi$ as VISITED
    return $queue$
```

First, with respect to the original A* algorithm we change the way the queue is initialized (InitQueue, line 2); more precisely, in G' we have that the costs associated with the edges between vn and the POIs in PC are zero. As such, we can ignore the additional edges added in G' and start the search directly from the POI candidates, since the reduction happens in linear time on size of PC

(by virtue of Theorem 1). Consequently, between lines 2 and 8 of InitQueue (Algorithm 2) each candidate p gets included in a priority queue, *visited*, which is sorted according to L_p, i.e., the distance from the virtual node, vn, to p, plus $h(p)$. Subsequently, CNE goes on until *visited* gets empty (line 3) or the query point is reached (lines 6-7). As in the case of original A*-search, at each iteration one node is dequeued and marked accordingly (lines 4-5), while the neighbors that were not marked as VISITED get enqueued (line 14). If a neighbor was already visited, and its new travel time is better than the current one, then its state get updated (lines 17-18).

4.2 Reducing the number of POI candidates

According to Theorem 1, CNE is an exact algorithm. However, expanding all the POIs in the network is unnecessary, since this would make the solution naïve. As such, we suggest to perform a *candidate generation* phase before executing Algorithm 1; the goal of this phase is to produce a set of candidate POIs according to the (Euclidean) distance between the POIs and the query point.

During the candidate generation phase, we compute k-NN queries between the query point and the POIs – to this end we observe that, the lower the number of candidates, the smaller the chance that the correct result will be returned by CNE; Section 5 studies the impact that this factor has on the quality of results. We also maintain an R-Tree that is used to index the vertices of the graph. Since search algorithms typically suppose that POIs are associated with the vertices of the graph, the goal of this data structure is to facilitate the related map-matching.

Overall, the candidate generation step helps to decrease the execution time of CNE and the time to map-match POIs.

5 Experimental Evaluation

The goal of this section is two-fold: first, we assess the accuracy of CNE and to do so we use the output returned by the Naive algorithm as our ground truth – we remind that the Naive approach computes the fastest paths between the POIs and the query point and picks up the fastest among these. Subsequently, we compare CNE with Breadth-First Search (BFS) in reverse graphs [1], the state-of-the-art in the current literature.

5.1 Setup

For the purposes of this work, all the approaches were implemented on top of the *Graphast* framework [10] and a public implementation of R-Trees[1]. The experiments are conducted on a virtual machine, provided by Amazon AWS, with 2 Intel Xeon CPUs clocked at 2.4 GHz, 8 GB of RAM, and Ubuntu OS (16.04).

[1] https://github.com/davidmoten/rtree

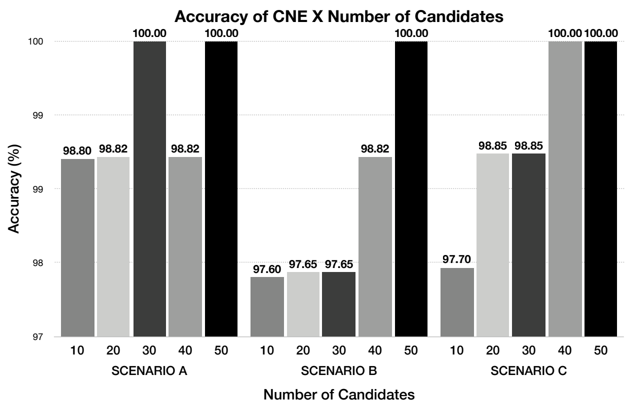

Fig. 3: CNE accuracy – variable *number of candidates*.

We use the road network related to the city of Fortaleza, obtained by collecting the data available from the *Open Street Map Project*[2]. The graph representing the network contains around 100.000 nodes, while travel time functions are synthetically generated. In the experiments, we consider three distinct scenarios involving real-world POIs, where we vary their density and distribution; more specifically, we consider taxis positions (A) at 7am with 360 POIs, (B) at 12am with 216 POIs, and (C) at 5pm, with 398 POIS. The set of POI locations was provided by *Taxi Simples*, a company that tracks and monitors cabs activities[3]; POI positions span one day (July 23, 2016).

5.2 Evaluation

In the first batch of experiments we want to evaluate the impact that the number of candidates has on CNE's accuracy; in general we expect that, the less the candidates provided, the less the resulting accuracy. To this end we compare the results returned by CNE with the ones returned by the NAIVE approach. Figure 3 presents the accuracy achieved by CNE when using 10, 20, 30, 40, and 50 candidates.

From the Figure 3, we see that the minimum accuracy achieved by CNE is 97.65% (scenario B, 10 candidates). With 50 candidates, all POIs returned were the correct nearest service provider. We also report that once the number of candidates gets above 50, CNE was always able to achieve total correctness (100%).

In the second batch of experiments we evaluate the execution time of CNE, NAIVE and BFS. We consider, again, the three scenarios introduced before, and measure the CPU time needed to *update* the POI locations and run the *search*.

[2] http://wiki.openstreetmap.org/wiki/Main_Page

[3] http://www.taxisimples.com.br/

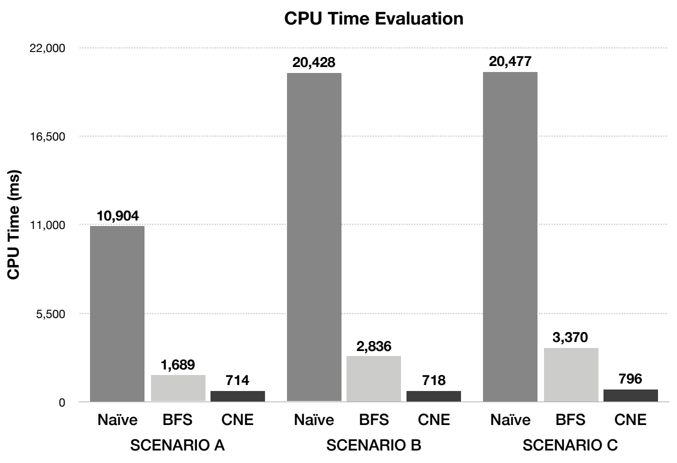

Fig. 4: CPU time comparison between CNE and BFS.

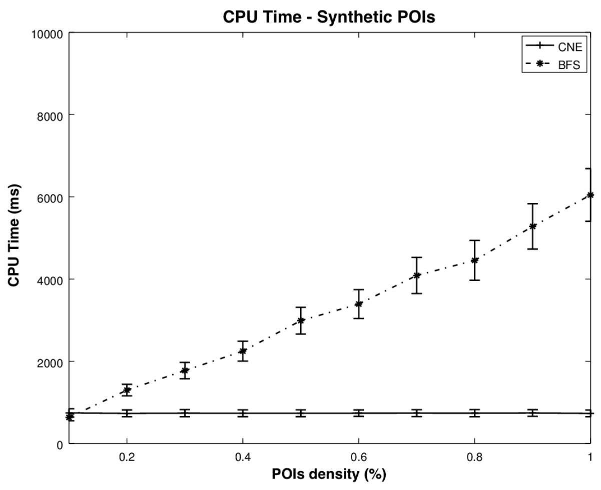

Fig. 5: CPU time comparison between CNE and BFS – variable *POI density*.

In this context, we also enforce CNE to produce 50 candidates. Each test is repeated 100 times. Figure 4 presents the results. From this figure, we see that CNE always outperforms the competitors; the average CPU times achieved by Naive when considering the scenarios A, B, and C were, respectively, 10.504 ms, 20.429 ms and 20.477 ms, almost two orders of magnitude greater than the times achieved by CNE.

If we compare CNE with BFS, we see that the speedups yielded by our approach are equal, respectively for the scenarios A, B, and C, to 2.4×, 3.9× and

4.2×. Finally, we observe that the execution time of CNE remains always below 800 ms and that it exhibits an almost constant trend: as such, we argue that the number of POIs in the network has a minimal impact on CNE's performance.

In the third, final batch of experiments we evaluate how the performance of CNE is affected when the POI density (i.e., the rate of POIs per node) increases. We generate 10 sets of synthetic POIs in the network, in the $[0,1\%; 1\%]$ density range, and compare the CPU time of CNE and BFS. We also enforce CNE to produce 50 candidates at the end of the candidate generation phase. Figure 5 shows the average time and standard deviation obtained with 100 executions.

From the Figure 5, we see that the CPU time of CNE remains constant – thus indicating that the candidate generation phase makes CNE scalable with respect to the POI density – while the CPU time of BFS increases linearly.

6 Conclusions and Future Work

In this paper we propose a new approach to solve nearest service provider queries in time-dependent road networks (TDNS). Among the contributions, we provide a reduction of the TDNS problem to the fastest path problem, and prove its correctness. Thanks to this, it is possible to solve our query by applying minor modifications to the A*-search algorithm [9].

Another contribution consists in showing that the application of a candidate generation phase allows to quickly update POI locations with negligible effects on the correctness of the final results. Indeed, in the experimental evaluation we show that our solution achieves at least 97% of accuracy with real-world datasets, even when using a very low amount of candidates.

Finally, in the experimental evaluation we show that our approach performs better than the state-of-the-art currently available in the literature [1].

As a future direction of research, we plan to improve the way candidate POIs are chosen; we also plan to solve the problem of computing a *continuous* version of the TDNS query, where we continuously monitor the service providers to identify the best one in real-time. Finally, we plan to investigate how we can compute TDNS queries over dynamic networks, where travel-time functions can be updated over time.

References

1. Chucre, M., Nascimento, S., Macedo, J.A.F., Monteiro, J.M.D.S., Casanova, M.A.: Taxi, please ! a nearest neighbor query in time-dependent road networks. 17th IEEE Int. Conf. on Mobile Data Management (2016)
2. Orda, A., Rom, R.: Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. J. ACM **37**(3) (July 1990) 607–625
3. Cooke, K.L., Halsey, E.: The shortest route through a network with time-dependent internodal transit times. Journal of mathematical analysis and applications **14**(3) (1966) 493–498

4. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional a* search for time-dependent fast paths. In: International Workshop on Experimental and Efficient Algorithms, Springer (2008) 334–346
5. Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Towards k-nearest neighbor search in time-dependent spatial network databases. In: International Workshop on Databases in Networked Information Systems, Springer (2010) 296–310
6. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: Proceedings of the 29th international conference on Very large data bases-Volume 29, VLDB Endowment (2003) 802–813
7. Cruz, L.A., Nascimento, M.A., Macedo, J.A.F.: K-nearest neighbors queries in time-dependent road networks. JIDM **3**(3) (2012) 211–226
8. Costa, C.F., Nascimento, M.A., De Macêdo, J.A.F., Machado, J.: A*-based solutions for knn queries with operating time constraints in time-dependent road networks. In: 2014 IEEE 15th International Conference on Mobile Data Management. Volume 1., IEEE (2014) 23–32
9. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A* search meets graph theory. In: Proceedings of the Sixteenth Snnual ACM-SIAM Symposium on Discrete algorithms, Philadelphia, PA, USA (2005) 156–165
10. Magalhães, R.P., Coutinho, G., Macêdo, J., Ferreira, C., Cruz, L., Nascimento, M.: Graphast: an extensible framework for building applications on time-dependent networks. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM (2015) 93