UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MIRLA RAFAELA RAFAEL BRAGA CHUCRE

K-NEAREST NEIGHBORS QUERIES IN TIME-DEPENDENT ROAD NETWORKS: ANALYZING SCENARIOS WHERE POINTS OF INTEREST MOVE TO THE QUERY POINT

FORTALEZA, CEARÁ

2015

MIRLA RAFAELA RAFAEL BRAGA CHUCRE

K-NEAREST NEIGHBORS QUERIES IN TIME-DEPENDENT ROAD NETWORKS: ANALYZING SCENARIOS WHERE POINTS OF INTEREST MOVE TO THE QUERY POINT

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientador: Prof. Dr. José Maria da Silva Monteiro Filho

Co-Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

FORTALEZA, CEARÁ

2015

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C484k Chucre, Mirla Rafaela Rafael Braga.
K-NEAREST NEIGHBORS QUERIES IN TIME-DEPENDENT ROAD NETWORKS: ANALYZING SCENARIOS WHERE POINTS OF INTEREST MOVE TO THE QUERY POINT / Mirla Rafaela Rafael Braga Chucre. – 2015.
65 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2015.
Orientação: Prof. Dr. Jose Maria da Silva Monteiro Filho.
Coorientação: Prof. Dr. Jose Antonio Fernandes de Macedo.

1. Processamento de consultas espaciais. 2. Redes dependentes do tempo. 3. Consultas de vizinho mais proximo. 4. TD-kNN Queries. 5. Time-dependent networks. I. Título.

CDD 005

MIRLA RAFAELA RAFAEL BRAGA CHUCRE

K-NEAREST NEIGHBORS QUERIES IN TIME-DEPENDENT ROAD NETWORKS: ANALYZING SCENARIOS WHERE POINTS OF INTEREST MOVE TO THE QUERY POINT

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. José Antônio Fernandes de Macêdo
(Co-Orientador)
Universidade Federal do Ceará - UFC

Prof. Dr. Marco Antônio Casanova
Pontifícia Universidade Católica do Rio de Janeiro - PUC - RJ

As minhas filhas.

AGRADECIMENTOS

Agradeço aos meus pais, Maria Lúcia e Antonio Braga, pelo grande amor e com ele poderam transformar em dedicação, dedicação para eu fazer obter educação, possibilitando assim chegar aqui.

Aos meus irmãos Rafael e Mirlane que ao longo dos anos sempre compartilhando boas experiências e apoio.

Ao meu marido Fernando por todo o apoio dado durante meu mestrado e sempre ter acredito nessa conquista.

Ao Professor José Maria pelo acompanhmento na pesquisa, oportunidades dadas, orientação, além de um suporte excepcional.

Ao Professor José Antônio pelo acompanhmento na pesquisa, oportunidades dadas, orientação, além de um suporte excepcional.

Ao Prof. Marco Antonio Casanova pela disponibilidade em participar da banca avaliadora, pela a receptividade na PUC-RJ. Assim, possibilitou contribuir com seus sólidos conhecimentos para este e para futuros trabalhos.

Por fim, todos os colegas e professores do grupo de pesquisa ARIDa, por todos os momentos compartilhados durante esta etapa. Em especial, a Samara que cotidianamente compartilhamos os desafios superados.

“Tentar não significa conseguir, mas todos que conseguiram tentaram.”

(Autor Desconhecido)

RESUMO

Uma consulta de vizinhos mais próximos (ou *k*NN, do inglês *k* nearest neighbours) recupera o conjunto de k pontos de interesse que são mais próximos a um ponto de consulta, onde a proximidade é computada do ponto de consulta para cada ponto de interesse. Nas redes de rodovias tradicionais (estáticas) o custo de deslocamento de um ponto a outro é dado pela distância física entre esses dois pontos. Por outro lado, nas redes dependentes do tempo o custo de deslocamento (ou seja, o tempo de viagem) entre dois pontos varia de acordo com o instante de partida. Nessas redes, as consultas *k*NN são denominadas TD-*k*NN (do inglês *Time-Dependent k*NN). As redes de rodovias dependentes do tempo representam de forma mais adequada algumas situações reais, como, por exemplo, o deslocamento em grandes centros urbanos, onde o tempo para se deslocar de um ponto a outro durante os horários de pico, quando o tráfego é intenso e as ruas estão congestionadas, é muito maior do que em horários normais. Neste contexto, uma consulta típica consiste em descobrir os *k* restaurantes (pontos de interesse) mais próximos de um determinado cliente (ponto de consulta) caso este inicie o seu deslocamento ao meio dia. Nesta dissertação nós estudamos o problema de processar uma variação de consulta de vizinhos mais próximos em redes viárias dependentes do tempo. Diferentemente das consultas TD-*k*NN, onde a proximidade é calculada do ponto de consulta para um determinado ponto de interesse, estamos interessados em situações onde a proximidade deve ser calculada de um ponto de interesse para o ponto de consulta. Neste caso, uma consulta típica consiste em descobrir os *k* taxistas (pontos de interesse) mais próximos (ou seja, com o menor tempo de viagem) de um determinado cliente (ponto de consulta) caso eles iniciem o seu deslocamento até o referido cliente ao meio dia. Desta forma, nos cenários investigados nesta dissertação, são os pontos de interesse que se deslocam até o ponto de consulta, e não o contrário. O método proposto para executar este tipo de consulta aplica uma busca A^* à medida que vai, de maneira incremental, explorando a rede. O objetivo do método é reduzir o percentual da rede avaliado na busca. A construção e a corretude do método são discutidas e são apresentados resultados experimentais com dados reais e sintéticos que mostram a eficiência da solução proposta.

Keywords: Processamento de consultas espaciais. Redes dependentes do tempo. Consultas de vizinho mais próximo.

ABSTRACT

A *k*NN query retrieve the *k* points of interest that are closest to the query point, where proximity is computed from the query point to the points of interest. Time-dependent road networks are represented as weighted graphs, where the weight of an edge depends on the time one passes through that edge. This way, we can model periodic congestions during rush hour and similar effects. Travel time on road networks heavily depends on the traffic and, typically, the time a moving object takes to traverse a segment depends on departure time. 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 Paris who is interested to visit the touristic attraction closest from him/her. Let us consider two points of interest in the city, the Eiffel Tower and the Cathedral of Notre Dame. He/she asks a query asking for the touristic attraction whose the path leading up to it is the fastest at that time, the answer depends on the departure time. For example, at 10h it takes 10 minutes to go to the Cathedral. It is the nearest attraction. Although, if he/she asks the same query at 22h, in the same spatial point, the nearest attraction is the Eiffel Tower. In this work, we identify a 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. With this approach, 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. Previous works have proposed solutions to answer *k*NN queries considering the time dependency of the network but not computing the proximity from the points of interest to the query point. We propose and discuss a solution to this type of query which are 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.

Keywords: TD-kNN Queries. Spatial querying process. Time-dependent networks

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUCTION

This chapter is organized as follows. The motivation for this work is presented in Section 1.1. In Section 1.2 the general and specific objectives are presented. Finally, Section 1.3 presents the contributions and Section 1.4 concludes the chapter.

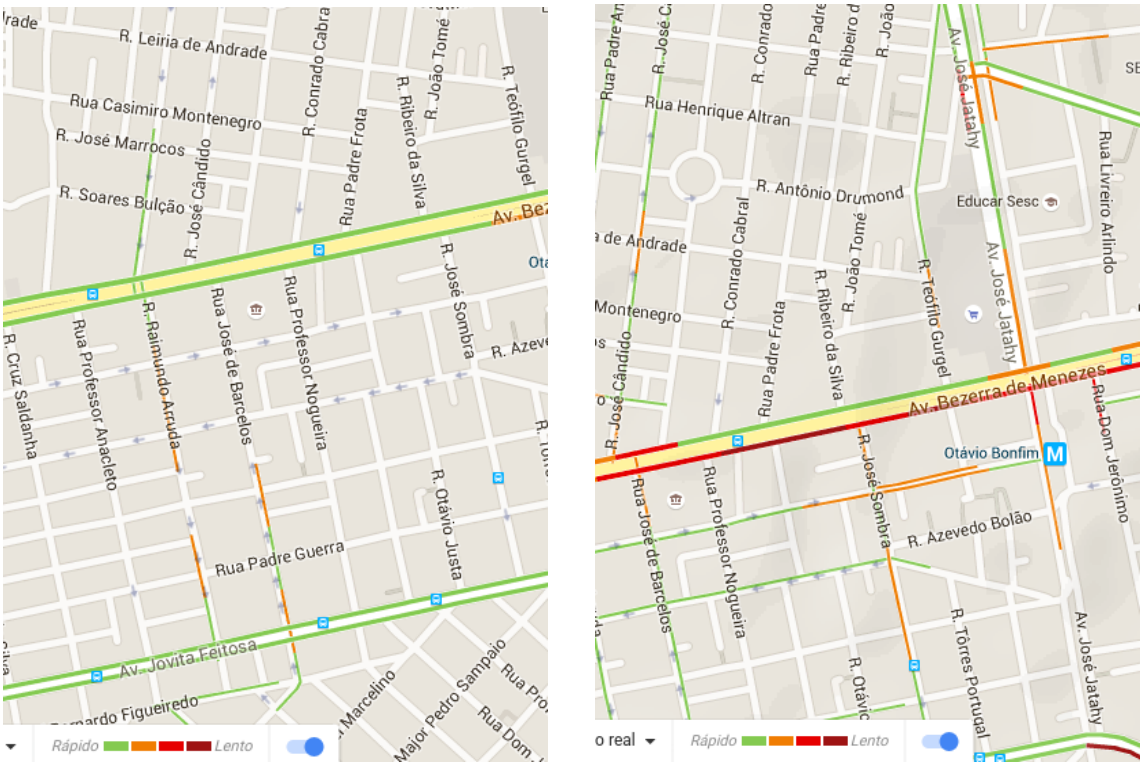
1.1 MOTIVATION

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.

Currently, traces modern methods allow you to capture a lot of points of moving objects. With the high availability of inexpensive tracking devices, such as GPS-enabled devices, besides the traffic sensors positioned in road segments in several countries, it is possible to collect large amounts of trajectory data of vehicles. Using such data along with the underlying road network information allows creating an accurate picture of the traffic conditions in time and space. Thus, it becomes feasible to model the dependence of traveling speed on the time of the day based on historical traffic data. Given this information it is possible to analyze and provide location-based services and to solve complex spatio-temporal queries. One possible use of this information is to compute more realistic travel time forecasting from an origin to a destination, which is a major issue for the intelligent transportation domain.

With the increasing interest in intelligent transportation, more complex and advanced query types were recently proposed, such as nearest neighbors (kNN) queries (SHARIFZADEH et al., 2008; LI et al., 2005; CHEN et al., 2011; PAPADIAS et al., 2003; JENSEN et al., 2003; KOLAHDOUZAN; SHAHABI, 2004b; KOLAHDOUZAN; SHAHABI, 2005) and route planning queries (SHARIFZADEH et al., 2008; LI et al., 2005; CHEN et al., 2011). A kNN query returns the k points of interest that are closest to the query point, where proximity is computed from the query point to the points of interest. So, as a result, many navigation systems are now enhanced to support kNN research. People use these systems for decision making. For example, users may ask questions like "What is the shortest path to go from home to the University?"or "What is the nearest train station to a hospital?".

Though, the majority of previous solutions fails to represent the reality in the sense that they assume that road networks are static and the cost to traverse each edge is given by the length of this edge and, thus, it does not vary with time. This assumption is certainly not true on a real scenario, where the travel time heavily depends on the traffic and, typically, the time that a moving object takes to traverse a segment depends on departure time. Figure 1 shows a



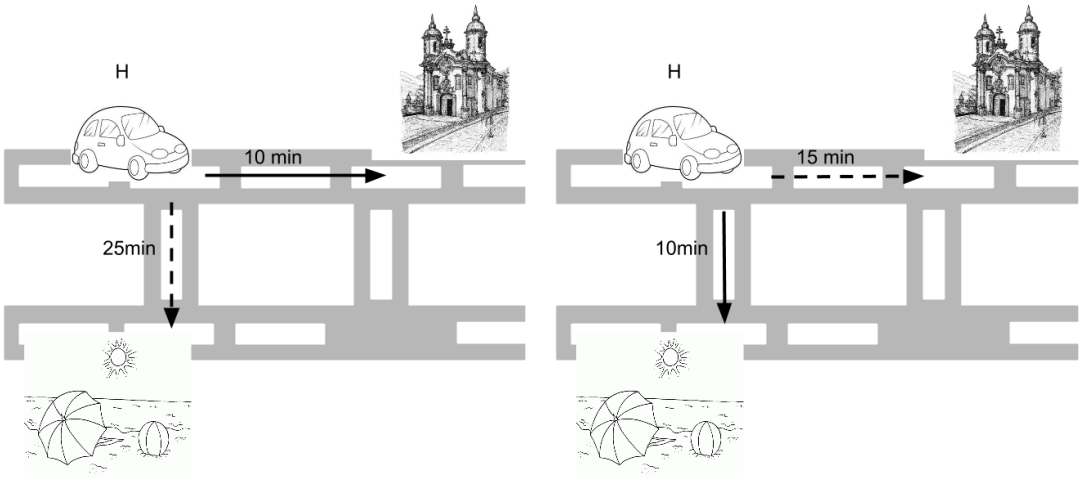
(a) Traffic at 06:22h. (b) Traffic at 08:40h.

Figura 1 – Traffic on the Bezerra de Menezes avenue in Fortaleza, Brazil, at two different times of a day. Source: Google Maps (https://maps.google.com/).

real example of how traffic is influenced by the time of day and, consequently, the time spent to move from one point to another within a large city. It illustrates two different moments of a same avenue in Fortaleza, Brazil. At 19:30h this avenue is completely congested and, thus, the time to cross it is longer than at 20:40h, when the traffic is less intense.

Some recent studies have included the temporal dependence to solve conventional spatial queries, such as *k*NN (DEMIRYUREK et al., 2010c; DEMIRYUREK et al., 2010d; CRUZ et al., 2012) and shortest path (NANNICINI et al., 2012) 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 to visit 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 the path leading up to it is the fastest at that time, the answer depends on the departure time. For example, at 10h it takes 10 minutes to go to the Cathedral (see Figure 2a). It is the nearest attraction. Although, if he/she asks the same query at 22h, in the same spatial point, the nearest attraction is the Iracema Beach (see Figure 2b).

According to (CRUZ et al., 2012), time-dependent road networks are represented as weighted graphs, where the vertices represent the network junctions, starting and ending points of a road segment (e.g. a street or an avenue), the edges connect vertices (depending on the application, additional points can represent a change in curvature or in maximum speed of a



(a) The nearest neighbor at 10h (b) The nearest neighbor at 22h

Figura 2 – An example of *k*NN query. The fastest path from the query point to the nearest neighbor is in solid line. The fastest path from the query point to the other point of interest is in dashed line

segment) and the weight (cost in time) of an edge is a function of the departure time. This way, we can model periodic congestion during rush hour and similar effects, for example.

Processing queries in time-dependent road networks is challenging due to a number of reasons. The space required to store these networks is significantly larger than the space used to store the time-independent ones, since it is necessary to keep, for each edge, the cost to traverse it for every time interval of a day. Furthermore, solutions for conventional queries in static networks can not be directly applied to solve the time-dependent problems. Particularly, it is complicated to apply the well-known speed-up technique of bi-directional search, that starts a search simultaneously from the source and the target, to solve the shortest path problem in a time-dependent network since the arrival time would have to be known in advance for such a procedure.

In this work, we identify a 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. Figure 3 illustrates how a TD-*k*NN query can be used. Consider two different scenarios, one at 8h30 and another at 14h. In both scenarios, there are two taxi drivers and different paths to reach the passenger. Taking into consideration the travel time to roaming the streets, the time of the call and the period of time that the passenger is willing to wait for the taxi, the proposed query should return the taxi driver who can answer first the customer call.

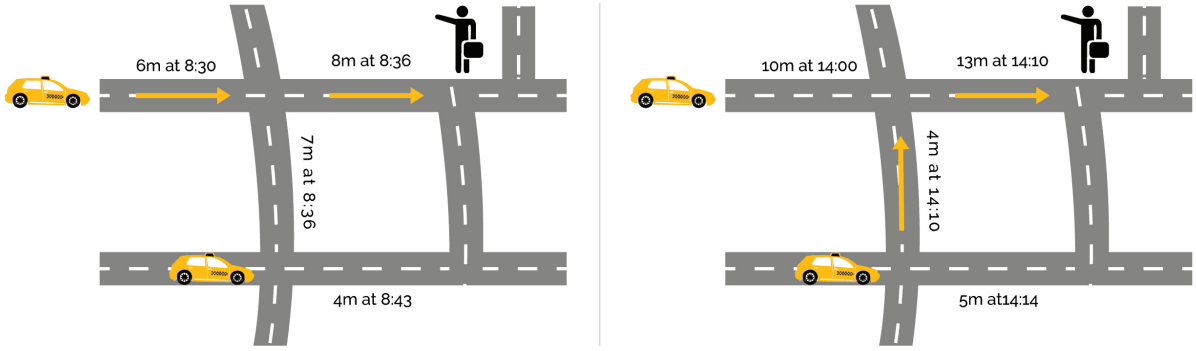


Figura 3 – An example of taxi call

Figure 4 shows a real example of how a cab company may be scattered in a city. This figure illustrates the location of the cabs, of a same company, in Fortaleza city at 10h.

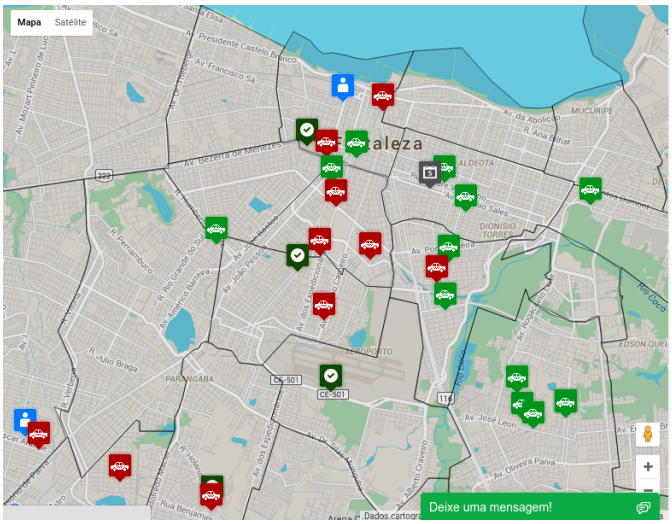


Figura 4 – Arrangement of taxis in the Fortaleza city. Source: (SIMPLES, 2015)

Previous works have proposed solutions to answer kNN queries considering the time dependency of the network but not computing the proximity from the points of interest to the query point. We propose and discuss a solution to this type of query which are 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 can note that in some cases, the answer returned by a regular TD-kNN query is coincidentally the same as returned by our query, for example, where all the roads allow bi-directional vehicles's flow and all conversions are allowed, what is unreasonable in real situations.

1.2 OBJECTIVES

Given the motivating scenario presented above, the general objective of this work is to study the problem of to find the k points of interest (e.g. cabs, ambulances or police cars) which can move to the query point (e.g. a taxi user or a citizen waiting for an emergency service) in the minimum amount of time, proposing efficient and optimal solutions to them. To achieve this objective, we established the following specific objectives:

- To propose a baseline solution, for comparison purposes, to the problem of minimizing the travel time from points of interest to the query point;
- To propose an efficient algorithm to solve the problem of minimizing the travel time from points of interest to the query point;
- To evaluate the proposed solutions to the problem of minimizing the travel time from points of interest to the query point and to indicate in which types of applications each one should be used;
- To specify and develop a new generator of synthetic networks, varying both the network size and the density of points of interest;
- To design and develop road networks based in real data about the streets and cabs positions in the Fortaleza city;

1.3 CONTRIBUTIONS

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. This solution find the k points of interest which can move to the query point in the minimum amount of time. 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-kNN 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.

The following items summarize the main contributions of this thesis:

- We propose and discuss a baseline solution, for comparison purposes, to the problem of minimizing the travel time from points of interest to the query point in time-dependent networks. The baseline solution is a variation of the algorithm proposed by Dijstra(DIJKSTRA, 1959);
- We propose an efficient algorithm to solve the problem of minimizing the travel time from points of interest to the query point in time-dependent networks;
- We evaluate the proposed solutions to the problem of minimizing the travel time from points of interest to the query point in time-dependent networks using synthetic and real data;
- We specify and develop a new generator of synthetic networks, varying both the network size and the density of points of interest;
- We design and develop road networks based in real data about the streets and cabs positions

in the Fortaleza city;

1.4 CONCLUSION

The next chapters of this thesis are structured as follows:

- Chapter 2 presents the key concepts involved in this work. It formalizes the concept of time-dependent graph and presents some definitions useful to formalize our problems. Furthermore, we discuss in details the A^* search (HART et al., 1968) and Incremental Network Expansion (INE) (PAPADIAS et al., 2003) algorithms, which are bases for our solutions and a method to access information about adjacency of vertices and history traffic in time-dependent networks.
- Chapter 3 presents the proposed solutions in details;
- Chapter 4 evaluates the proposed approaches;
- Chapter 5 discuss the related works;
- Chapter 6 concludes this thesis with a summary of our findings and some suggestions for further work.

2 THEORETICAL FOUNDATION

This chapter describes the concepts and models used to represent time-dependent networks and some of their properties. Section 2.1 shows the definitions of interest point and query point. Section 2.2 presents the graph used to model this network, called time-dependent graph, as well as some concepts necessary to the formulation of the problems investigated in this work. In Section 2.3 we discuss in details the Dijkstra, A^* search (HART et al., 1968) and the Incremental Network Expansion (INE) (PAPADIAS et al., 2003) algorithms, which are of key importance to our proposed solutions. Section 2.4 describes an efficient access method for time-dependent networks. In Section 2.5 we discuss some characteristics of the taxi business. Finally, Section 2.6 concludes this chapter.

2.1 PRELIMINARIES

We start this section discussing some terminologies used in this work. These terminologies include interest point and query point.

Definição 1 (Interest Object or Interest Point or Point of Interest - POI) *An interest object is any object on a network and it is of interest to users. An interest point is where the interest object is located. We use the terms interest objects and interest points interchangeably. We assume a road network containing a set of interest objects, such as cabs, ambulances, etc.*

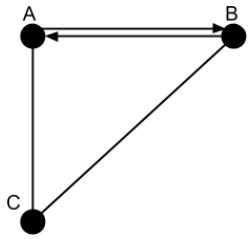
Definição 2 (Query Object or Query Point) *A query object is an object on the network and its influence on interest objects is determined as the query is called. A query point is where the query object is located. We use the terms query object and query point interchangeably.*

2.2 TIME-DEPENDENT GRAPH (TDG)

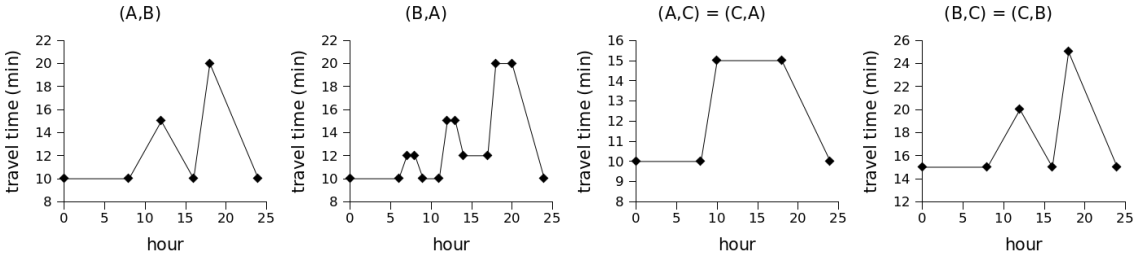
We consider that the structure of a time-dependent road network is modeled by a graph where the vertices represent the intersections of road segments. Those are connected by edges, which the cost to traverse vary with time. More formally, the network is modeled by a **time-dependent graph (TDG)** $G = (V, E, C)$, where V is a set of vertices, E is a set of edges and the cost (time in our domain of interest), represented by C, to traverse an edge is a function of the departure time. In other words, a TDG is a graph in which the costs of the edges varies with time. The concept of TDG is formally defined below.

Definição 3 *A **time-dependent graph** (TDG) $G = (V, E, C)$ is a graph where: (i) $V = \{v_1, ..., v_n\}$ 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 C is a set of functions 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



(a) A graph representing a road network.



(b) Cost functions, or travel time on edges.

Figura 5 – A graph representing a road network and the costs of its edges for different times of a day.

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 (ORDA; ROM, 1990a), 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 5 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 5b. 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.

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.

Definição 4 *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 $AT(v_i,v_j,t) = t + c_{(v_i,v_j)}(t)$ mod 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 5b, 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 for the calculation of the arrival-time be 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.

Definição 5 *Given a TDG G = (V,E,C), a path* $p = \langle v_{p_1}, ..., v_{p_k} \rangle$ *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 (COSTA et al., 2014). As an example, consider the path $\langle B,A,C \rangle$ in the graph shown in Figure 5a. At a departure time t = 10:00 am, the cost of traversing this path is given by $TT(\langle B,A,C \rangle$, 10:00) which is equal to 25 minutes, since the cost to go from B to A at 10 am is 10 minutes and the arrival time at A is 10:10 am and the cost to go from A to C at 10:10 am is 15 minutes.

We assume that points of interest as well as origin and destination points are located on a vertex throughout this thesis. On the original network, those points are not necessarily vertices, however, they can be transformed into new vertices of the network as shown in (CRUZ et al., 2012). That work proposes the IncludePOI algorithm which take as input a TDG G and a POI $p = \langle (u,v), \tau_p \rangle$, where (u,v) is the edge over which p is positioned and τ_p is a ratio which indicates how far p is from the begin of the edge (u). To illustrate how this algorithm works, let us consider that we want to include a new vertex (not necessarily a POI) represented by $p = \langle (C,A), \frac{1}{3} \rangle$ in the network shown in Figure 5a. The IncludePOI algorithm works as follows. It first inserts the new vertex p in the set of vertices V. Figure 6a shows the network with the inclusion of p. As p is a point over (C,A), (C,A) is removed from E and two new edges (C,p) and (p,A) are created. The travel time functions for (C,p) and (p,A) are $c_{(C,p)} = \frac{1}{3} c_{(C,A)}$ and $c_{(p,A)} = \frac{2}{3} c_{(C,A)}$ and the travel time function $c_{(C,A)}$ is removed from C. The graphics representing the travel times of the new edges are shown in Figure 6b. Next, as (A,C) is also in E, we need to repeat the same process executed for the edge (C,A). (A,C) is removed from E and the edges (A,p) and (p,C) are created. The new cost functions are $c_{(A,p)} = \frac{2}{3} c_{(A,C)}$ and $c_{(p,C)} = \frac{1}{3} c_{(A,C)}$ as shown in Figure 6b.

2.3 DIJKSTRA, INE AND A^* SEARCH ALGORITHMS

In this section we discuss in details how the Dijkstra, Incremental Network Expansion (INE) and the A^* search algorithms, bases for the proposed solutions in this thesis, work.

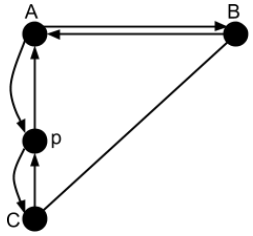
2.3.1 Dijkstra's algorithm

Dijkstra's algorithm is an classical algorithm to solve the problem of the shortest path from a given source to all other vertice in a directed graph with non-negative edge weights is due to Dijkstra (DIJKSTRA, 1959). The algorithm keep, for each vertice u, a label $distance[u]$ with the tentative distance from s to u. A priority queue Q contains all vertices that describe the current search horizon around s. At each step, the algorithm removes the node u from Q with minimum distance from s. So, all outgoing edges (u,v) of u are relaxed, i.e., we check if $d(s,u) + len(u,v) < distance[v]$ holds. If it holds, a shorter path to v via u has been found. In this way, v is either inserted to the priority queue or its priority is decreased.

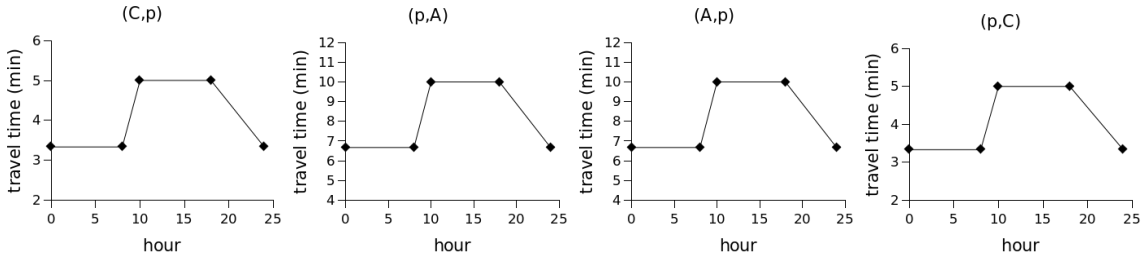
2.3.2 Incremental Network Expansion (INE)

The problem of processing k-nearest neighbor (k-NN) queries in road networks has been investigated since the pioneering study by (PAPADIAS et al., 2003), where the Incremental Euclidean Restriction (IER) and the Incremental Network Expansion (INE) methods were proposed.

The basic idea of the IER method is to first find the k POIs from the query point q



(a) A graph representing the network with the inclusion of the new vertex p.



(b) Cost functions of the new edges.

Figura 6 – A new graph representing the inclusion of the vertex p over the edge (A,C) and the travel time functions of the new edges created.

on the Euclidean distance using R-trees (GUTTMAN, 1984). Then, the network distances from q to these POIs are calculated and the distance to the farthest of these POIs is used as an upper bound. Next, all the POIs with an Euclidean distance from q less or equal to the upper bound are investigated, that is, their network distances are calculated, because they offer a chance to be part of the k-NN result. Figure 7 shows how this method works when one NN is required. IER first retrieves the Euclidean nearest neighbor p_{E1} of q. Then, the network distance $d_N(q, p_{E1})$ of p_{E1} is computed. This distance is then used as an upper bound, that is, all the objects closer (to q) than p_{E1} in the network, should be within Euclidean distance at most $d_N(q, p_{E1})$, and, thus, they should lie in the shaded area of the left figure. Next, as shown in the right figure, the second Euclidean NN, p_{E2}, is found. Similarly, the network distance $d_N(q, p_{E2})$ of p_{E2} is computed. Since $d_N(q, p_{E2}) < d_N(q, p_{E1})$, p_{E2} becomes the new NN and the upper bound is updated accordingly. As the distance to the next Euclidean NN p_{E3} is greater than $d_N(q, p_{E2})$, the algorithm stops and returns p_{E2} as the nearest neighbor (COSTA et al., 2014).

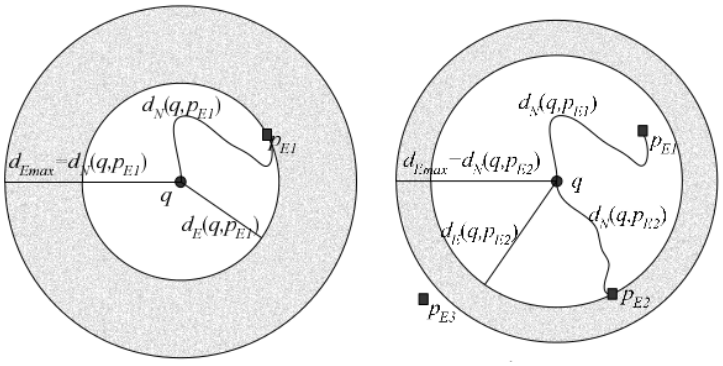


Figura 7 – Illustration of the Incremental Euclidean Restriction (IER) (PAPADIAS et al., 2003).

Clearly, the problem with this approach is that, generally, k-NN POIs on the Euclidean distance are not always k-NN on the road network distance, especially when time-dependent costs are considered. Thus, several false hits must be investigated. To remedy this problem, the Incremental Network Expansion (INE) algorithm was proposed. It performs network expansion and searches neighbor POIs by visiting vertices in order of their proximity from q, using Dijkstra's algorithm (DIJKSTRA, 1959), until all k nearest points of interest are located. Returning to the example shown in Figure 7, as p_{E2} is the NN from q considering the network distance, the INE algorithm first locates this POI without investigating p_{E1}. The blue shaded area in Figure 8 indicates the search area on the road network of a k-NN query with the INE approach for k = 10 (COSTA et al., 2013). As shown in this figure, the search area is enlarged from q until the k POIs have been found.

One drawback of this algorithm is that the search is not guided, i.e., the vertices are examined in order of proximity from q without any estimate for the cost of achieving the POIs. In order to guide the execution of this method, we incorporate an A* search to the INE expansion, being possible to discard the verification of paths that do not lead to the solution. We explain how this search works below.

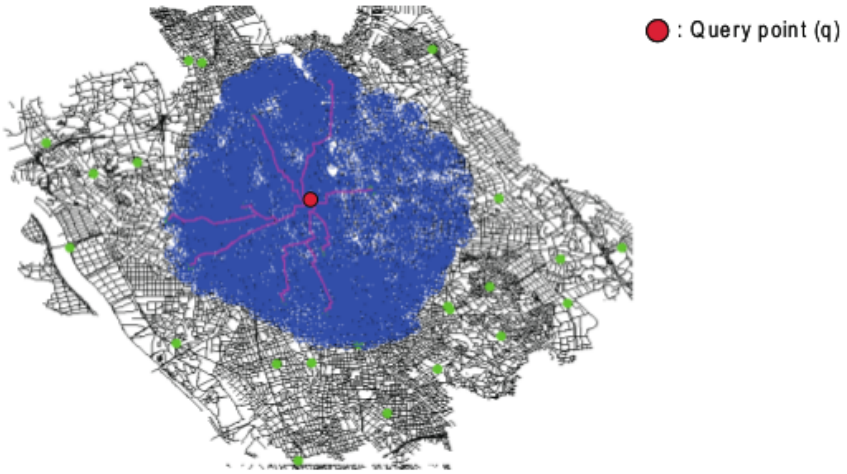


Figura 8 – Illustration of the Incremental Network Expansion (INE) for k = 10. Source: (HTOO, 2013).

2.3.3 A^* search

The A^* search is an algorithm that was originally proposed to find the shortest path from an origin to a goal node and it is similar to Dijkstra's algorithm. The main difference to this algorithm lies in the use of a potential function that guides the search towards the goal. The A* algorithm determines the order in which vertices are expanded in a search by using a cost function, $f(v)$. This function is a sum of two other functions: the known distance from the starting vertex to the current vertex, $d(q,v)$, plus a heuristic function, $h(v)$, that estimates the distance from this vertex to the goal.

As A^* traverses the graph, it follows a path of the lowest expected total cost. It maintains a priority queue of nodes to be traversed and it expands first vertices that appear to be most likely to lead towards the goal. The lower $f(v)$ for a given node v, the higher its priority. At each step of the algorithm, the node with the lowest $f(v)$ value is removed from the queue. Then, the f and g values of its neighbors are updated accordingly, and these neighbors are added to the queue (CRUZ et al., 2012). The algorithm stops when the destination vertex is removed from the queue or when the queue is empty.

If the potential function h does not overestimate the cost to reach the goal from all $v \in V$, then A* always finds shortest paths. If h(v) is a good approximation of the cost to reach the goal, A* efficiently drives the search towards the goal, and it explores considerably fewer nodes than Dijkstra's algorithm. If $h(v) = 0 \; \forall v \in V$, A* behaves exactly like Dijkstra's algorithm, i.e., it explores the same vertices.

Unlike the solutions used in the calculation of shortest paths, in the proposed solutions in this thesis, the A* search is incorporated directly into the incremental expansion of the network, rather than being used to calculate the travel-time from the query vertex q to each candidate point of interest. Particularly, in the TD-kNN-OTC query, where we aim at finding the nearest POIs from q considering the operating time of the POIs, there are multiple and unknown goals. In the TD-OSR query, the goal is not only to reach the destination the quickest, but also to

pass through a number of POIs belonging to certain categories in a given order (COSTA et al., 2013).

2.4 ACCESS METHOD FOR TDGS

Building strategies and algorithms for correct and efficient query processing in time-dependent networks is a challenge, since the common properties of graphs can not be satisfied in the time-dependent case (GEORGE et al., 2007a). Particularly, these networks can not be stored in the same way than a static network, the same applies to the access to the network information. Thus, it emerges the need for storage methods that facilitate the access to the network information and that support the design of efficient algorithms for computing the frequent queries on such networks.

Some characteristics of the time-dependent networks should be considered in the development of this method. First of all, these networks require more space than the static ones to store the costs, since, for each edge, we need to keep the cost to traverse it for each time interval of constant size (COSTA et al., 2014). Another important observation is that the cost of storing the edges of the network grows as the time granularity (number of intervals) increases. Finally, to store the costs of traversing an edge for all the time intervals together implies accessing unnecessary information when retrieving disk page(s) that contains this edge, since the access to the adjacency list is executed to get the cost of the edges for a given time.

Based on these observations, in order to process our queries in a more efficient way, we resort to the access method proposed by (COSTA et al., 2014). It is important to notify that we use this method without any modification or extension. It is composed by three levels, the Time-Level, the Graph-Level and the Data-Level shown in Figure 9. The data pages in the time-level contain pointers to index structures in the graph-level. As a TDG can be seen as a set of static graphs for each time interval, the idea behind the first level is to access first the graph corresponding to a given time interval, avoiding retrieving the edge costs for every possible departure time. The graph-level has an index structure for each time partition, such that it is possible to, given a vertex identifier (Nid), access its adjacency list in the graph corresponding to the departure time. The data pages of the structures in the graph-level contain pointers to a disk page in the data-level that stores the adjacency list of a vertex.

The index structures for the time and graph levels are generic in the originally proposed method. In this work, we opted to use a B^+-tree in the two index levels, which is provided by the XXL library (BERCKEN et al., 2001). The pointers to the graph and data levels are stored in the leaves and each node (including the internal nodes) is a page in disk (CRUZ et al., 2012). Thus, the number of pages accessed for each retrieved data entry is of the order of $O(\log_b |TP| + \log_b |V|)$, where b is the order of the tree, TP is the number of temporal partitions that compose the cost of an edge and V is the number of vertices.

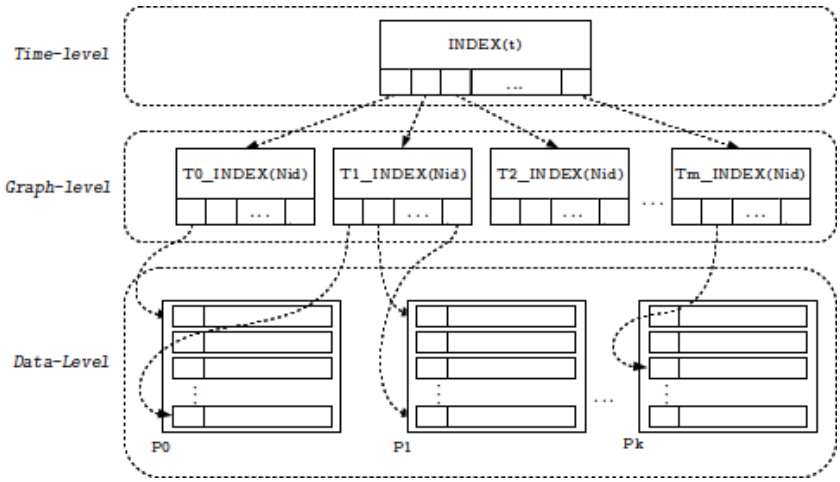


Figura 9 – An access method for time-dependent road networks (CRUZ et al., 2012).

2.5 TAXI BUSINESS

The ground taxicab service is a public transport mode widely used in big cities. Unlike mass public transit (such as buses, trolley buses, trams, subways and ferries), the cabs does not have a regular route and no pre-set schedule. Besides, taxicab is an individual transport service and has higher tariff than mass public. However, taxicab provides a higher quality service.

As part of the taxi companies' efforts on improving their quality of service, in general, each taxi is equipped with a smartphone, which is mainly used to handle taxi bookings and monitor a taxi's real time status. More specifically, it receives taxi booking tasks from the backend service (taxi call center), and sends back taxi driver's decision (accept or reject the task) via mobile internet service (3G). Moreover, each smartphone keeps logging and updating a taxi's real time state by collecting the information from its frontend touch screen. Figure 10 depicts an architecture generally used by cabs companies.

Based on the collected real time information, the smartphone app is able to identify different taxi states. Table 1 lists all the taxi states with their descriptions.

Taxi State	**Description**
FREE	Taxi unoccupied and ready for taking a new passenger or booking
POB	Passenger on board
STC	Taxi soon to clear the current job and ready for new bookings
ONCALL	Taxi unoccupied, but accepted a new booking
ARRIVED	Taxi arrived at the booking pickup location and waiting for the passenger
NOSHOW	No passenger showing up at he booking pickup location

Tabela 1 – Taxi State

The taxi state transitions depend on the type of a taxi job. In general, all taxi jobs can be classified into two categories: street job and booking job. A street job means a cab picks up new passengers by street hail, and the following is the typical taxi state transitions on a street

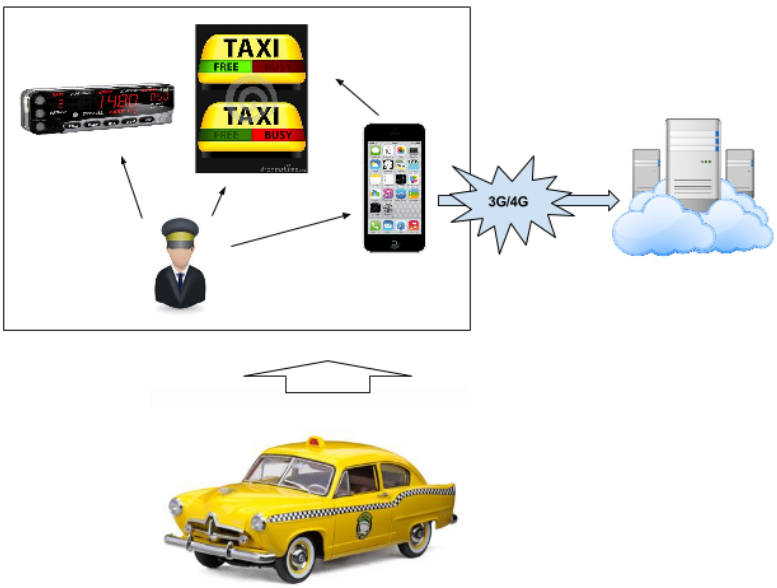


Figura 10 – Architecture generally used by cabs companies.

job:

1. A passenger down a cab with FREE state along a road or a taxi stand.
2. The taxi driver starts the taximeter for a new trip and reports this situation using the smartphone app, which updates the taxi state to POB.
3. During the trip, the taxi state keeps POB while the smartphone app periodically updates the taxi GPS location.
4. The taxi is approaching the destination and the driver presses the FREE button on the smartphone app to update the taxi state to FREE.

A booking job means a cab picks up new passengers, who have made a booking via mobile phone applications (apps), short message service (SMS) or telephone. The typical taxi state transitions on a booking job can be described as next:

1. A passenger makes a taxi booking, and the backend service dispatches the booking information to the nearby taxis with FREE state. For this purpose, the backend service makes a query looking for cabs inside a circle, with a certain radius, where the center of the circle is the passenger position. In general, the current applications use the smallest Euclidean distance. The cabs should be part of the zone which represents the customer's location. Zone is the area where a set of taxi drivers operates. The result of this query is stored in a queue ordered by the Euclidean distance from the taxi to the passenger in

ascending order. Then, the queue is consumed and the driver notified of the passenger's call. If the selected driver does not accept the passenger's request, the next one in the queue is notified.

2. A taxi driver successfully bids the booking job by pressing the button on the smartphone touch screen, and meanwhile the app updates the taxi state to ONCALL.
3. Upon arrival of the booking pickup location, the app updates the taxi state to ARRIVED.
4. If the passengers do not show up within a specific time period (e.g., 15 minutes), the app updates the taxi state to NOSHOW first and then to FREE within 10 seconds.
5. If the passenger gets on the taxi in time, the app updates the taxi state to POB once the driver starts the taximeter.
6. Tthe subsequent taxi state transitions are the same as street job's procedure, i.e., from street job's.

Figure 11 shows a complete taxi state transition diagram, which includes the procedures of both street jobs and booking jobs.

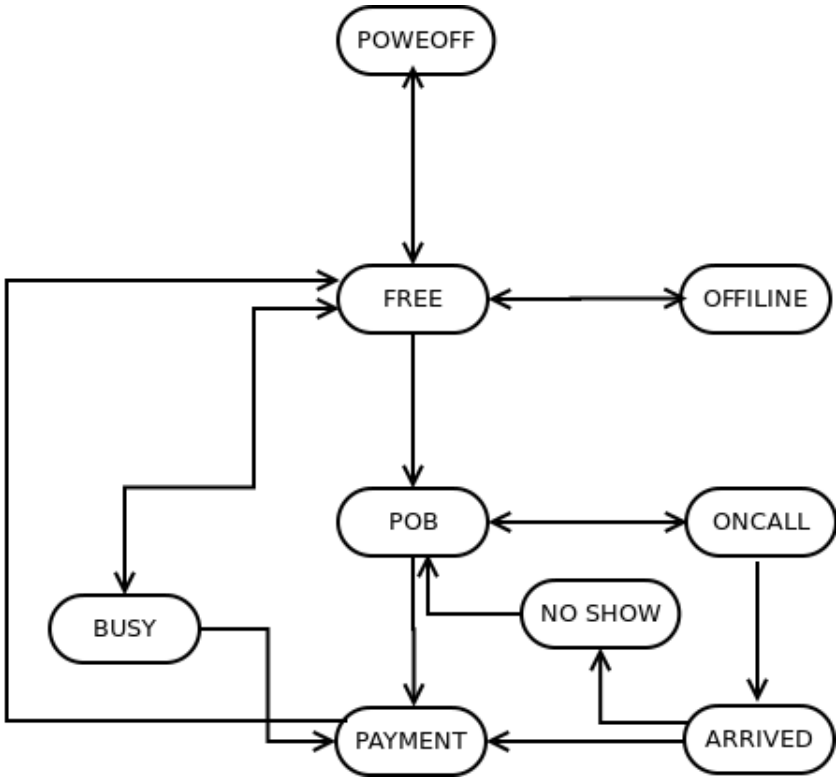


Figura 11 – Taxi call state machine.

The smartphone app keeps updating and tracking any changes of taxi state and other important information, e.g., GPS location, vehicle speed and taxi fares. The smartphone app writes all such information to its local storage, and meanwhile selectively and periodically sends them to the backend service via mobile internet. The app logging frequency is not fixed by default, and a logging action is triggered by the taxi state changes, GPS location updates and a few other critical vehicle events. Different from the traditional GPS localizer traces, the app

adopts the event-driven logging mechanism, which are explicitly driven by the taxi state transition events. Therefore, the app captures much more accurate information than the traditional GPS traces, and accordingly provides more opportunities to discover and understand activities of both taxis and passengers.

We use the app log from a large local taxi operator, and select some fields: cabid, latitude, longitude, altitude, accuracy, altitudeaccuracy and timestamp. Table 2 gives the selected fields and a sample record.

cabid	latitude	longitude	altitude	accuracy	altitudeaccuracy	timestamp
982335450	-37316237	-385125479	-6	30	98	Mon Nov 16 2015 11:01:43 GMT-0300 (BRT)
982335450	-37316237	-385125479	-6	30	98	Mon Nov 16 2015 11:01:43 GMT-0300 (BRT)
982335450	-37316654	-38512422	-5	29	100	Mon Nov 16 2015 11:01:44 GMT-0300 (BRT)

Tabela 2 – Selected Fields of Smartphone Log with a Sample (Fonte: (SIMPLES, 2015))

2.6 CONCLUSION

In this chapter we formally defined the concept of time-dependent graph and presented some definitions useful to formalize our problems in the following chapters. Furthermore, we discussed how the Dijkstrae, INE and A^* algorithms work. Besides, we showed some characteristics of the taxi Business.

3 NN-REVERSE-TD QUERY

In this chapter, we describe the problem of finding nearest neighbor on a time-dependent network, hereinafter called NN-Reverse-TD query), which aims to find the nearest neighbor POI *tx* of a query object *q* in a time-dependent network such that *tx* has the minimum time-dependent travel-time to *q*.

We assume that a street network contains a set of identical objects (that is, points of interest such as taxi, ambulance, car, etc). Our street network model is represented in a form of a time dependent graph 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 start this chapter stating NN-Reverse-TD Query problem. Next, in Section 3.1, we present a naive solution for this problem, which it will serve as our baseline in our experiments. In Section 3.2, we describe an effective solution for the NN-Reverse-TD query problem.

3.1 PROBLEM STATEMENT

In this section we define 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 3, 4 and 5 presented in Chapter 2.

Let $P = \{p1, ..., pN\}$ be the set of PoIs mapped on vertices of a TDN graph TDG = (V,E,C). Each PoI p is univocally mapped to a vertice of TDG. Let a travel time function $tt(s,t)$, which computes the travel time from source node s to target node t over TDG, where s,$t \ni V$. We formalize as N-Reverse-TD query problem as follows:

Definição 6 (NN-Reverse-TD Query Problem) *An instance of NN-Reverse Query problem is a tuple NNRQ = (TDG, POI, qp, qt, wt) where TDG is a TDG graph, POI is a set of point of interests where POI $\subseteq$ P, qp is a query point such that qp $\ni$ 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 NNRQ, the problem is to find a p $\ni$ P that satisfies the following conditions:*

1. $tt(p,qp) \leq wt$
2. $\forall\ pi \ni V \mid tt(p,qp) \leq tt(pi,qp)$ *and* $p \neq pi$

Condition 1 stablishes 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.

3.2 NAIVE SOLUTION (BASELINE)

This approach is basically an extension of the Dijkstra's algorithm (DIJKSTRA, 1959), which addresses the problem of finding the shortest path from an origin to a destination

node. Here, the Dijkstra's algorithm was applied in time-dependent networks.

Dijkstra's algorithm computes all the shortest path between the source node to every target nodes. Although this solution does not take into consideration the travel time of the each edge, in the expansion of Dijkstra's algorithm we performed the following criteria:

1. The target is a point of interest, a POI;
2. The non-negative weights is an heuristic function $H(.)$ adds to each edge an estimate of the cost to reach any another edge.

The idea is to reach POIs the quickest. Then, after they are reached, we calculate the travel time to *query object*, 2. This is our baseline. It is important to stress that our naive solution as well as Dijkstra's algorithm determine if any two 'shortest path' only one will be returned.

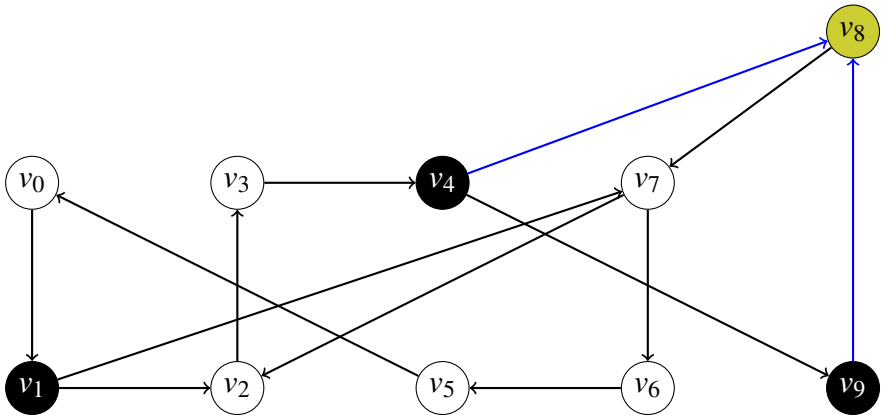
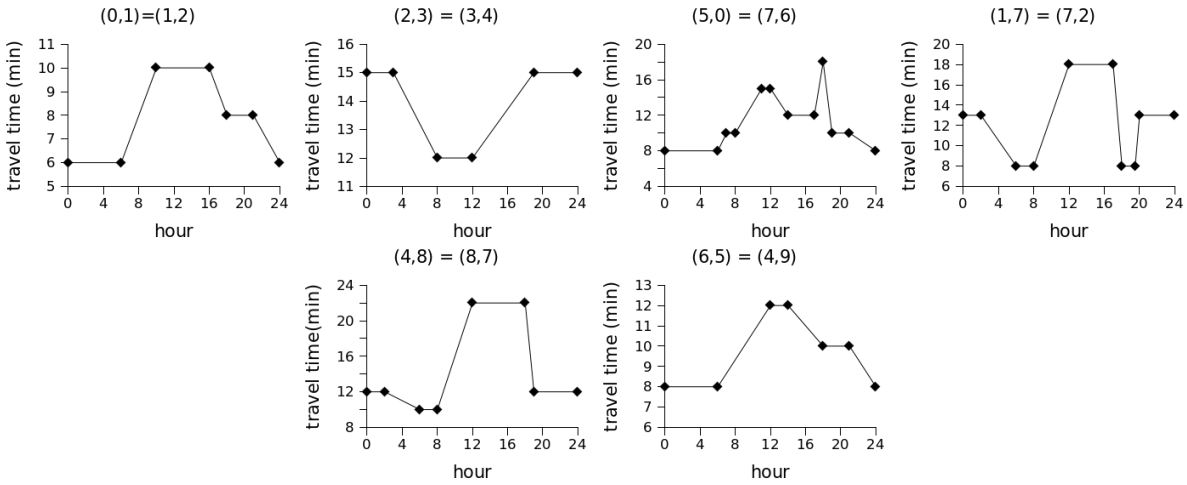


Figura 12 – Graph of the TDG



(a) Time-dependent edges cost.

Figura 13 – Time-dependent edges cost.

3.2.1 Off-line Pre-processing

The Naive 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, the chapter 4 is described more comprehensively this point.

Algoritmo 1: NN-Reverse-TD-Dijkstra (Baseline)

```
   input  : A query point q ∈ V, maximum travel time t ∈ [0, T] and day time dt
   output : The nearest neighbor to q considering the maximum travel time t
 1 begin
 2     poiIds[] ← G.getPoiIds();
 3     TT_min ← t;
 4     POI ← -1;
 5     VV ← ∅;
 6     for i = 0 poiIds.size() do
 7         Path ← Dijkstra(q, poiIds[i], dt);
 8         if Path.getCost() ≤ t then
 9             if Path.getCost() ≤ TT_min then
10                 POI ← poiIds[i];
11                 TT_min ← Path.getCost();
12                 VV ← Path.getVerticesVisited();
13             end
14         end
15     end
16     if POI > -1 then
17         NN(POI, TTmin, VV);
18     end
19 end
```

3.2.2 Query Processing

Algorithm 1 (NN-Reverse-TD-Dijkstra) presents the algorithm to solve the NN-TD Query problem. Using polymorphism,(CARDELLI; WEGNER, 1985), 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 the approach Backtracking Search. First, returns all $v \in V$ that representing a point of interest, POI, 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 $H(.)$ that the input value is a time of day and the function returns is time required to through the edge.

When there is a shortest path from point of interest to query point the next step is to carry out the following checks:

First

If the total time that was returned in the baseline query is less or equals the input value t, this is *maximum travel time* for moving from *point of interest* to *query point* (line 8);

Second

If the total time that was returned in the baseline query is less or equals a value already attributed (line 3) with the lowest value for moving from *point of interest* to *query point* (line 9);

3.3 NN-REVERSE-TD ALGORITHM

In this section we present the proposed algorithm, show how to perform the pre-processing step. This algorithm is based on the Incremental Network Expansion (INE) algorithm, initially proposed by (**??**). The INE is an algorithm based on Dijkstra's algorithm, where initiating a query point, 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 Naive solution does not take into consideration reverse graph. Its goal is to reach POIs the closer with 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 solution. Clearly, a heuristic that besides considering an estimate to reach POIs can better guide the search for POIs where one can be the service provider, it using the reverse graph heuristic.

Based on this, we propose reverse nearest neighbor solution. The reverse graph was motivated of the heuristic *If you can't find a solution, try assuming that you have a solution and seeing what you can derive from that ("working backward")* from George Pólya's 1945 book, How to Solve It (POLYA, 1945). This heuristic allows the search to be expanded capture the heuristic function $H(.)$ not from vertex to destination vertex, but the destination vertex to the source.

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

Definição 7 (Time-Dependent Graph Reverse (TDG-Reverse)) *A reverse graph of G is a graph 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)*. A Time-Dependent Graph Reverse (TDG-Reverse) is a graph defined as* G^R*.*

Figure 14a shows an example of a time-dependent graph G. Figure 14b illustrates the reverse graph G^R of G.

3.3.0.1 Offline Pre-processing

The pre-processing of this solution also has two steps. In the first one we created the graph G in memory with the values of the heuristic function. Differently from the previous

(a) G (b) G^R

Figura 14 – The Graph x The Reverse Graph

solution we carry out the process to reverse graph, G^R, where all directed edges are reversed.

3.3.0.2 Query Processing

As the two solutions differs in implementation, but can have the same input value: query point $q \in V$, maximum travel time $t \in [0, T]$ and 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.

First, the algorithm loads the reverse graph of G, G^R, from the disk, line 2. Then, the algorithm begins the expansion and inserts q in a priority queue Q (line 9) 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 11). When a vertex u is dequeued from Q it is marked as visited (line 12) 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.

For every element u pull of the Q check if the vertex was added to $Visited$ list (line 12), 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 13 and line 14. And if current vertex is an POI, line 16.

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 20) "jumps over"one iteration in the loop (line 21), doing this we avoid re-expand vertices unnecessarily. Otherwise, continue checking. TT is updated, added travel time value to reach the current neighbor, line 23. Again to verify that the maximum service time has been exceeded, it has been exceeded, "jumps over"one iteration, line 24. The $Parents$ list is updated with v, line 27. Already, as the network is expanded, ATv_i is decremented with t travel time value of the current neighbor (line 28).

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.

Algoritmo 2: NN-Reverso-TD

```
   input  : A query point q ∈ V, maximum travel time t ∈ [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 ← G.reverseGraph();
 3     AT_q ← t + dt;
 4     TT_max ← 0;
 5     Parent ← ∅;
 6     if q is POI then
 7         Return NN(q, 0, [q]);
 8     end
 9     En-queue (q, AT_q, TT_max) in Q;
10     while Q ≠ ∅ do
11         (q, AT_q, TT_max) ← De-queue Q;
12         Mark q as visited;
13         if TTcurrent > t then
14             Return -1;
15         end
16         if q is POI then
17             Return NN(POI, TTmin, Parent)
18         end
19         for u ∈ adjacency(q) do
20             if u is visited then
21                 continue;
22             end
23             TTcurrent ← q.getTravelTime() + u.getTravelTime();
24             if TTcurrent > t then
25                 continue;
26             end
27             Parent.add(q, u);
28             ATcurrent ← q.getArrivalTime()- u.getTravelTime();
29             En-queue (u, ATcurrent, TTcurrent) in Q;
30         end
31     end
32 end
```

4 EXPERIMENTAL EVALUATION

This chapter describes the experimental evaluation conducted in order to evaluate the proposed algorithm and is organized as follows. Section 4.1 describes how the experiments were conducted. Section 6.2 describes the experimental results obtained from a synthetic data set. Section 6.3 shows the results obtained from a real data set. Finally, Section 6.4 concludes the chapter.

4.1 EXPERIMENTAL SETUP

As far as we know there is no published research addressing the same query investigated in this thesis, which it is: find the k points of interest which can move to the query point in the minimum amount of time. Nonetheless, in order to have an idea of how effective our solution is, we compare it to a naive solution used as a baseline. The proposed baseline is a variation of the algorithm proposed by Dijstra(DIJKSTRA, 1959), as discussed previously. With the aim of validate the proposed approach, we have used two different scenarios. In the first scenario, a synthetic data set has been employed. In the second scenario, a real data set was utilized. Besides, both the proposed approach and the baseline were implemented in Java programming language and using the Graphast framework (Group Advanced Research in Database, 2015).

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 (see Table 3). We performed the experiments without caching.

Configuration	**Information**
Architecture	`x86_64` (32-bit, 64-bit)
CPU(s)	4
CPU family	6
Processor	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
MemTotal	8174664 kB (8GB)
Linux version	3.13.0-65-generic
Distributor ID	Ubuntu
Release	14.04 (trusty)

Tabela 3 – Server machine used in the experiments.

We have defined a synthetic cost function, denoted by $H(.)$, where for each edge, we chose a random speed between 3 km/h and 60 km/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 (See Table 4). The cost function $H(.)$, 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. Function $H(.)$ was inspired by the *Haversine* method (ABRAMOWITZ; STEGUN, 1964).

We can see according to Table 4 that higher speed values are arranged in hours where,

Interval of the Day	Range to generate speed
1 a.m to 6 a.m	30km/h - 50km/h
6 a.m to 7 a.m	20km/h - 30km/h
7 a.m to 9 a.m	3km/h - 14km/h
9 a.m to 11 a.m	20km/h - 30km/h
11 p.m to 14 p.m	3km/h - 14km/h
14 p.m to 16 p.m	50km/h - 60km/h
16 p.m to 17 p.m	20km/h - 30km/h
17 p.m to 20 p.m	3km/h - 14km/h
20 p.m to 22 p.m	21km/h - 30km/h
22 p.m to 00 p.m	50km/h - 60km/h

Tabela 4 – Interval of the day x Speed.

traditionally, there is a low flow of moving objects and the lower speed values are arranged in hours where, generally, there is a high flow of moving objects. Figure 16 shows a graphical representation of the behavior of the cost function $H(.)$. Figure 16 illustrates the distribution of the cost function $H(.)$.

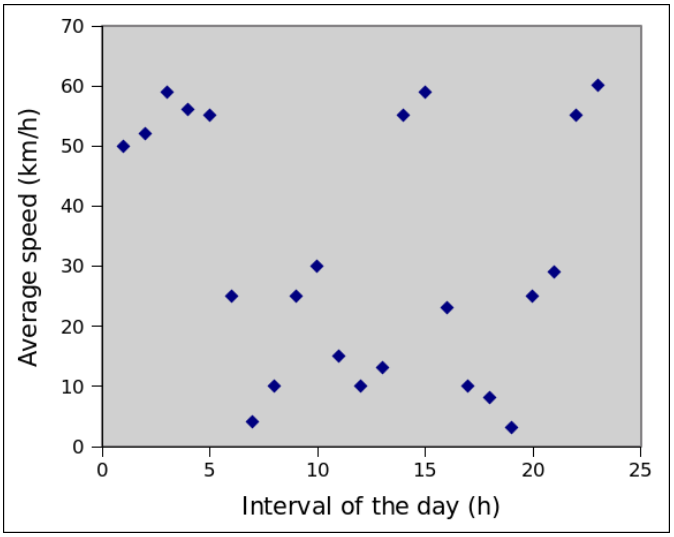


Figura 15 – A graphical representation of the behavior of the cost function $H(.)$.

4.2 SCENARIO 1: USING A SYNTHETIC DATA SET

In this section, we present the results of the experiments we carried out using a synthetic data set. We generated synthetic time-dependent road networks. A network is generated as a grid where each point corresponds to a vertex. Each vertex has a uniformly distributed number of neighbors from one to four, which are chosen between its adjacent vertices (including the diagonal ones). The POIs are uniformly distributed over the network. To generate the edges cost, we used the cost function $H(.)$, described previously.

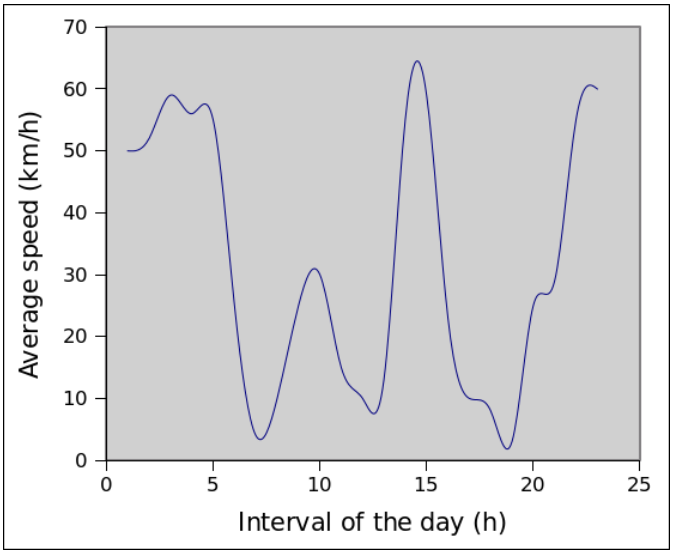


Figura 16 – The distribution of the cost function $H(.)$.

Figure 3 shows the algorithm used in this work to generate time-dependent road networks. This algorithm receives as input parameters: the number of points in the the x-axis (longitude), the number of points in the the y-axis (latitude) and the percentage of vertices that are points of interest. In lines 2 and 3 is calculated the distance between two vertices. Line 7 defines the geographic coordinate in the the x-axis. Line 9 defines the geographic coordinate in the y-axis. Hence, a vertex is created using these two geographical coordinates and added in the graph (lines 11 to 15). After the node insertion, its edges are created, firstly in the vertical direction (y-axis), line 18, and next in the horizontal direction (x-axis), line 19. It is noteworthy that the edges have created two paths and their costs are set according to the Table 4 described previously. It is important to note that this algorithm has a quadratic asymptotic complexity, that is, $O(n^2)$.

We evaluated how the proposed approach and the baseline work according to two variables that are shown in Table 5, the network size (i.e., number of vertices) and the density of POIs (i.e., the percentage of POIs in relation to the number of vertices). For each experiment, we vary a parameter and set the other parameter to default values (in bold) and for each triplet of different parameters, we generated 10 distinct time-dependent road networks and executed 10 queries randomly selected for each network, for a total of 100 queries. We calculated upper and lower 95% confidence limits for the relative gain, assuming the data to be normally distributed. As only the network size and the density of POIs affect the cost of the pre-processing of all solutions, we investigate how these costs are influenced by these variables.

We compare the algorithms according to the number of expanded vertices and the number of I/O operations to access the disk pages in data-level, both with respect to the density of POIs and network size. We simulated the I/O operations in data pages (data-level), using LRU as cache politics and 5% of the number of disk page in data-level as the cache size. We can verify, as expected, that there is a correlation between the number of expanded vertices and the number of accessed disk pages.

Algoritmo 3: Algorithm to generate time-dependent road networks

```
   input  : Number x of latitude points, number y of latitude points and p percentage of vertices that are POIs
   output : The Graph G with x × y vertices
 1 begin
 2     interadorX ← 180 ÷ x;
 3     interadorY ← 180 ÷ y;
 4     randomPOI[] ← RandomPOI(x, y, p);
 5     poi ← 0;
       // iterate to create nodes
 6     for i = 0 to x do
 7         longitude ← interadorX · i + (-90);
 8         for j = 0 to y do
 9             latitude ← interadorY · j + (-90);
10             node ← Node(latitude, longitude);
11             if poi ∈ randomPOI[] then
12                 node.isPoi ← true;
13             end
14             G.addNode(node) poi ← poi + 1;
15         end
16     end
       // iterate to create edges
17     for i = 0 to (x × y- 1) do
           // create parallels edge to the axis abscises
18         if i < (x × y- x) then
19             edge ← Edge(i, i + y);
20             edge.RandomCost();
21             G.addEdge(edge);
22             edge ← Edge(i + y, i);
23             edge.RandomCost();
24             G.addEdge(edge);
25         end
           // create parallels edge to the axis of the ordinates
26         if (i mod x) ≠ (x-1) then
27             edge ← Edge(i, i + 1);
28             edge.RandomCost();
29             G.addEdge(edge);
30             edge ← Edge(i + 1, i);
31             edge.RandomCost();
32             G.addEdge(edge);
33         end
34     end
35 end
```

Density of POIs	1%, 5%, 10%
Network Size	1k, 10k, 100k, 1000k

Tabela 5 – Parameters values of experiments using synthetic data.

4.2.1 Effect of the density of POIs

We set the density of POIs to be 1%, 5% and 10% of the number of points of interest (uniformly distributed). For each density, we generated 10 distinct time-dependent networks with 2000 vertices and executed 10 randomly selected queries with k=1 on each one.

Efficiency has been applied to measure the average time consumed to execute queries by using the proposed approach and the baseline solution. In the first experiment, performance was measured, i.e., the average time required to process a query. Figure 17 shows the performance of the baseline solution and the proposed approach using time-dependent road networks with

1k nodes. Note that the proposed approach outperforms the baseline solution in this scenario. For example, for the POI density of 10%, the proposed approach presented an average time of 368,143.58 *ns*, while the baseline solution produced an average time of 1,672,657,402.00 *ns*. Figure 18 shows the performance of the baseline solution and the proposed approach using time-dependent road networks with 10k nodes. Note that the proposed approach outperforms the baseline solution in this scenario. For example, for the POI density of 10%, the proposed approach presented an average time of 474.856,76 *ns*, while the baseline solution produced an average time of 166,660,430,461.00 *ns*. Besides, it is important to note that for the baseline solution the average response time increases as the POI density increases. This is reasonable since the number of possible solution grows when the number of POIs increases and the baseline computes all possible solutions.

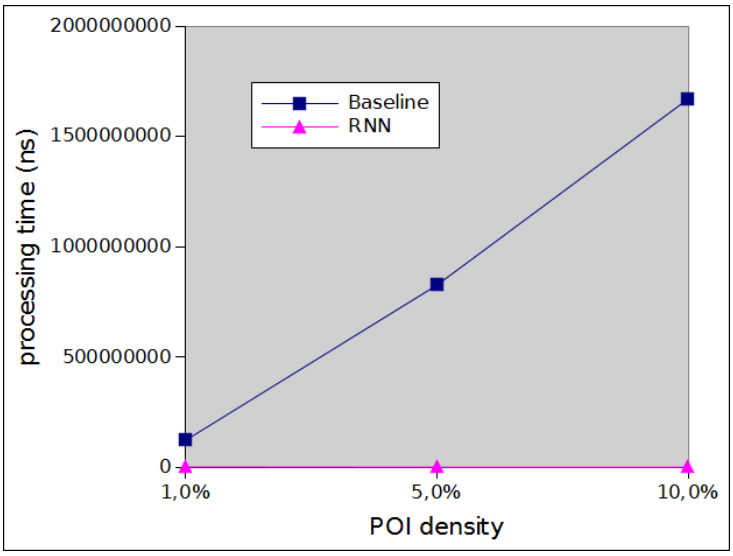


Figura 17

The effectiveness metric has been used to compare how many times the propose approach has found the optimal solution, that is, the same solution found by the baseline.

Figure 21 illustrates the effectiveness of the proposed approach in time-dependent road networks with 1k nodes. Note that, for a POI density of 5% the proposed approach found the optimal solution in 95% of cases. Figure 22 illustrates the effectiveness of the proposed approach in time-dependent road networks with 10k nodes. Note that, for a POI density of 5% the proposed approach found the optimal solution in 90% of cases. It is important to note that the proposed approach's effectiveness decreases as the POI density increases, regardless of the network size.

Figures 21 and 22 illustrate the difference between the average of visited vertices in both algorithms, using, respectively, time-dependent road networks with 1k and 10k nodes.

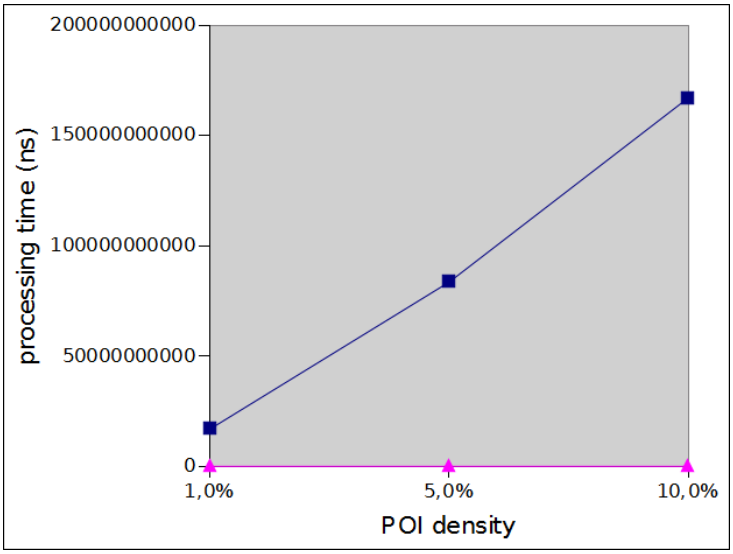


Figura 18 – Processing Time X POI Density in a $10k$ road network.

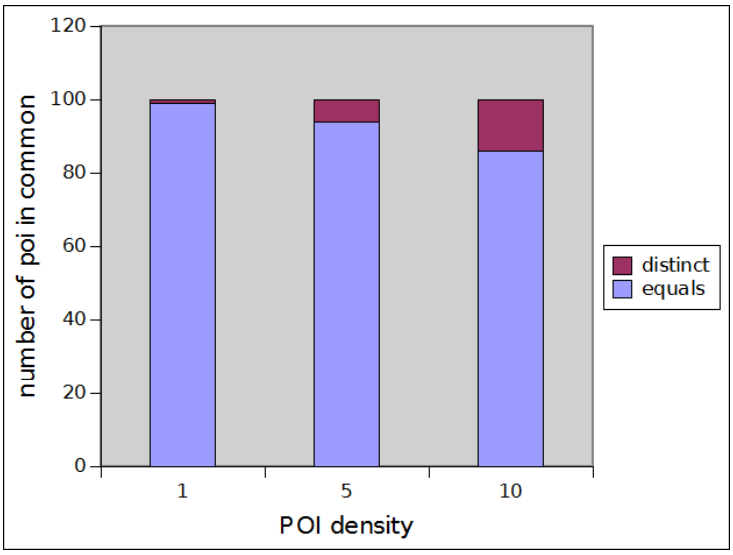


Figura 19 – Effectiveness in a $1k$ road network.

4.2.2 Effect of the network size

In this experiment, we generate 10 time-dependent networks with $1k$, $10k$, $100k$ and $1000k$ vertices. Each one with 10% of points of interest. We executed 10 randomly selected queries with $k = 1$ on each network.

4.3 SCENARIO 2: USING A REAL DATA SET

In this section, we present the results of the experiments we carried out using a real data set. In this scenario, 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) (WIKI, 2014),

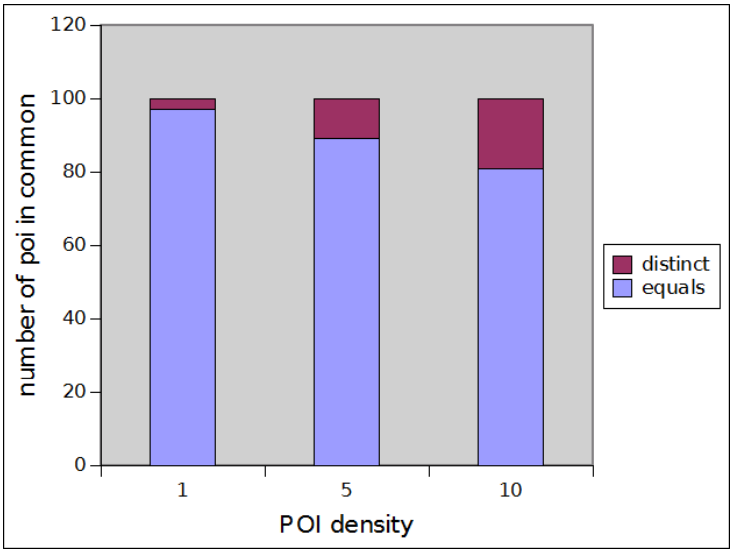


Figura 20 – Effectiveness in a 1k road network.

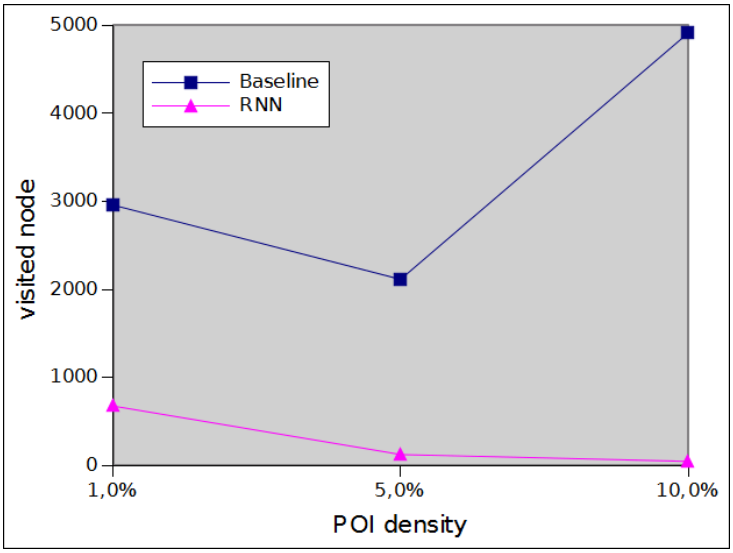


Figura 21 – The average of visited vertices in a 1k road network.

contains information (street segments, etc) about the road network of Fortaleza city 23 (See Figure 23). The second data set, available from a Brazilian cab fleet monitoring company called Taxi Simples (SIMPLES, 2015), 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).

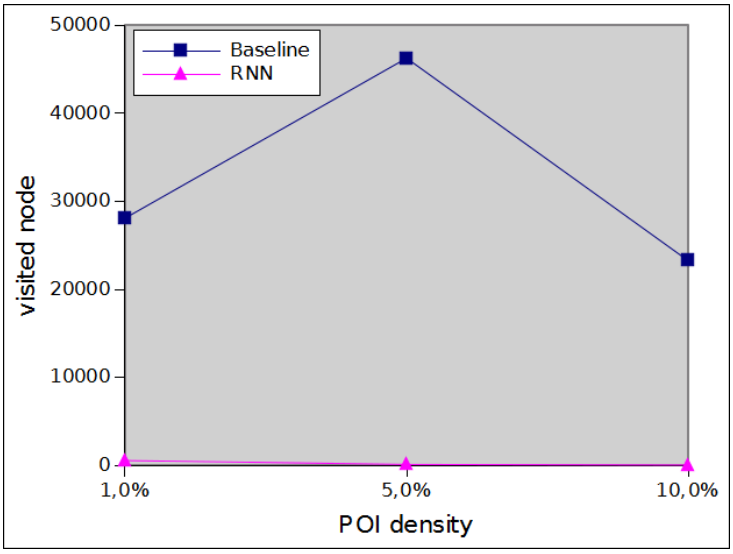


Figura 22 – The average of visited vertices in a 10k road network.

Figura 23 – Page about Fortaleza city in Open Street Map Wiki.

4.3.1 Data Sets Pre-processing

In order to build time-dependent road networks, we had to process and extract information from these two data sets. For this, we have used the following strategy:

1. First, we created a database, using *PostgreSQL 9.3.10* with the spatial database extension *Postgis* version 2.1, containing two tables (See Figure 24). The "ROADS" table 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, for each street corner. The "TAXI-CAB" table stores information about the taxi positions. This table has tree 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. For more information about the types of geographical data *Multilinestrings*, *Linestrings* and *Point*, you can visit the site of *Open Geospatial Consortium* (OPENGIS, 2015).

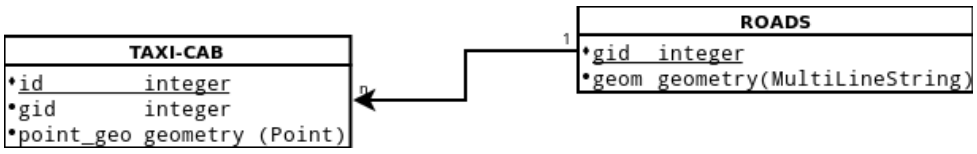


Figura 24 – Diagram of the database used to store the road network of Fortaleza and the taxi positions

2. Next, we exported the *Open Street Map* data set to the "ROAD" table in the *POSTGIS* database. This data set is available as a *OSM* file. Then, we used a script called "osm2pgsql" in order to export the information about the road network of Fortaleza to the "ROAD" table.
3. Then, we exported the *Taxi Simples* data set to the "TAXI-CAB" table in the *POSTGIS* database. This data set is available as a *Comma-separated values* (CSV) file, See Appendix A. In this file, the taxi positions follows the GEOLOCATION API specification (GEOLOCATION..., 2015) and have an average error of 13.60 meters. This file contains about 1,500,000 records (positions) for 430 cabs. So, there are many registers (points) to the same cab. For this reason, we selected for each cab the register with the highest timestamp. Thus, the "TAXI-CAB" table has only 430 registers, one for each cab. In this step, only the columns "*id*" and "point_geo" were filled.
4. Finally, 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. This task was carried out running a geographical query in *POSTGIS*, which performs a data interpolation. Figures 25a and 25b show the relationship between the taxi positions and the road network before and after the mapping match process. In this step, the column "*gid*" in the "TAXI-CAB" table is filled.

(a) Before the map matching process (b) After the map matching process

Figura 25 – The relationship between the taxi positions and the road network

Next, we have used the data exported to the *POSTGIS* database to build time-dependent road networks. For this, we have used Java programming language and the Graphast framework (Group Advanced Research in Database, 2015).

4.3.2 Effect of the density of POIs

4.3.3 Effect of the network size

In this experiment, we generate 100 time-dependent networks with $1k$, $10k$, $50k$ and $100k$ vertices. Each one with 1% of points of interest. We executed 10 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, using a strategy proposed by (KNUTH, 1969). The chosen snippets of the Fortaleza road network (with $1k$, $10k$, $50k$ and $100k$ vertices) are illustrated in Figures **??**.

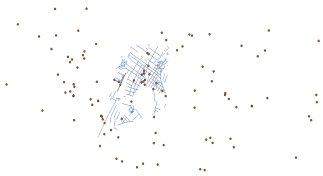


Figura 26 – Fortaleza road network with 1,000 points.

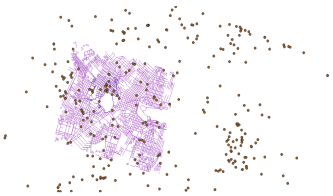


Figura 27 – Fortaleza road network with 10,000 points.

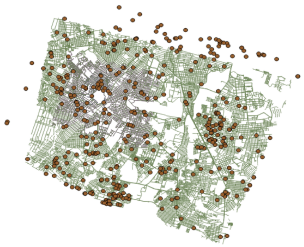


Figura 28 – Fortaleza road network with 50,000 points.

Figura 29 – Fortaleza road network with 100,000 points.

Figura 30 – Fortaleza road network with $1k$, $10k$, $50k$ and $100k$ vertices (Images obtained using QGIS (QGIS Development Team, 2009)).

It is important to note that two streets may have points in common (corners, for example). In the *Postgis* database, for a point that belongs to two streets there are two records. However, in the corresponding time-dependent graph, for a point that belongs to two streets there is just one node. For this reason, the number of nodes in a time-dependent graph and the number of corresponding records in the *Postgis* database are different. Figure 31 illustrates this difference.

Besides, the *Open Street Map* data set about the Fortaleza city is relatively recent, which afforded some troubles. The first identified problem was the road network disconnection, producing certain kinds of “islands”. 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

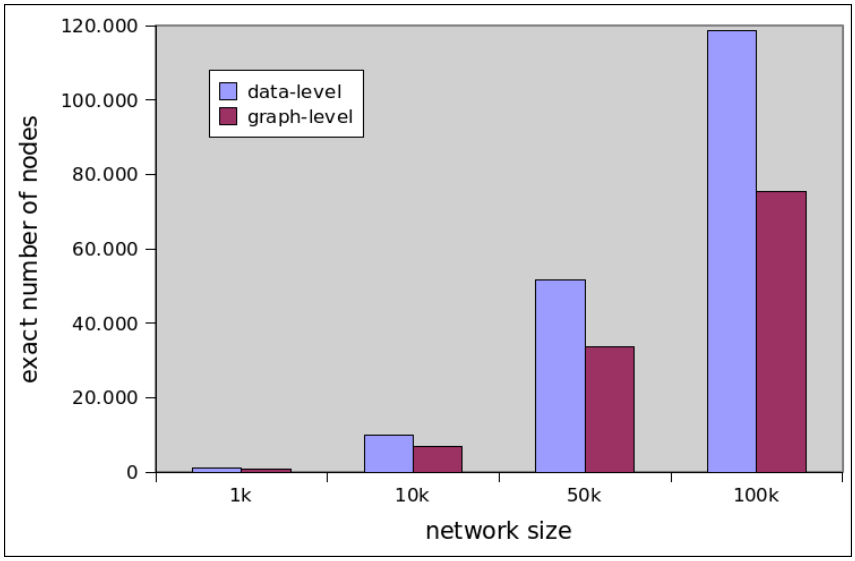


Figura 31 – Number of database registers X Number of nodes in the corresponding graph.

network provided by *Open Street Map* is disconnected, the average number of incident edges is very low (close to 1), as illustrated in Figure 32. Figure 33 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.

Figure 34, 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 proposed algorithm outperforms the baseline solution.

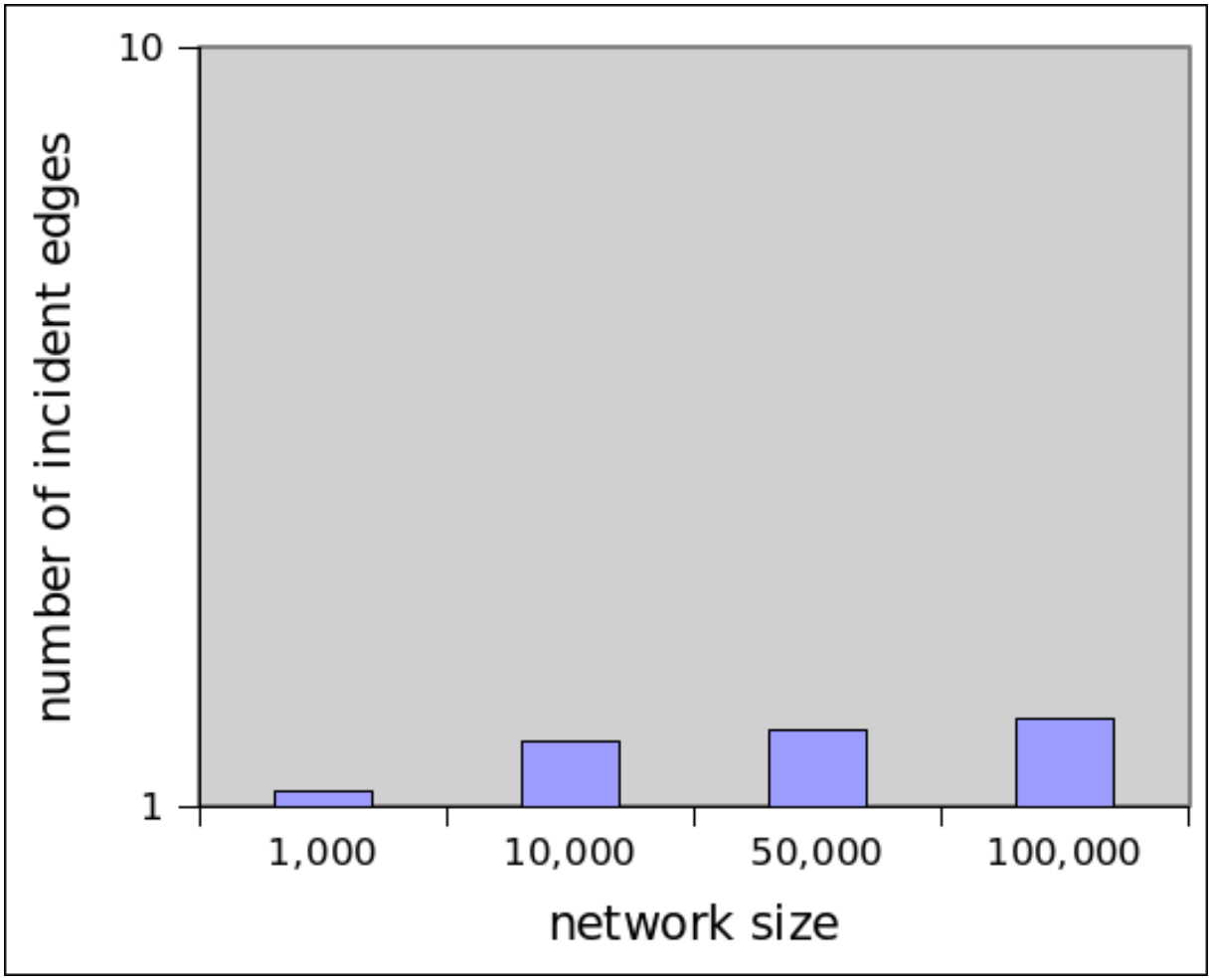


Figura 32 – Number of incident edges.

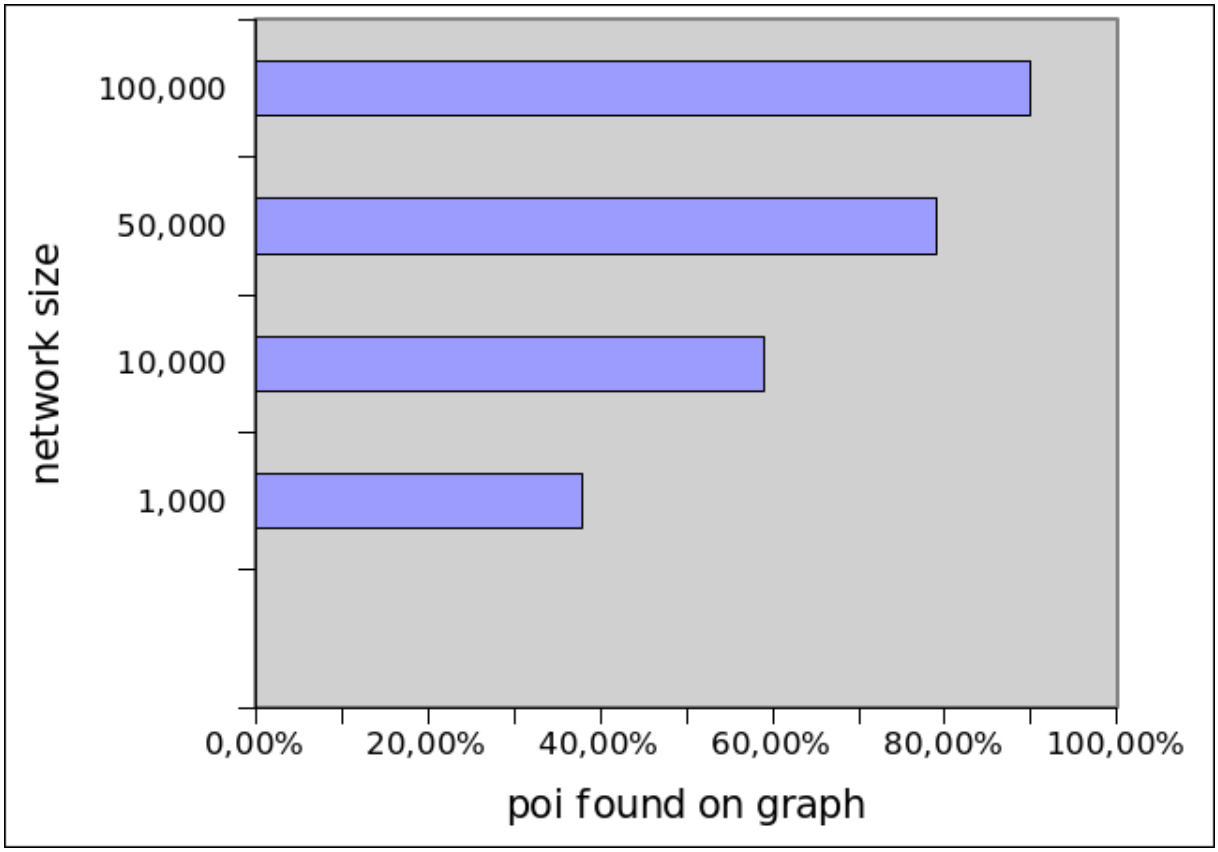
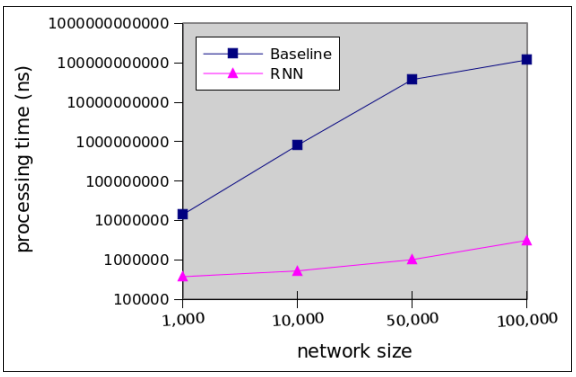
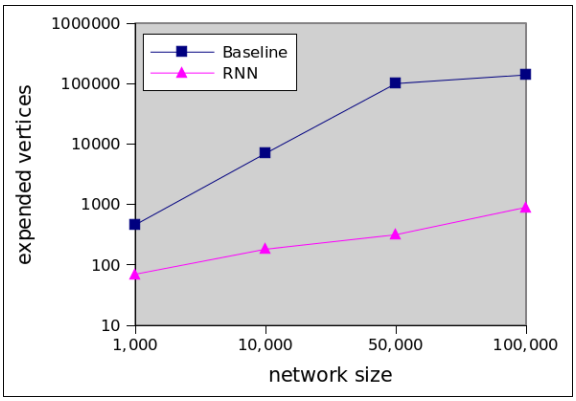


Figura 33 – Ration of executions that found a POI X Network size.



(a) Network size X Processing time.



(b) Network size X Number of expanded vertices.

Figura 34 – Evaluation of network size influence in real road networks.

5 RELATED WORK

In this chapter we review works that relate to our approach and discuss in detail how our solution advances the state of the art. Section 5.1 presents the approaches to find the shortest path in time-dependent networks. In Section 5.2 are discussed the solutions to execute *k*NN queries in road networks. Section 5.3 shows some initiatives to run *k*NN queries in time-dependent networks. Finally, Section 5.4 discuss two types of proximity queries called Reverse Nearest Neighbor (RNN) and Reverse Farthest Neighbor (RFN) queries.

5.1 TIME-DEPENDENT SHORTEST PATH

This problem has an intrinsically relation with the time-dependent *k*NN problem, since the nearest POIs from the query depend on the cost (travel time) of the shortest (fastest) path to each POI object. The usual solution to shortest path problem in static graphs is Dijkstra's algorithm (DIJKSTRA, 1959). In (WAGNER; WILLHALM, 2007), the authors present a survey with many others ideas that have been proposed to find point-to-point shortest paths.

Unfortunately, these ideas would fail when time-dependent networks are considered. Much less work has been proposed to the time-dependent case. The first algorithm that considers a time-dependent variant of shortest paths is addressed in (COOKE; HALSEY, 1966). This algorithm is a modified form of Bellman's (BELLMAN, 1957) iteration scheme for finding the shortest route between any two vertices in a network.

In 1997 and 1998, Chabini (CHABINI, 1997)(CHABINI, 1998) proposed two types of time-dependent shortest path (TDSP) algorithms in discrete dynamic networks. George and Shekhar proposed (GEORGE et al., 2007b) a time-aggregated graph where they aggregate the travel-times of each edge over the time instants into a time series. Their model has less storage requirements than the time-expanded networks. All these studies assume that the edge weight functions are defined over a finite discrete time window. They proposed two algorithms for shortest path computations. For each time window, the first algorithm computes the shortest path for a given start-time, through the static network. The second algorithm computes the shortest paths which result in the least travel time over the entire time period. Orda et al. (ORDA; ROM, 1990b) proposed a Bellman-Ford based solution where edge weights are piece-wise linear functions.

In (DEAN, 1999), Dean proposed a label-setting algorithm where arrival times are considered as the labels of the nodes. The *Time-Interval All Fastest Path (allFP) query* (KANOULAS et al., 2006) proposed solve. This algorithm was based on *A**. Instead of sorting the priority queue by scalar values, they maintain a priority queue of all paths to be expanded. Therefore, they enumerate all the paths from the source to a destination node which incurs exponential running time in the worst case. In addition, their algorithm is efficient when estimation (heuristic function in *A**) can enable the effective pruning of the search space. Another approach was proposed by (NANNICINI et al., 2008b), it applies bidirectional *A** search with landmarks on a time-dependent network to calculate the fastest path between two vertices.

5.2 *k*NN QUERIES IN ROAD NETWORKS

The problem of *k*NN queries in road networks was introduced in (PAPADIAS et al., 2003). In that paper, the authors present two different solutions to this problem: the Incremental Euclidean Restriction (IER) and the Incremental Network Expansion (INE) algorithms. IER uses the assumption that the Euclidean distance between two points on the network is less than the network distance. This assumption is used to retrieve *k*NN points, according to the Euclidean distance, and use their network distance as an upper bound for the distance of *k*NN points according to the network distance. INE is an adaptation of Dijkstra's algorithm. With this algorithm, starting from the query object q all network nodes reachable from q in every direction are visited in order of their proximity to q until all k nearest data objects are located.

In (KOLAHDOUZAN; SHAHABI, 2004a) the authors presented an approach based on pre-computing the network's voronoi polygons (NVP) (ERWIG; HAGEN, 2000), indexed by a spatial access method. Using NVPs one can immediately find the first nearest neighbor of a query object and reduce the on-line cost in a *k*NN search. These approaches cannot be directly applied to solve TD-*k*NN queries. Berchtold et al. (BERCHTOLD et al., 1998) too used *Voronoi diagram*, suggest precalculating approximating and indexing the solution space for the nearest neighbor problem in m dimensional spaces. Precalculating the solution space means determining the *Voronoi diagram* of the data points, cells.

In (HU et al., 2006) the authors developed a network reduction technique where the network topology is simplified by a set of interconnected tree-based structures (SPIE's). By doing that the number of edges is reduced while all network distances are preserved. They proposed the *nd* (nearest descendant) index on the SPIE such that a *k*NN query on those structures follows a predetermined tree path, avoiding unnecessary network expansion. In (LEE et al., 2009) the authors exploited the search space pruning technique. With the observation that during a search some subspaces of the network with no objects can be skipped, they organized a road network as a hierarchy of interconnected regional sub-networks (Rnets). They speed-up the search performance by incorporating shortcuts that avoid detailed traversal and object lookup within Rnets, allowing bypass those Rnets that do not contain objects of interest.

Kollios et al. (KOLLIOS et al., 1999) propose an elegant solution for answering *k*NN queries for moving objects in one dimensional space. Their algorithm uses a duality transformation, where the future trajectory of a moving point $x(t) = x_0 + v_x t$ is transformed into a point (x_0, v_x) in a so-called dual space.

Raptopoulou et al. (RAPTOPOULOU et al., 2003) and Tao et al. (TAO et al., 2004) consider the nearest neighbor problem for a query point moving on a line segment, for static or moving interest points. Raptopoulou et al. (RAPTOPOULOU et al., 2003) consider simplified and less effective heuristics for directing and pruning the search in the *TPR-tree*. Tao et al. (TAO et al., 2004) also consider the general concept of so-called time-parameterized queries.

An efficient and scalable approach to *k*NN queries in a road network was proposed in (CHO; CHUNG, 2005). This paper presented a system called UNICONS (a *uni*que *con*tinuos *s*earch algorithm) where the main idea is to integrate the precomputed *k*NNs into the Dijkstra'

algorithm.

5.3 TIME-DEPENDENT *k*NN QUERIES

The problem of *k*NN queries in time-dependent networks was introduced by (DEMIRYUREK et al., 2010b). In this paper, the authors compare two different baseline methods to solve this problem. The first approach uses time-expanded graphs to model the network. Time-expanded graphs allows us to exploit previous solutions in static networks to solve TD-*k*NN queries. However, this solution has numerous shortcomings, such as high storage overhead, slower response time and also incorrect results, as shown by (DEMIRYUREK et al., 2010b). The second approach is an adaptation of the INE algorithm (PAPADIAS et al., 2003) that does a blind search while expanding the network.

In (DEMIRYUREK et al., 2010a), a pre-computation process that builds two different indexing structures, the Tight Network Index (TNI) and Loose Network Index (LNI) was proposed. Both are composed for cells that reference the points of interest such that, if a query point q is in a tight cell of a point p, p is its nearest neighbor, and if q is out of a loose cell of p, p is not its nearest neighbor. As in the NVP method, using TNI one can immediately find the first nearest neighbor of a query object. However, it is not clear how this process works well when travel time functions in edges with opposite directions can be different. That is an important aspect in time-dependent networks, since the cost of a path can be determined by its orientation.

Goal-directed search, also called *A** (HART et al., 1968) was adapted to time-dependent scenarios by Flinsenberg (FLINSENBERG, 2004). Another version of *A**, more efficient for time-dependent scenarios, called *A*LT, was presented in (GOLDBERG; HARRELSON, 2005) and (GOLDBERG; WERNECK, 2005). Delling et al. (DELLING; WAGNER, 2007) presented an unidirectional *A*LT that was evaluated in time-dependent graphs (fulfilling the FIFO property) speed-ups the previous approach in a factor between 3 and 5 times, depending on the degree of time-dependency. Goal-directed search has also been successfully applied to time-dependent networks in (PYRGA et al., 2004) (PYRGA et al., 2007) and (DISSER et al., 2008). The time-dependent *A*LT approach can be used in a bidirectional manner. It was shown in (NANNICINI et al., 2008a). The key idea of bidirectional search in time-dependent networks is that the backward search is only used to bound the nodes that the forward search has to visit. This approach can be further accelerated by limiting *A*LT the search to a core set of nodes, extracted during the preprocessing step (DELLING; NANNICINI, 2008).

An improved solution is proposed by (CRUZ et al., 2012). In this work, the authors proposed an algorithm that is based on INE expansion (PAPADIAS et al., 2003) and use an *A** search to guide this expansion. The vertices that offer a great chance to be in a fast path to a POI are expanded first. A pruning process is also proposed in order to avoid expanding unpromising vertices, i.e, vertices that are far to any POI. To do that, was proposed an heuristic function that works well in time-dependent networks. The idea behind it approach is to discard vertices that are near from the query but far from any point of interest. It approach adds to each vertex an expectation to find a point of interest quickly in a path that pass by this vertex. Furthermore, it

was used upper bound values of travel time to prune unpromising paths. It experiments show that TD-NE-*A** require up 50% less I/O operations than TD-NE. Was also propose a generic and scalable access method to support it approach on a disk-based environment.

In (COSTA et al., 2013), the authors 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 and the service time of the facilities are taken into consideration. More specifically, the proposed query aims to find the point of interest (e.g., a museum) in which the user can be served in the minimum amount of time, accounting for both the travel time to the point of interest and the waiting time, if it is closed.

5.4 REVERSE NEAREST/FARTHEST NEIGHBOR

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 (TRAN et al., 2009).

A Reverse Nearest Neighbor (RNN) query retrieves interest points which consider the query point as their nearest neighbor. A RNN query is used to find places where the query point has most impact. An example of RNN query is to find residential that consider a restaurant as the nearest restaurant (TRAN et al., 2009). Therefore, RNN search is used to find places which are most affected by a given location.

In contrast, a Reverse Farthest Neighbor (RFN) query retrieves interest points which consider the query point as their farthest neighbor. This query type is used to find places that are least affected by a given location. For example, a estate company may want to know which properties is least affected by a road construction.

A Reverse k Nearest Neighbor (RkNN) query is a generalization of basic RNN where the interest points retrieved consider the query point as one of their nearest neighbors ($k > 1$) rather the only nearest neighbor. As an example of RkNN query is retrieves all restaurants which assign the query restaurant as one of the two ($k = 2$) nearest neighbors.

A Reverse k Farthest Neighbor (RkFN) query is a generalization of basic RFN where the query point is considered as one of the farthest neighbors ($k > 1$) by the interest points retrieved. As an example of RkFN query is, given a train station A as the query point, which other train stations consider A as one of the two ($k = 2$) farthest train stations.

A discussion about RNN queries and its variants is found in (KORN; MUTHUKRISHNAN, 2000). This paper proposes an approach for RNN queries and a method for large data sets using R-tree. Yang et al (YANG; LIN, 2001) introduced a single index structure called 'Rdnn-tree' to replace the multiple indexes used in their previous work. This index structure can be used for both RNN and NN queries.

A new concept, called 'Reverse Skyline Queries', was introduced in (DELLIS; SEEGER, 2007) by Dellis and Seeger in order to find RNN of any given query point. It uses the 'Branch and Bound' technique to retrieve a candidate point set and refines this set to find the

exact answers. In (KUMAR et al., 2008), the authors introduced a method to process reverse proximity queries in two and three dimensions using a lifting transformation technique. Though, the results were only approximate. Although these methods are helpful in geographical and spatial works, they cannot be used for spatial network problems.

In (SAFAR et al., 2009), the authors discussed various types of RNN queries and proposes several algorithms to process these queries in spatial network databases. These algorithms were based on Network Voronoi Diagram, PINE algorithm and the pre-computation of network distances. 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 RkNN, where k can be any number given at the query time. Finaly, (TRAN et al., 2009) introduced other types of reverse proximity queries, called RFN (Reverse Farthest Neighbor) and RkFN.

6 CONCLUSION AND FUTURE WORK

This chapter is organized as follows. The main contributions and results achieved with this research are presented in Section 6.1. Finally, Section 6.2 we present direction for future works.

6.1 CONCLUSION

In this thesis, we identify a variation of TD-kNN queries in time-dependent road networks that has wide applications and requires novel algorithms for processing. Differently from TD-kNN queries, we aim at minimizing the travel time from points of interest to the query point. With this approach, 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.

Previous works have proposed solutions to answer TD-kNN queries considering the time dependency of the network but not computing the proximity from the points of interest to the query point. We proposed and discussed a solution to this type of query which are 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 can note that in some cases, the answer returned by a regular TD-kNN query is coincidentally the same as returned by our query, for example, where all the roads allow bi-directional vehicles‘s flow and all conversions are allowed, what is unreasonable in real situations.

The general objective of this work was to study the problem of to find the k points of interest (e.g. cabs, ambulances or police cars) which can move to the query point (e.g. a taxi user or a citizen waiting for an emergency service) in the minimum amount of time, proposing efficient and optimal solutions to them.

In this context, we proposed two solutions to the problem of minimizing the travel time from points of interest to the query point, that is, to find the k points of interest which can move to the query point in the minimum amount of time. More specifically:

- We proposed and discussed a baseline solution, for comparison purposes, to the problem of minimizing the travel time from points of interest to the query point in time-dependent networks;
- We proposed an efficient algorithm to solve the problem of minimizing the travel time from points of interest to the query point in time-dependent networks;
- We evaluated the proposed solutions to the problem of minimizing the travel time from points of interest to the query point in time-dependent networks using synthetic and real data;
- We specified and developed a new generator of synthetic networks, varying both the network size and the density of points of interest;

- We designed and developep road networks based in real data about the streets and cabs positions in the Fortaleza city;

6.2 FUTURE WORK

As a future work we intend to modify the time-dependent query proposed in this thesis adding some filters based on POIs' characteristics, such as: cabs with six (6) luggage or with accessibility aids. Besides, we pretend to compute the cost function using an estatistical distribution closer to reality. Moreover, we will investigate the possibility of update the cost function at real time, in order to represent some events like car crashes and road construction. The experimental evaluation could include other baselines. Another possible future work is to create time-dependent road networks using trajectories of real moving objects. Then, we would like to perform the experiments using such networks. Furthermore, it would be interesting to implement the proposed approaches in a database, instead of using a file system. Finally, we would also like to propose solutions to solve others popular spatial queries but in the context of time-dependent networks.

REFERÊNCIAS

ABRAMOWITZ, M.; STEGUN, I. A. **Handbook of mathematical functions: with formulas, graphs, and mathematical tables**. [S.l.]: Courier Corporation, 1964. 78 p.

BELLMAN, R. **Dynamic Programming**. Princeton, NJ: Princeton University Press, 1957.

BERCHTOLD, S.; ERTL, B.; KEIM, D. A.; KRIEGEL, H.; SEIDL, T. Fast nearest neighbor search in high-dimensional space. In: **Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998**. [s.n.], 1998. p. 209–218. Disponível em: <http://dx.doi.org/10.1109/ICDE.1998.655779>.

BERCKEN, J. V. den et al. Xxl - a library approach to supporting efficient implementations of advanced database queries. In: **Proc. of the 27th VLDB Conf.** [S.l.: s.n.], 2001. p. 39–48.

CARDELLI, L.; WEGNER, P. On understanding types, data abstraction, and polymorphism. **ACM Comput. Surv.**, v. 17, n. 4, p. 471–522, 1985. Disponível em: <http://doi.acm.org/10.1145/6041.6042>.

CHABINI, I. A new shortest paths algorithm for discrete dynamic networks. In: **8th IFAC Symposium on Transportation Systems: Preprints : Chania, Greece, 16-18.06.1997**. [s.n.], 1997. p. 551–556. Disponível em: <https://books.google.com.br/books?id=JMMwrgEACAAJ>.

CHABINI, I. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. **Transportation Research Record: Journal of the Transportation Research Board**, v. 1645, p. 170–175, 1998. Disponível em: <http://dx.doi.org/10.3141/1645-21>.

CHEN, H.; KU, W.-S.; SUN, M.-T.; ZIMMERMANN, R. The partial sequenced route query with traveling rules in road networks. **GeoInformatica**, p. 541–569, 2011.

CHO, H.; CHUNG, C. An efficient and scalable approach to CNN queries in a road network. In: **Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005**. [s.n.], 2005. p. 865–876. Disponível em: <http://www.vldb2005.org/program/paper/fri/p865-cho.pdf>.

COOKE, K. L.; HALSEY, E. The shortest route through a network with time-dependent internodal transit times. **Journal of Mathematical Analysis and Applications**, v. 14, p. 492–498, 1966.

COSTA, C. F.; NASCIMENTO, M. A.; MACÊDO, J. A. F. de; MACHADO, J. C. 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**. [s.n.], 2013. p. 22–29. Disponível em: <http://doi.acm.org/10.1145/2534190.2534194>.

COSTA, C. F.; NASCIMENTO, M. A.; MACÊDO, J. A. F. de; MACHADO, J. C. A*-based solutions for KNN queries with operating time constraints in time-dependent road networks. In: **IEEE 15th International Conference on Mobile Data Management, MDM 2014, Brisbane, Australia, July 14-18, 2014 - Volume 1**. [s.n.], 2014. p. 23–32. Disponível em: <http://dx.doi.org/10.1109/MDM.2014.9>.

CRUZ, L. A.; NASCIMENTO, M. A.; MACÊDO, J. A. F. de. k-nearest neighbors queries in time-dependent road networks. **JIDM**, v. 3, n. 3, p. 211–226, 2012. Disponível em: <http://seer.lcc.ufmg.br/index.php/jidm/article/view/187>.

DEAN, B. C. Algorithms for minimum cost paths in time-dependent networks. **Networks**, 1999.

DELLING, D.; NANNICINI, G. Bidirectional core-based routing in dynamic time-dependent road networks. In: **Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings**. [s.n.], 2008. p. 812–823. Disponível em: <http://dx.doi.org/10.1007/978-3-540-92182-0_71>.

DELLING, D.; WAGNER, D. Landmark-based routing in dynamic graphs. In: **Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings**. [s.n.], 2007. p. 52–65. Disponível em: <http://dx.doi.org/10.1007/978-3-540-72845-0_5>.

DELLIS, E.; SEEGER, B. Efficient computation of reverse skyline queries. In: **Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007**. [s.n.], 2007. p. 291–302. Disponível em: <http://www.vldb.org/conf/2007/papers/research/p291-dellis.pdf>.

DEMIRYUREK, U.; BANAEI-KASHANI, F.; SHAHABI, C. Efficient k-nearest neighbor search in time-dependent spatial networks. In: **Proc. of the 21st DEXA Conf.** [S.l.: s.n.], 2010. p. 432–449.

DEMIRYUREK, U.; BANAEI-KASHANI, F.; SHAHABI, C. Towards k-nearest neighbor search in time-dependent spatial network databases. In: **Proc. of the 6th DNIS Workshop**. [S.l.: s.n.], 2010. p. 296–310.

DEMIRYUREK, U.; KASHANI, F. B.; SHAHABI, C. 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**. [s.n.], 2010. p. 432–449. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15364-8_36>.

DEMIRYUREK, U.; KASHANI, F. B.; SHAHABI, C. Towards k-nearest neighbor search in time-dependent spatial network databases. In: **Databases in Networked Information Systems, 6th International Workshop, DNIS 2010, Aizu-Wakamatsu, Japan, March 29-31, 2010. Proceedings**. [s.n.], 2010. p. 296–310. Disponível em: <http://dx.doi.org/10.1007/978-3-642-12038-1_20>.

DIJKSTRA, E. W. **Communication with an Automatic Computer**. Tese (Doutorado) — University of Amsterdam, 1959. Disponível em: <http://www.cs.utexas.edu/users/EWD/PhDthesis/PhDthesis.PDF>.

DISSER, Y.; MÜLLER-HANNEMANN, M.; SCHNEE, M. Multi-criteria shortest paths in time-dependent train networks. In: **Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings**. [s.n.], 2008. p. 347–361. Disponível em: <http://dx.doi.org/10.1007/978-3-540-68552-4_26>.

ERWIG, M.; HAGEN, F. The graph voronoi diagram with applications. **Networks**, v. 36, p. 156–163, 2000.

FLINSENBERG, I. C. **Route Planning Algorithms for Car Navigation**. Tese (Doutorado) — Technische Universiteit Eindhoven, 2004. Disponível em: <https://pure.tue.nl/ws/files/2427738/200412420.pdf>.

GEOLOCATION API Specification. 2015. Http://dev.w3.org/geo/api/spec-source.html. Accessed: 2015-12-13.

GEORGE, B.; KIM, S.; SHEKHAR, S. Spatio-temporal network databases and routing algorithms: A summary of results. In: PAPADIAS, D.; ZHANG, D.; KOLLIOS, G. (Ed.). **Advances in Spatial and Temporal Databases**. [S.l.: s.n.], 2007, (Lecture Notes in Computer Science). p. 460–477.

GEORGE, B.; KIM, S.; SHEKHAR, S. Spatio-temporal network databases and routing algorithms: A summary of results. In: **Advances in Spatial and Temporal Databases, 10th International Symposium, SSTD 2007, Boston, MA, USA, July 16-18, 2007, Proceedings**. [s.n.], 2007. p. 460–477. Disponível em: <http://dx.doi.org/10.1007/978-3-540-73540-3_26>.

GOLDBERG, A. V.; HARRELSON, C. 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**. [s.n.], 2005. p. 156–165. Disponível em: <http://dl.acm.org/citation.cfm?id=1070432.1070455>.

GOLDBERG, A. V.; WERNECK, R. F. F. Computing point-to-point shortest paths from external memory. In: **Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005**. [s.n.], 2005. p. 26–40. Disponível em: <http://www.siam.org/meetings/alenex05/papers/03agoldberg.pdf>.

Group Advanced Research in Database. **Graphast: A framework tool that allows users to compose a number of user-generated/populated network models**. 2015. Disponível em: <http://www.graphast.org/>.

GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. **SIGMOD Rec.**, p. 47–57, 1984.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Trans. Systems Science and Cybernetics**, v. 4, n. 2, p. 100–107, 1968. Disponível em: <http://dx.doi.org/10.1109/TSSC.1968.300136>.

HTOO, H. **Query Algorithms for Location Based Services in Road Network Distance**. Tese (Doutorado) — Saitama University, 2013.

HU, H.; LEE, D. L.; XU, J. Fast nearest neighbor search on road networks. In: **In Proc. of the 10th EDBT Conf.** [S.l.: s.n.], 2006. p. 186–203.

JENSEN, C. S.; , J. ; PEDERSEN, T. B.; TIMKO, I. Nearest neighbor queries in road networks. In: **Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems**. [S.l.: s.n.], 2003. p. 1–8.

KANOULAS, E.; DU, Y.; XIA, T.; ZHANG, D. Finding fastest paths on A road network with speed patterns. In: **Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA**. [s.n.], 2006. p. 10. Disponível em: <http://dx.doi.org/10.1109/ICDE.2006.71>.

KNUTH, D. E. **The Art of Computer Programming, Volume II: Seminumerical Algorithms**. [S.l.]: Addison-Wesley, 1969.

KOLAHDOUZAN, M.; SHAHABI, C. Voronoi-based k nearest neighbor search for spatial network databases. In: **Proc. of the 13th VLDB Conf.** [S.l.: s.n.], 2004. p. 840–851.

KOLAHDOUZAN, M. R.; SHAHABI, C. Voronoi-based K nearest neighbor search for spatial network databases. In: **(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004**. [s.n.], 2004. p. 840–851. Disponível em: <http://www.vldb.org/conf/2004/RS21P6.PDF>.

KOLAHDOUZAN, M. R.; SHAHABI, C. Alternative solutions for continuous K nearest neighbor queries in spatial network databases. **GeoInformatica**, v. 9, n. 4, p. 321–341, 2005. Disponível em: <http://dx.doi.org/10.1007/s10707-005-4575-8>.

KOLLIOS, G.; GUNOPULOS, D.; TSOTRAS, V. J. Nearest neighbor queries in a mobile environment. In: **Spatio-Temporal Database Management, International Workshop STDBM'99, Edinburgh, Scotland, September 10-11, 1999, Proceedings**. [s.n.], 1999. p. 119–134. Disponível em: <http://dx.doi.org/10.1007/3-540-48344-6_7>.

KORN, F.; MUTHUKRISHNAN, S. Influence sets based on reverse nearest neighbor queries. In: **Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.** [s.n.], 2000. p. 201–212. Disponível em: <http://doi.acm.org/10.1145/342009.335415>.

KUMAR, Y.; JANARDAN, R.; GUPTA, P. Efficient algorithms for reverse proximity query problems. In: **Proceedings of the 16th ACM SIGSPATIAL International Conference on**

Advances in Geographic Information Systems. New York, NY, USA: ACM, 2008. (GIS '08), p. 39:1–39:10. ISBN 978-1-60558-323-5. Disponível em: <http://doi.acm.org/10.1145/1463434.1463483>.

LEE, K. C. K.; LEE, W.-C.; ZHENG, B. Fast object search on road networks. In: **Proc. of the 12th EDBT Conf.** [S.l.: s.n.], 2009. p. 1018–1029.

LI, F.; CHENG, D.; HADJIELEFTHERIOU, M.; KOLLIOS, G.; TENG, S.-H. On trip planning queries in spatial databases. In: **Proceedings of the 9th international conference on Advances in Spatial and Temporal Databases**. [S.l.: s.n.], 2005. (SSTD'05), p. 273–290.

NANNICINI, G.; DELLING, D.; LIBERTI, L.; SCHULTES, D. Bidirectional a* search for time-dependent fast paths. In: **Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings**. [s.n.], 2008. p. 334–346. Disponível em: <http://dx.doi.org/10.1007/978-3-540-68552-4_25>.

NANNICINI, G.; DELLING, D.; LIBERTI, L.; SCHULTES, D. Bidirectional A^* on time-dependent graphs. In: **Seventh Cologne Twente Workshop on Graphs and Combinatorial Optimization, gargano, Italy, 13-15 May, 2008**. [S.l.: s.n.], 2008. p. 132–135.

NANNICINI, G.; DELLING, D.; SCHULTES, D.; LIBERTI, L. Bidirectional A^* search on time-dependent road networks. **Networks**, v. 59, n. 2, p. 240–251, 2012. Disponível em: <http://dx.doi.org/10.1002/net.20438>.

OPENGIS. 2015. Http://www.opengeospatial.org/standards. Accessed: 2015-12-10.

ORDA, A.; ROM, R. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. **Journal of the ACM**, v. 37, p. 607–625, 1990.

ORDA, A.; ROM, R. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. **J. ACM**, v. 37, n. 3, p. 607–625, 1990. Disponível em: <http://doi.acm.org/10.1145/79147.214078>.

PAPADIAS, D.; ZHANG, J.; MAMOULIS, N.; TAO, Y. Query processing in spatial network databases. In: **VLDB**. [s.n.], 2003. p. 802–813. Disponível em: <http://www.vldb.org/conf/2003/papers/S24P02.pdf>.

POLYA, G. **How to Solve it: A New Aspect of Mathematical Method**. Princeton U.P., 1945. ISBN 0691080976. Disponível em: <https://books.google.com.br/books?id=hzO1nQEACAAJ>.

PYRGA, E.; SCHULZ, F.; WAGNER, D.; ZAROLIAGIS, C. D. Towards realistic modeling of time-table information through the time-dependent approach. **Electr. Notes Theor. Comput. Sci.**, v. 92, p. 85–103, 2004. Disponível em: <http://dx.doi.org/10.1016/j.entcs.2003.12.024>.

PYRGA, E.; SCHULZ, F.; WAGNER, D.; ZAROLIAGIS, C. D. Efficient models for timetable information in public transportation systems. **ACM Journal of Experimental Algorithmics**, v. 12, p. 2.4:1–2.4:39, 2007. Disponível em: <http://doi.acm.org/10.1145/1227161.1227166>.

QGIS Development Team. **QGIS Geographic Information System**. [S.l.], 2009. Disponível em: <http://qgis.osgeo.org>.

RAPTOPOULOU, K.; PAPADOPOULOS, A.; MANOLOPOULOS, Y. Fast nearest-neighbor query processing in moving-object databases. **GeoInformatica**, v. 7, n. 2, p. 113–137, 2003. Disponível em: <http://dx.doi.org/10.1023/A:1023403908170>.

SAFAR, M.; EBRAHIMI, D.; TANIAR, D. Voronoi-based reverse nearest neighbor query processing on spatial networks. **Multimedia Syst.**, v. 15, n. 5, p. 295–308, 2009. Disponível em: <http://dx.doi.org/10.1007/s00530-009-0167-z>.

SHARIFZADEH, M.; KOLAHDOUZAN, M. R.; SHAHABI, C. The optimal sequenced route query. **VLDB J.**, p. 765–787, 2008.

SIMPLES, T. **Tracking Táxi Simples**. 2015. [Online; accessed 5-novembro-2015]. Disponível em: <https://s3.amazonaws.com/txsimples_tracking/index.html>.

TAO, Y.; PAPADIAS, D.; LIAN, X. Reverse knn search in arbitrary dimensionality. In: **(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004**. [s.n.], 2004. p. 744–755. Disponível em: <http://www.vldb.org/conf/2004/RS20P1.PDF>.

TRAN, Q. T.; TANIAR, D.; SAFAR, M. Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks. **T. Large-Scale Data- and Knowledge-Centered Systems**, v. 1, p. 353–372, 2009. Disponível em: <http://dx.doi.org/10.1007/978-3-642-03722-1_14>.

WAGNER, D.; WILLHALM, T. Speed-up techniques for shortest-path computations. In: **IN PROCEEDINGS OF THE 24TH INTERNATIONAL SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE (STACS'07**. [S.l.]: Springer, 2007. p. 23–36.

WIKI, O. **Main Page — OpenStreetMap Wiki,**. 2014. [Online; accessed 10-dezembro-2015]. Disponível em: <http://wiki.openstreetmap.org/w/index.php?title=Main_Page&oldid=1060762>.

YANG, C.; LIN, K. An index structure for efficient reverse nearest neighbor queries. In: **Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany**. [s.n.], 2001. p. 485–492. Disponível em: <http://dx.doi.org/10.1109/ICDE.2001.914862>.

ANEXOS

ANEXO A – GPS capture file

cabid	latitude	longitude	altitude	accuracy	altitudeaccuracy	timestamp
982335450	-37316237	-385125479	-6	30	98	Mon Nov 16 2015 11:01:43 GMT-0300 (BRT)
982335450	-37316237	-385125479	-6	30	98	Mon Nov 16 2015 11:01:43 GMT-0300 (BRT)
982335450	-37316654	-38512422	-5	29	100	Mon Nov 16 2015 11:01:44 GMT-0300 (BRT)
982335451	-37345546	-384968942	33	47	206	Mon Nov 16 2015 10:01:56 GMT-0300 (BRT)
982335451	-37345546	-384968942	33	47	206	Mon Nov 16 2015 10:01:56 GMT-0300 (BRT)
982335451	-37345544	-384968941	33	48	206	Mon Nov 16 2015 10:01:57 GMT-0300 (BRT)
982335175	-37715261	-385787434	27	12	206	Mon Nov 16 2015 09:59:05 GMT-0300 (BRT)
982335175	-37716108	-385787616	23	12	202	Mon Nov 16 2015 09:59:06 GMT-0300 (BRT)
982335175	-37716108	-385787616	23	12	202	Mon Nov 16 2015 09:59:06 GMT-0300 (BRT)
982335175	-37716167	-385788077	24	12	214	Mon Nov 16 2015 09:59:07 GMT-0300 (BRT)
982335443	-37765462	-384817017	-1	11	110	Mon Nov 16 2015 10:01:23 GMT-0300 (BRT)
982335443	-37765464	-384817017	-1	11	110	Mon Nov 16 2015 10:01:23 GMT-0300 (BRT)
982335443	-37765467	-384817086	0	10	109	Mon Nov 16 2015 10:01:24 GMT-0300 (BRT)
982335454	-37316689	-38512422	-5	29	100	Mon Nov 16 2015 11:01:44 GMT-0300 (BRT)
982335454	-37345543	-384968942	33	47	206	Mon Nov 16 2015 10:01:56 GMT-0300 (BRT)
982335454	-37345523	-384968942	33	47	206	Mon Nov 16 2015 10:01:56 GMT-0300 (BRT)
982335460	-37345523	-384968941	33	48	206	Mon Nov 16 2015 10:01:57 GMT-0300 (BRT)
982335160	-37715265	-385787434	27	12	206	Mon Nov 16 2015 09:59:05 GMT-0300 (BRT)
982335160	-37716154	-385787616	23	12	202	Mon Nov 16 2015 09:59:06 GMT-0300 (BRT)
982335190	-37716143	-385787616	23	12	202	Mon Nov 16 2015 09:59:06 GMT-0300 (BRT)
982335190	-37716187	-385788077	24	12	214	Mon Nov 16 2015 09:59:07 GMT-0300 (BRT)
982335481	-37765432	-384817017	-1	11	110	Mon Nov 16 2015 10:01:23 GMT-0300 (BRT)
982335481	-37765487	-384817017	-1	11	110	Mon Nov 16 2015 10:01:23 GMT-0300 (BRT)
982335481	-37765432	-384817086	0	10	109	Mon Nov 16 2015 10:01:24 GMT-0300 (BRT)

Tabela 6 – File CSV, GPS capture file (Fonte:)