UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

LUÍS GUSTAVO COUTINHO DO RÊGO

BIDIRECTIONAL SEARCH FOR NEAREST NEIGHBORS QUERIES OVER ROAD NETWORKS

FORTALEZA

2017

LUÍS GUSTAVO COUTINHO DO RÊGO

BIDIRECTIONAL SEARCH FOR NEAREST NEIGHBORS QUERIES OVER ROAD NETWORKS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Banco de Dados

Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

Co-Orientador: Prof. Dr. Mário Antonio do Nascimento

FORTALEZA

2017

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)

R267b Rêgo, Luís Gustavo Coutinho do.
Bidirectional search for nearest neighbors queries over road networks / Luís Gustavo Coutinho do Rêgo. – 2017.
38 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2017.
Orientação: Prof. Dr. José Antônio Fernandes de Macêdo.
Coorientação: Prof. Dr. Mário Antonio do Nascimento.

1. k Nearest Neighbors. 2. Road Networks. 3. Spatial Queries. I. Título.

CDD 005

LUÍS GUSTAVO COUTINHO DO RÊGO

BIDIRECTIONAL SEARCH FOR NEAREST NEIGHBORS QUERIES OVER ROAD NETWORKS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Banco de Dados

Aprovada em:

BANCA EXAMINADORA

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

Prof. Dr. Mário Antonio do Nascimento (Co-Orientador)
University of Alberta (UofA)

Prof[a]. Dr[a]. Ticiana Linhares Coelho da Silva
Universidade Federal do Ceará (UFC)

Prof. Dr. Javam de Castro Machado
Universidade Federal do Ceará (UFC)

À memória de minha tia Mazé, que sempre torceu pelo meu sucesso e acompanhou todos os meus passos.

AGRADECIMENTOS

Agradeço a Deus.

“Eu sou de uma terra que o povo padece
Mas não esmorece e procura vencer.
Da terra querida, que a linda cabocla
De riso na boca zomba no sofrer
Não nego meu sangue, não nego meu nome
Olho para a fome, pergunto o que há?
Eu sou brasileiro, filho do Nordeste,
Sou cabra da Peste, sou do Ceará.”

(Patativa do Assaré)

RESUMO

O presente trabalho estuda a consulta dos *k* vizinhos mais próximos em redes de ruas estáticas, considerando Pontos de Interesse Voláteis (PoI-V). Esse novo tipo de PoI tem uma alta frequência de atualização de localização em um mapa e sua disponibilidade é incerta. Aplicações de compartilhamento de caronas representam um bom exemplo de uso dessa consulta: motoristas podem tornar-se disponíveis ou indisponíveis a qualquer momento para aceitarem uma chamada ou ter suas localizações alteradas com frequência. Soluções anteriores utilizam índices espaciais ou demandam uma fase de pré-processamento no algoritmo, fazendo com que as suas utilizações com PoI-V's sejam inviáveis uma vez que se um desses objetos tornar-se disponível ou indisponível, o pré-processamento ou o índice deverão ser refeitos. A solução proposta utiliza uma combinação do algoritmo A* com uma busca bidirecional, direcionando a expansão dos vértices, bem como diminuindo o tempo de processamento da consulta. Técnicas de poda de espaço de busca também são aplicadas para reduzir a quantidade de potenciais PoI-V's a serem verificados. A corretude e a construção do algoritmo são apresentadas, assim como uma avaliação experimental empírica com redes de ruas reais.

Palavras-chave: k Vizinhos mais Próximos. Redes de Ruas. Consultas Espaciais.

ABSTRACT

The present work studies the k nearest neighbors queries in static road networks, considering Volatile Points of Interest (VPoI). This new type of PoI has a high frequency of location update on a map and its availability is uncertain. Car-sharing applications are a good example of this kind of query use: drivers can become available or unavailable at any time to accept a call or have their locations changed frequently. Previous solutions use spatial indexes or require a preprocessing phase in the algorithm, making their use with VPoI's not feasible since if one of these objects becomes available or unavailable, preprocessing or indexing should be redone. The proposed solution uses a combination of the A* algorithm with a bidirectional search, directing an expansion of the vertices as well as reducing the processing time of the query. Search space pruning techniques are also applied to reduce the amount of potential VPoI's to be scanned. The correctness and construction of the algorithm are presented, as well as an empirical experimental evaluation with real road networks.

Keywords: k Nearest Neighbors. Road Networks. Spatial Queries.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND ACRONYMS

*k*NN	*k* Nearest Neighbors
Bi-*k*-NN-R	Bidirectional *k* Nearest Neighbors in Road Networks
GIS	Geographic Information System
GPS	Global Positioning System
IER	Incremental Euclidean Restriction
INE	Incremental Network Expansion
LBS	Location-Based Services
LDSQ	Location-Dependent Spatial Queries
PoI	Point of Interest
VPoI	Volatile Point of Interest

CONTENTS

1 INTRODUCTION

Geographic Information Systems (GISs) have the goal to capture, store, check, and display data related to positions on Earth's surface (INFORMATION.; CHORLEY, 1987). GISs can show different types of information on the same map – for instance, roads, buildings, vegetation, and so on – hence allowing users to see, analyze, and understand patterns and relationships among entities. Since 2005 *web mapping* (HAKLAY *et al.*, 2008), i.e., the process of using maps delivered by Geographic Information System (GIS)s, has evolved rapidly and gave birth to multitudes of applications – among the most popular, Google Maps, Google Earth, Mapbox, and Wikimapia.

Earlier applications and services did not provide many *points of interest* (*Point of Interest (PoI)*)[1]; however, with the aid of crowdsourcing and new mapping techniques, websites and data sources like Google Maps[2] and OpenStreetMaps[3] provided much more detailed geographic information (HAKLAY *et al.*, 2008).

In turn, developers took advantage of these to devise *Location-Based Services (LBS)*, i.e., services that exploit real-time geographical data generated by mobile devices equipped with Global Positioning System (GPS) receivers to enhance the *user experience*. Notorious examples of LBSs are Foursquare, Airbnb, Uber, and Tinder.

Considering the massive diffusion of LBSs in the recent years, users now play a central role in that they *themselves* represent points of interest. Indeed, it is now common to answer questions such as:

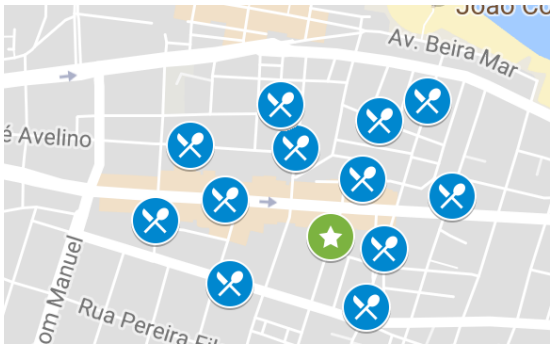
"What is the set of the k closest friends with respect to the user's current location?"
"What is the set of k closest drivers with respect to the user's current position?"

Considering the current massive diffusion of LBSs, and depending on the specific scenario, it is clear that computing the above *k-NN* queries requires dealing with possibly huge amounts of data to be processed in real-time. To this end, Figure 1 provides two different examples, each having different types and quantities of PoIs. More specifically, in Figure 1a we have a few blue circles, each representing a restaurant, and the problem is to find out the k closest restaurants with respect to the user (depicted by the green circle). Similarly, in Figure 1 we have several orange circles, each representing the current position of some person, and a green circle, representing

[1] Examples of PoIs are schools, banks, restaurants, and so on
[2] https://maps.google.com/
[3] https://openstreetmap.org/

the current position of some user; again, the problem is to find out the k closest persons with respect to the user. Clearly, the second scenario requires performing many more computations.

(a) Density of restaurants



(b) Density of users

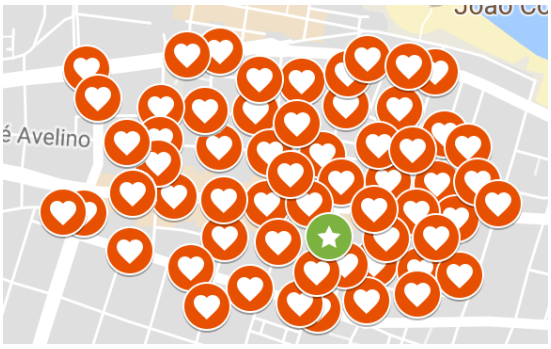


Figure 1 – Density of different Points of Interests.

In general, solving a *k-NN* query requires to find the set of k closest objects with respect to a given query point; in the context of LBSs, devising an appropriate solution depends on the characteristics of the underlying space (e.g., a road-network) and on the ability to manage effectively multitudes of PoIs. Challenges become even harder when having to manage *volatile* PoIs (VPoI), i.e., PoIs that can appear and disappear from the system at any time instant. In this sense, car-sharing applications represent the typical example: registered drivers may become available or unavailable at any time. For instance, let us suppose that a user needs a ride as soon as possible: when the user requests a car the system computes a k Nearest Neighbors (kNN) query, thus returning the k nearest drivers – the first driver that accepts the ride is the one in charge of providing the service.

All the best approaches to compute kNN queries in such scenarios requires a heavy preprocessing step of index creation for a given road network. Calculate some distances and other useful information in advance may be suitable for the application, speeding up the processing time (SAMET *et al.*, 2008; LEE *et al.*, 2009; ZHONG *et al.*, 2013). If a PoI is deleted or added, part of the preprocessing should be then performed again (SHEN *et al.*, 2017).

The main contribution of this thesis consists of an algorithm, Bi-k-NN-R, that deals with the problem of computing k-NN queries over road networks, where the density of VPoIs is high and the number of objects to retrieve,k, is possibly high. The proposed solution does not require a preprocessing step, thus resulting attractive when designing location-based applications or services.

The thesis is structured as follows: Chapter 2 presents important concepts and notions related to this work, as well as well-established algorithms that are leveraged by our solution. Chapter 3 presents the related works. Chapter 4 introduces Bi-*k*-NN-R and proves its correctness. Chapter 5 presents the experimental evaluation, where we employ real-world maps of different countries and study the effects that the density of VPoIs has on the quality of results and performance. Finally, Chapter 6 draws the final conclusions and outlines some possible future lines of research.

1.1 Academic contributions

This thesis is based on the following publications:

1. Magalhães, R. P., Coutinho, G., Macêdo, J., Ferreira, C., Cruz, L., & Nascimento, M. (2015, November). *Graphast: an extensible framework for building applications on time-dependent networks*. In Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems (p. 93). ACM.
2. Magalhães, R. P., Coutinho, G., Macêdo, J., Vidal, V. (2015, Outubro). *Processamento de Grafos em Big Data*. Em Tópicos em Gerenciamento de Dados e Informações (p. 8). SBC.

The first article presents the framework developed during the research process – we also report that the framework is used to conduct the experimental evaluation presented in this work. The second article presents an overview of the literature of graph-processing, with a particular focus on the state-of-the-art.

2 THEORETICAL FOUNDATION

In this chapter we present the foundations on which our work relies. More specifically, Section 2.1 introduces some basic terminology, Section 2.2 introduces the *bidirectional search*, Section 2.3 covers the *Incremental Network Expansion* (INE) algorithm, while Section 2.4 introduces the *A* search*. Finally, Section 2.5 pieces everything together to prepare the ground for the presentation of the Bi-*k*-NN-R algorithm.

2.1 Preliminaries

Definition 2.1.1 (Road network) *A road network can be defined as a directed weighted graph $G = (N, E)$ such that N represents the set of nodes of G, where a node may be associated with an intersection or a geometry point, and E represents the set of edges, where each edge is associated with a road segment (SHAHABI et al., 2002).*

Many functions can be used to calculate the distance between two nodes – some examples are the Euclidean, Manhattan and Chebyshev distances. In this work we focus on the network distance.

Definition 2.1.2 (Network Distance) *Given two nodes $u, v \in N$, we define the network distance between u and v, $d(u, v)$ as the length of the shortest path going from u to v.*

Some nodes in the graph may represent locations of particular importance: we call such nodes *points of interest* (PoIs).

Definition 2.1.3 (Point of Interest) *Given a road network $G = (N, E)$, we define a node $n \in N$ to be a Point of Interest (PoI) if it represents an entity that is important in the context of the application domain considered.*

Common examples of PoIs are restaurants, commercial centers, hospitals, and so on. We finally introduce a particular class of PoIs called Volatile Points of Interest (VPoI).

Definition 2.1.4 (Volatile point of interest) *We define a Volatile Point of Interest (VPoI) to be a PoI that is frequently added or removed from the G.*

Examples of VPoIs can be seen in *mobile* applications (for instance, Uber, Tinder, etc.), where users enter and leave frequently the network.

2.2 Bidirectional Search

In this work we use a variant of the Dijkstra's algorithm based on a bidirectional expansion schema (LUBY; RAGDE, 1989). More precisely, given a directed weighted graph $G = (V,E)$, its reverse $\bar{G} = (V,\bar{E})$ (where $(v,u) \in \overline{E}$ only if $(u,v) \in E$), a source node $s \in V$, and a target node $t \in V$, our variant works by computing two different shortest path trees – one rooted in the source node s in the *original* graph, and the other one rooted in the *target node* t in the *reverse* graph.

The source s will also be the source node of the forward search, but the target node t will be the source node of the backward search. The outline of the bidirectional search, as detailed in (HAKLAY *et al.*, 2008), is presented in the following pseudocode:

1. Let S and T be the empty sets that will contain the visited nodes of the forward and backward searches, respectively. The potential of a node n, $p_{source}(n)$, represents the shortest path from a source node to n. In the bidirectional search, $p_{source}(n)$ is $+\infty$ except for s and t. Let $p_s(n)$ and $p_t(n)$ be zero.
2. Find the node n_0 which has the minimum potential for s in $V \setminus S$ and add n_0 to S. If n_0 is in T. In other words, if the node n_0 is in both searches, jump to step 7.
3. For all nodes n such that (n_0,n) is in E, if $p_s(n_0) + l(n_0,n)$ is less than $p_s(n)$, replace the path from s to n with the path from s to n_0 and the edge (n_0,n), and let $p_s(n_0)$ be $p_s(n_0) + l(v_0,v)$.
4. Find the node n_0 which has the minimum potential for t in $V \setminus T$ and add n_0 to T. If n_0 is in S, then go to step 7.
5. For all nodes n such that (n,n_0) is in E, if $l(n_0,n) + p_t(n_0)$ is less than $p_t(n)$, replace the path from n to t with the edge (n,n_0) and the path from n_0 to t, and let $p_t(n)$ be $l(n,n_0) + p_t(n_0)$.
6. Go to step 2.
7. Find the edge (u,n) minimizing $p_s(u) + l(u,n) + p_t(n)$ such that u is in S and n is in T. The shortest path from s to t consists of the path from s to u and the edge (u,n) and the path from n to t if $p_s(u) + l(u,n) + p_t(n)$ is less than $p_s(n_0) + p_t(n_0)$, otherwise it consists of the path from s to n_0 and the path from n_0 to t.

Figure 2 shows the main difference between the expansion of the unidirectional and bidirectional searches. Dijkstra's algorithm grows a searching circle around the source node. In the bidirectional search, a searching circle also grows from the target node. Thinking of

(a) Dijkstra Algorithm (b) Bidirectional Search

Figure 2 – The difference between search trees of different types of searching algorithms (NANNICINI *et al.*, 2012).

Dijkstra's algorithm as a growing searching circle that represents the nodes expansion, the circle in Figure 2a has a radius equivalent to the Euclidean distance from the source node s to the destination node t. The circles in Figure 2b have a radius equal to the Euclidean distance from s to a meeting node n for the blue circle, and to the Euclidean distance from n to t for the red circle. The area of the two circles in Figure 2b will be smaller than the area in Figure 2a. The bidirectional search will be faster up to a factor or two since it will explore fewer nodes than the Dijkstra's algorithm (NANNICINI *et al.*, 2012).

2.3 Incremental Euclidean Restriction (IER) and Incremental Network Expansion (INE)

(PAPADIAS *et al.*, 2003) proposes two approaches to solve the problem of computing k Nearest Neighbors queries over road networks; the two approaches are called, respectively, *Incremental Euclidean Restriction (IER)* and *Incremental Network Expansion (INE)*. In the following we provide a brief overview.

The IER algorithm relies on the assumption that given any two points a and b lying on the network, their *Euclidean distance* is always *smaller* or equal than their *network distance*. Consequently, given a query point q the algorithm first retrieves its k closest PoIs by means of the Euclidean distance – to this end, a R-tree-based data structure is used. Subsequently, the algorithm determines the farthest PoI among the selected ones, and uses its network distance as threshold. Finally, all the PoIs that have a network distance smaller or equal than the threshold are evaluated, since they represent potential nearest neighbors. In the following we provide an

example (Figure 3) where the IER algorithm is executed with $k = 1$.

(a) 1st Euclidean NN (b) 2nd Euclidean NN

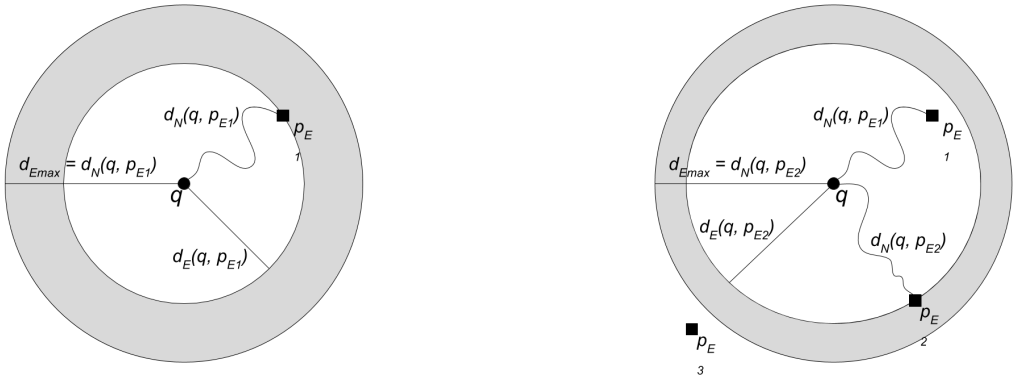


Figure 3 – Finding the NN p_{E2} (PAPADIAS *et al.*, 2003).

Given a query point q, the IER algorithm first determines the q's nearest neighbor by means of the Euclidean distance – in the Figure, this happens to be p_E. Subsequently, the algorithm calculates the network distance between q and p_E and uses it as a threshold (d_{Emax} in Figure 3a). Finally, IER considers all the PoIs whose distance is smaller than the threshold and, if necessary, decreases the threshold accordingly (Figure 3b).

The main problem behind IER is that the Euclidean distance does not represent a good approximation of the network distance, hence driving the method to produce lots of *false hits*.

The INE algorithm tackles these issues by introducing a variant of the Dijkstra's algorithm – more specifically, the key idea is to expand a *search tree* around the query point q until all the q's k nearest neighbors are found. Figure 4 shows an example about how this algorithm works.

Let us consider the query point q and $k = 1$: the INE algorithm first analyzes the edge (n_1, n_2) – in this case no PoI is found and the queue is set to $Q =< (n_1, 3), (n_2, 5) >$. Subsequently, the algorithm expands the next closest node, n_1; since no PoI is found over (n_1, n_7), n_7 is enqueued in $Q =< (n_2, 5), (n_7, 12) >$. The algorithm goes on with the expansion until p_5 is reached when the edge (n_2, n_4) is considered: indeed, p_5's network distance, i.e., $d_N(q, p_5)$, provides the threshold that ends the search tree, since the distance between q and n_4 is above $d_N(q, p_5)$.

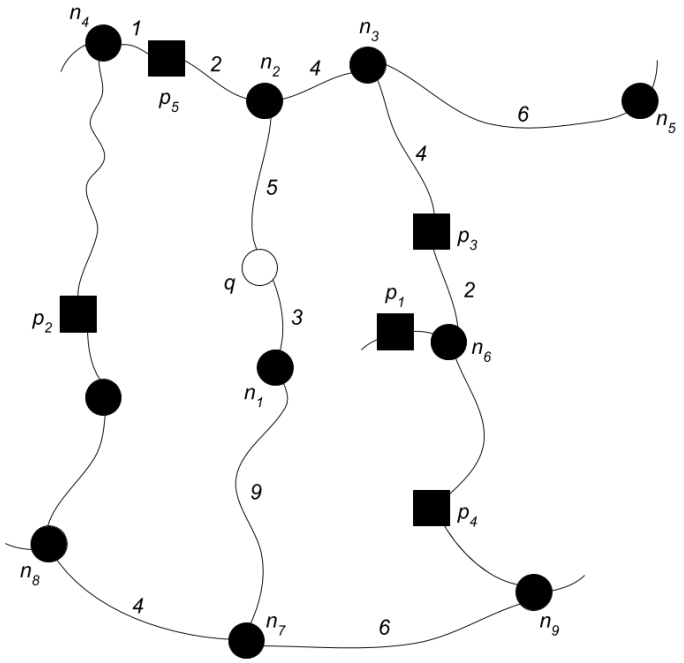


Figure 4 – INE's execution example – the query point, q, is depicted as a white circle, network's nodes are depicted as black circles, while PoIs are depicted as black squares. In the above we have that q's NN is p_5 (PAPADIAS *et al.*, 2003)

2.4 A* Search

The *A* search* is a *goal-directed* algorithm, similarly to Dijkstra's algorithm (HART *et al.*, 1968). The main difference between the two is that the latter exploits the notion of *potential* (heuristic) function $\pi(n)$ to guide the expansion of the search from a node n to the target node t.

More precisely, when a node n gets analyzed, for every neighbor m of n the search adds $\pi(n)$ to the value of the distance between the two nodes, $d(n,m)$ – the goal is to prioritize neighbors that may be closer to the target node (e.g., PoI). We report that when considering road networks the most commonly used potential function is the Euclidean distance, since it represents the minimum admissible value for any shortest path between two nodes.

In general, the choice of a potential function affects the accuracy of the results (i.e., the estimated distance) in two distinct ways:

- $\pi(n) \leq d(n,t)\ \forall n \in V$, where $d(n,t)$ is the distance from n to the target node t: in this case, the potential function never overestimates the exact distance, and the algorithm is capable

to find the shortest path from the source node to the target node.

- $\pi(n) = 0 \; \forall n \in V$: the A* search will perform the same steps of the Dijkstra's algorithm since the potential function will not modify the costs of the edges.

2.5 Problem Statement

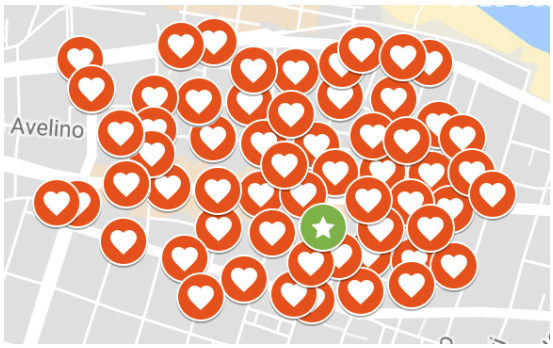
In this work we consider scenarios where we want to compute k-NN queries over road networks with Volatile PoIs. As such, we begin by providing the definition of k-NN queries.

Definition 2.5.1 (k Nearest Neighbors (k-NN) Query over road networks) *Let $G = (V, E)$ be a directed graph and $P \subseteq V$ be a set of points of interest in G. Then, given a query point q and a timestamp t, returning the set of k nearest neighbors query translates into solving the problem of returning the following set of points:*

$$R = \{r \in P \mid (\forall v \in \{P \setminus R\}) \; , \; d_{net}(q,r) \leq d_{net}(q,v)\}.$$

In other words, computing a k-NN query requires to return the k points of interest whose distance with q is the lowest among all the PoIs in P. Figure 5 provides a scenario example.

(a) Overall presence of users in the area



(b) Insertion and deletion of vPoIs

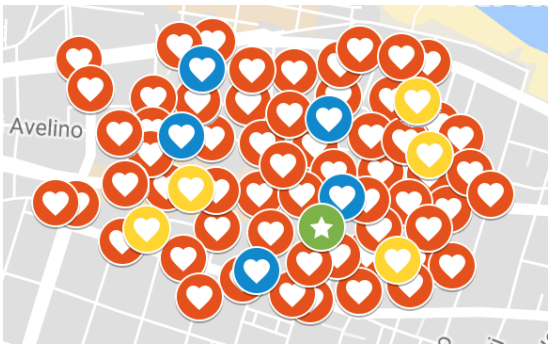


Figure 5 – Scenario example: dating application that requires to track multitides of users (VPoIs) over a road network.

Let us suppose that some application needs to track in real-time multitudes of users over a road network (these are represented by the red circles in Figure 5a), and that users can appear and disappear from the system *at will* – more specifically, users may become available (blue circles) and unavailable (yellow circles) at any time instant (Figure 5b). As such, these

users represent *volatile PoIs*. Let us also suppose that the application should allow an individual user – in the Figure this is represented by the green circle – to find out their k closest users.

3 RELATED WORKS

Computing k Nearest neighbors queries over road networks have been studied for many years in the literature. For instance, (PAPADIAS *et al.*, 2003) consider computing k-NN queries over road networks with static objects (PoIs). In the same work, the authors introduce the INE and IRE algorithms to reduce the overall execution time. With the increasing popularization of online mapping services, for instance, Google Maps and OpenStreetMap, the problem of computing k-NN queries started to present novel challenges when considered with massive graphs, hence demanding more effective strategies. Indeed, the approaches presented in (PAPADIAS *et al.*, 2003) expand visits in a graph without exploiting information associated with promising paths: as such, their performance heavily degrades with large graphs.

(LEE *et al.*, 2009) proposes the ROAD framework to solve Location-Dependent Spatial Queries (LDSQ) on road networks, which incorporate a novel search algorithm for nearest neighbors queries. The underlying idea is to preprocess the original road network to recursively partition it into subgraphs called *Rnets*, and arrange Rnets into a hierarchical structure; this, in turn, allows to pre-compute the distances of the shortest paths associated with pairs of nodes belonging to the border of Rnets. Also, ROAD employs an algorithm, similar to Dijkstra, to incrementally expand and solve k-NN queries. Finally, we report that (ZHONG *et al.*, 2013) show how ROAD exhibits poor performance when considering large road networks with uniformly distributed PoIs.

(SAMET *et al.*, 2008) propose SILC, a solution that employs a pre-computation step that determines the shortest paths between all the nodes of a road network, and stores such distances into a quadtree-based data structure to reduce the overall storage cost. The authors argue that spatial networks seldom exhibit none or minor changes over time, thus focusing the resolution of the problem on changes regarding objects of interest. As such, their solution separates the problem of calculating shortest path distances between pairs of nodes of the network from the problem of finding the k nearest neighbors of some query point. We finally report that (ZHONG *et al.*, 2013) show how SILC is inefficient when considering large amounts of PoIs distributed over small regions. Finally, SILC has a spatial complexity of $O(|V|^{1.5})$ for what concerns the data structures used in its pre-computation step, while newer approaches require $O(|V|log|V|)$.

Similarly to (LEE *et al.*, 2009), (ZHONG *et al.*, 2013) proposes a hierarchical data structure, called *G-Tree*, to compute the nearest neighbors of a query node q. First, the

approach employs a *pre-processing* phase to create a data structure, called *G-Tree*, by recursively partitioning a road network G into subgraphs – this is done by means of a multi-level partitioning algorithm; the approach computes also a distance matrix that contains the distances of the shortest paths between pairs of nodes that belong to the borders of each subgraph. The approach then creates an occurrence list referencing the leaves of the G-Tree containing at least one object of interest or the query point q; occurrence lists are also created for their parent subgraphs until the root node is reached. The information created during the pre-processing phase is finally used to speed up the computation of k-NN queries.

All the techniques presented so far do not consider several important challenges that come with scenarios in which moving objects *frequently* issue their current location over time. (SHEN *et al.*, 2017) attempt to tackle this issue by proposing a new data structure, the *V-Tree*, that builds on the previously introduced G-Tree. More precisely, a V-Tree represents an augmented G-Tree in which information about the current status of moving objects, as well as their relationship with the nodes of the graph, are taken into account. Finally, to compute k-NN queries the authors propose an algorithm that visits the V-Tree.

In general, all the approaches presented so far use some pre-processing step to speed up the computation of k-NN queries. (SHEN *et al.*, 2017) represents the only work that explicitly considers the problem of tracking moving objects over road networks.

4 BIDIRECTIONAL k NEAREST NEIGHBORS SEARCH

In this chapter, we present the approach we propose to compute k-NN queries in network spaces with VPoIs, as per the problem definition provided in Definition 2.5.1. The chapter is organized as follows: Section 4.1 presents our proposal, the Bi-k-NN-R algorithm. Section 4.2 proves Bi-k-NN-R's correctness. Finally, Section 4.3 presents a running example.

4.1 The Bi-k-NN-R Algorithm

In general, existing literature attempt to tackle the problem of computing k-NN queries over road networks by incorporating a *pre-processing* phase in their solution; the role of the pre-processing phase is to create an index whose purpose is to reduce the time spent to compute queries. In this thesis, we propose the *Bidirectional k Nearest Neighbors in Road Networks* (Bi-kNN-R) algorithm. The algorithm combines and leverages the INE, A*, and bidirectional search algorithms. Overall, our approach is structured in two phases:

- **Pruning**: the first phase takes in input a query point q and the set of VPoIs over the road network, and sorts them according to their Euclidean distance from q. VPoIs are subsequently stored in a priority queue, $queueVPoIs$, according to their distance. Section 4.1.1 presents this phase.
- **Bidirectional search**: the second phase takes in input the priority queue returned at the end of the first phase, as well as the query point q, and conducts $k+1$ searches – one backward search for each of the top k VPoIs in $queueVPoIs$ targeting q, and one search that calculate the trees of shortest paths from q. The goal is to find the final q's k nearest neighbors according to their network distance. Searches are conducted in parallel, always *prioritizing* the search having an *expanded distance* smaller than the others. The process goes on until the final set of k nearest VPoIs is determined. Section 4.1.2 illustrates this phase.

4.1.1 Phase I – Pruning

The goal of this phase is to reduce the number of VPoIs that must be considered in the second phase. Algorithm 1 presents the related pseudocode. Given a query point q and a set of VPoIs VP, the algorithm calculates the Euclidean distance between q and all the elements

$vp \in VP$. VPoIs are subsequently stored in the priority queue *queueVPoIs*, where the priority is determined by the Euclidean distance.

Algorithm 1: GETTOPKEUCLIDEAN

Input:
- The query point q.
- The set of VPoIs, $VPoI$.

Output: The priority queue $queueVPoIs$, ordered according to the Euclidean distance between q and the VPoIs in VP.

```
1 begin
2     queueVPoIs ← ∅
3     foreach vp ∈ VPoI do
4         distance ← getEuclideanDistance(q, p)
5         queueVPoIs ← enqueue(vp, distance)
6     end
7     return queueVPoIs
8 end
```

4.1.2 Phase II – Bidirectional search

The goal of the second phase is to determine the set of q's k closest VPoIs according to the *network* distance. To this end, our solution takes advantage of the information available in $queueVPoIs$ and leverages the bidirectional search and A* algorithms.

The key idea is to initially conduct $k+1$ searches: one *forward search* that computes the tree of shortest paths from q (this is done by means of the Dijkstra's algorithm), while the remaining k *backward searches* originate from the k closest (according to the Euclidean distance) VPoIs with respect to q and target the query point – we call these VPoIs *candidate VPoIs*; backward searches are conducted by means of the A* algorithm. Each time some backward search *meets* with the forward search at some node in the graph, the information related to their meet is used to compute the final network distance between their sources.

As the searches progress, it is possible that VPoIs that are not among the candidate ones may turn out potentially closer to q than the candidates: each time this happens, the algorithm creates an additional backward search that is executed in parallel with the other searches. The phase terminates once the final set of k q's nearest VPoIs is determined. Algorithm 2 presents the pseudocode.

First, the functions INITFORWARDSEARCHENTRY and INITSEARCHENTRYQUEUE

Algorithm 2: BIDIRECTIONAL k NEAREST NEIGHBORS SEARCH

Input:
- The query point q.
- The priority queue created during the first phase, $queueVPoIs$.

Output: The final set of q's k nearest neighbors.

```
1  forwardSearchEntry ← INITFORWARDSEARCHENTRY(q);
2  backwardSearchEntryQueue ←
     INITBACKWARDSEARCHENTRYQUEUE(q,queueVPoIs);
3  Res ← ∅;
4  while backwardSearchEntryQueue ≠ ∅ do
5      (backwardSearchEntryQueue, queueVPoIs) ←
         CHECKLOWERBOUND(backwardSearchEntryQueue, queueVPoIs);
6      if CHECKMEET(backwardSearchEntryQueue, forwardSearchEntry) = false then
7          (currentBackwardSearch, backwardSearchEntryQueue) ←
             DEQUEUE(backwardSearchEntryQueue);
8          currentBackwardSearch ←
             EXECUTESTEPSEARCH(currentBackwardSearch, forwardSearchEntry);
9          backwardSearchEntryQueue ←
             ENQUEUE(backwardSearchEntryQueue, currentBackwardSearch);
10     else
11         Res ← ENQUEUE(Res, currentBackwardSearch);
12     end
13 end
14 return GETTOPK(Res)
```

(lines 1 and 2, respectively) create and initialize $k+1$ *searchEntry* data structures, i.e., one entry associated with the forward search, and k entries associated with the backward searches – the latter has the effect of dequeueing the associated VPoIs from $queueVPoIs$. All the k *searchEntry* data structures related to the backward searches are subsequently stored to a priority queue, *backwardSearchEntryQueue*, according to their *expanded distance*, i.e., the distance between their source node s and the node of the *search boundary* that has the lowest distance with respect to s. Each *searchEntry* holds information used to manage the associated search (Table 1): more precisely, it holds (i) a priority queue, $unsettleNodes$, and (ii) a set, $settleNodes$, that keep track, respectively, of the boundary of the search and the set of visited nodes – initially, $settleNodes = \emptyset$ while $unsettleNodes$ contains the neighbors of s. Finally, each *SearchEntry* contains (iii) *meetingNode*, a reference that is used at a later point to refer the node where a backward search meets with the forward search, and (iv) *parentNodes*, a data structure that holds information about the parents of s. Finally, the algorithm creates an empty priority queue, *Res*, (line 3) that is used to subsequently hold the VPoIs considered among the potential nearest

SearchEntry
+ unsettleNodes: Queue<LowerBoundDistanceEntry> + parentNodes: Map<Long, RouteEntry> + settleNodes: Map<Long, Integer> + meetingNode: DistanceEntry

Table 1 – Search Entry structure.

neighbors – the priority is determined according to the Network distance.

After the initialization step, the algorithm starts to compute the initial $k+1$ searches – this is realized by the *while* loop at line 4. The loop iterates until *backwardSearchEntryQueue* is empty, i.e., all the backward searches have terminated. At each iteration, the algorithm first checks if the expanded distance of the search at the top of *backwardSearchEntryQueue* is greater or equal than the Euclidean distance associated with the top VPoI in *queueVPoIs* (function CHECKLOWERBOUND, line 5): if this condition is true, the VPoI represents a potential nearest neighbor. As such, the VPoI is dequeued from *queueVPoIs* and a corresponding backward search is enqueued in *backwardSearchEntryQueue*.

The CHECKMEET function (line 6) determines if any bidirectional search in *backwardSearchEntryQueue* has *met* the $forwardSearchEntry$ at some node of the graph: if this is true, the search is removed from *backwardSearchEntryQueue* and the associated VPoI is added to *Res*. Conversely, the algorithm proceeds to dequeue the top element in *backwardSearchEntryQueue* and stores the related *SearchEntry* in the *currentBackwardSearch* variable (line 7). The function EXECUTESTEPSEARCH is then executed (line 9) to compute a step of the search in *currentBackwardSearch*.

The phase terminates by returning the top k VPoIs in *Res*.

4.2 Correctness of the solution

In the following, we prove the correctness of Bi-k-NN-R.

Theorem 4.2.1 (Correctness of the Bi-k-NN-R algorithm) *Let $R = \{vp_1, \cdots, vp_k\}$ be the set of k VPoIs returned by the Bi-k-NN-R algorithm. Then, R is indeed the set of the k nearest neighbors, according to the network distance, with respect to q.*

Proof. To demonstrate the above statement, we need to prove that Bi-k-NN-R truly returns the correct set of q's k nearest neighbors. Let R_p be the set of VPoIs that Bi-k-NN-R *should consider* during its execution, i.e., the set of the *candidate* VPoIs plus the VPoIs whose Euclidean distance

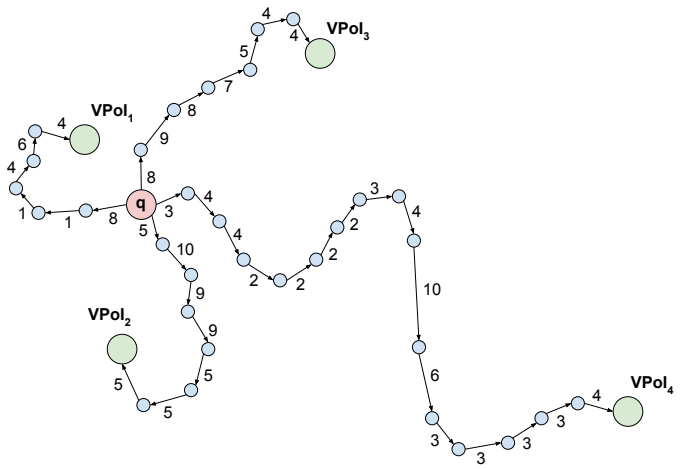


Figure 6 – Running example of the execution of the Bi-k-NN-R algorithm.

happens to be less or equal than the network distance between $VPoI_{r_{max}}$ and q at some point of the algorithm's execution, where $VPoI_{r_{max}}$ represents the farthest nearest neighbor candidate considering the Euclidean distance from q.

Let us also suppose that Bi-k-NN-R does not consider some $VPoI_{r^*} \in R_p$ during its execution: this is absurd, since $d_{eucl}(q, VPoI_{r^*}) \leq d_{net}(q, VPoI_{r_{max}})$ becomes true at some point of the execution; this ensures that only the VPoIs having an Euclidean distance greater than $d_{net}(q, VPoI_{r_{max}})$ are left out by the algorithm.

Once all the shortest paths of the VPoIs in R_p are computed, Bi-k-NN-R picks up the k VPoIs having the smaller network distance with q: this ensures the correctness of the output. □

4.3 Running Example

This section presents a running example of the execution of the Bi-k-NN-R algorithm. Figure 6 presents a toy road-network. Here, the query point q is represented by a red circle, while the four VPoIs are represented by green circles; blue circles represent regular nodes. Finally, let us suppose that $k = 2$.

First, Bi-k-NN-R computes the Euclidean distance d_E between q and all the VPoIs in the graph (Figure 7) – in the example, we have that $d_E(VPoI_1) = 10$, $d_E(VPoI_2) = 20$, $d_E(VPoI_3) = 30$, and $d_E(VPoI_4) = 70$.

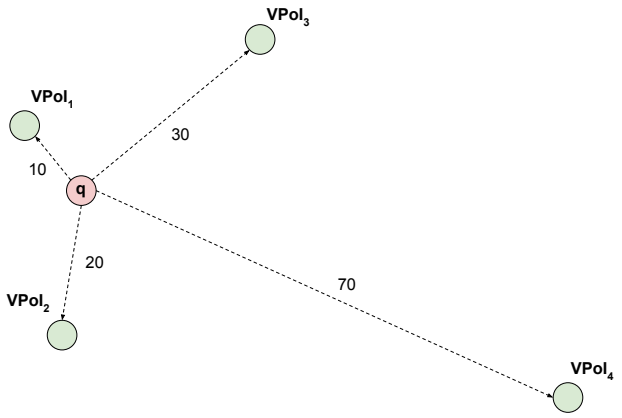


Figure 7 – Computation of the Euclidean distance between q and the VPoIs.

In phase II, the algorithm starts to compute the forward search (related to q) and the backward searches (related to the k closest VPoIs (according to the Euclidean distance). In the example (Figure 7) we see that only $VPoI_1$ and $VPoI_2$ are initially considered since they are the closest VPoIs according to the Euclidean distance. Once the searches reach the status depicted in Figure 8, the algorithm finds out that the Euclidean distance of $VPoI_3$ is lower than the expanded distance of $VPoI_2$, i.e., that also $VPoI_3$ represents a potential nearest neighbor of q. Consequently, a new backward search having origin in $VPoI_3$ is added to the priority queue *backwardSearchEntryQueue*.

As soon as some backward search finishes, the algorithm stores its information to the priority queue *Res*. When all the backward searches finish, the algorithm retrieves the top k searches from *Res*. Figure 9 shows the final iteration, where the algorithm determines that $VPoI_1$ and $VPoI_3$ are the nearest neighbors of q.

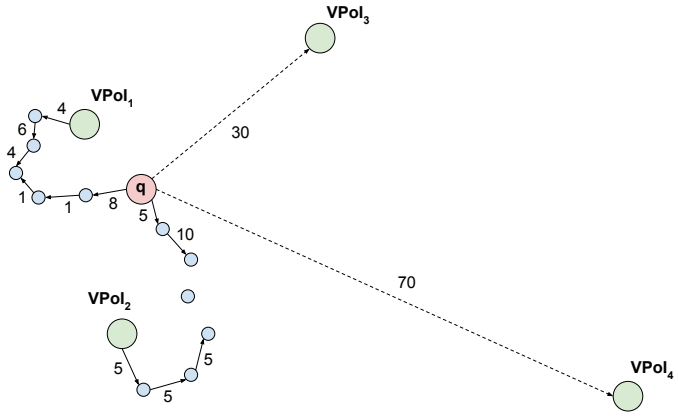


Figure 8 – Example about how the algorithm determines the next search to be expanded.

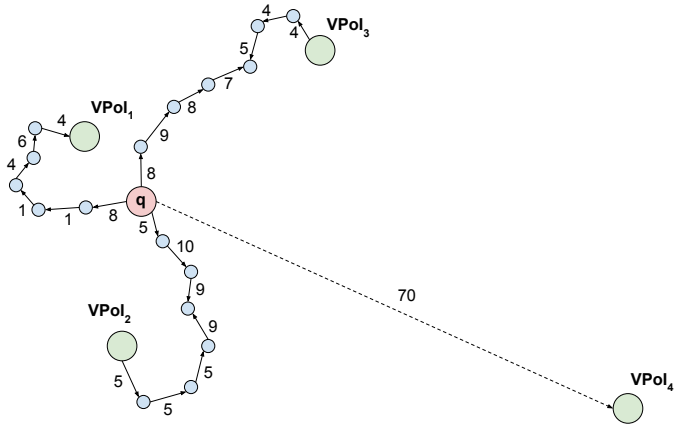


Figure 9 – Algorithm finished and the nearest neighbors are $VPoI_1$ and $VPoI_2$.

5 EXPERIMENTAL EVALUATION

In this chapter we present the experimental evaluation. More precisely, Section 5.1 introduces the experimental setup, while Section 5.2 presents the experiments.

5.1 Experimental setup

5.1.1 Dataset

All maps were imported from OpenStreetMap[1], a collaborative project that allows to create and edit maps. Maps were imported by means of the importer module available in the Graphast framework (MAGALHÃES *et al.*, 2015). Table 2 recap the main characteristics of the road-networks used in the experiments.

Table 2 – Main characteristics of the used maps.

Monaco	#nodes	777
	#edges	1340
	File Size	348KB
Seychelles	#nodes	2284
	#edges	4822
	File Size	786KB
Andorra	#nodes	3496
	#edges	7074
	File Size	1.5MB

5.1.2 Competitors

We implement our algorithm by means of the Graphast framework (MAGALHÃES *et al.*, 2015). We also implemented the Voronoi-based approach as the baseline solution for presented work.

5.1.3 Methodology

The Bi-*k*-NN-R algorithm is evaluated according to three different parameters: *network size* (i.e., the number of nodes in a graph), the *number k* of nearest neighbors to be returned, and the density of VPoI density. The latter measure is calculated as the ratio between

[1] https://www.openstreetmap.org/

the number of VPoIs and the number of nodes in the network. Table 3 provides the ranges of values for the various parameters. The Voronoi approach was also tested with the same parameters.

Table 3 – Parameters used in the experimental evaluation.

Network Size	777, 3494
Network Density	1%, 25%, 50%, 75%, 100%
Query Size (k)	1, 2, 4, 8, 16, ..., (Network Size)

For each combination of network size, density, and query size, 100 queries are executed. For each executed query, the query node q is chosen randomly among the nodes of the network, as well as all VPoIs. For each combination of parameters, in the results we report the average execution time yielded by the 100 queries.

Finally, to validate the shortest paths calculated by our algorithm we used the Graphhopper framework (KARICH; SCHRÖDER, 2014).

5.1.4 Test System

All the experiments were performed on a computer with a macOS Sierra OS, a 2.2GHz Intel Core i7 CPU and 16GB of RAM.

5.1.5 Experimental goals

For this experimental evaluation, the aim is to compare and see how our algorithm behave when the previously mentioned parameters change. The following list summarizes the batches of experiments

- Study on the performance of the algorithms when varying the density of volatile PoIs,
- Study on the performance of the algorithms when varying the query size k,
- Evaluation of the network size.

5.2 Experimental evaluation

In this section, we present the experimental evaluation. In general, all the studies study how some parameter affects the performance of the competitors. As such, the section is structured as follows: in Section 5.2.1 we focus on the density of volatile PoIs; in Section 5.2.2

we focus on the query size; finally, in Section 5.2.3 we focus on of the size of the network.

5.2.1 Study on the performance of the algorithms when varying the density of volatile PoIs

In this batch of experiments, we study how the VPoI density influences the performance of the two competitors. The set of values used is $\{25\%, 50\%, 75\%, 100\%\}$. For each value, the nodes assuming the role of VPoIs were randomly chosen. Figure 10 shows how the ratio between the execution time of the Voronoi-based approach and the Bi-k-NN-R algorithm grows when the density changes in the road network of Monaco.

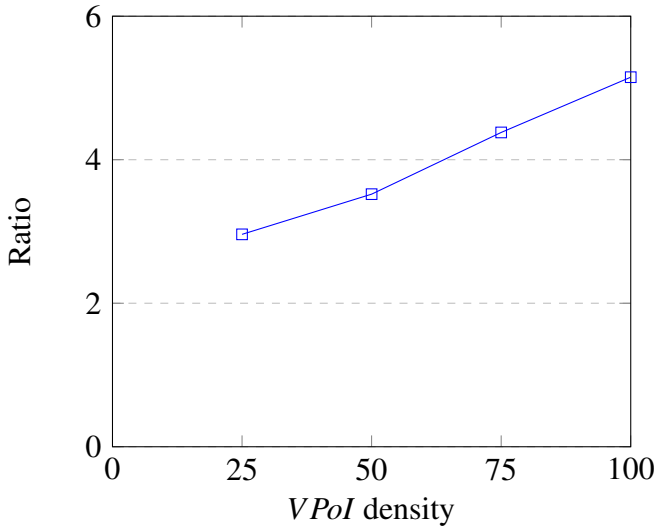


Figure 10 – Ratio growth between the Voronoi-based approach and the Bi-k-NN-R when the VPoI density changes.

5.2.2 Study on the performance of the algorithms when varying the query size k

In this section, we study how variations in the query size k affect the performance of the algorithms. We vary k in the $[1, |VP|]$ range, and use steps that increase according to the power of two. The other parameters are kept fixed to their respective defaults, i.e., the network size is equal to 777 nodes and the density is equal to 100%. Finally, we report that the query point is chosen randomly at each repetition. Figure 11 presents the results.

From the results, we observe that the Voronoi approach outperforms Bi-k-NN-R when $k < 64$.

Analyzing from the user point of view, queries that retrieve less than 64 nearest

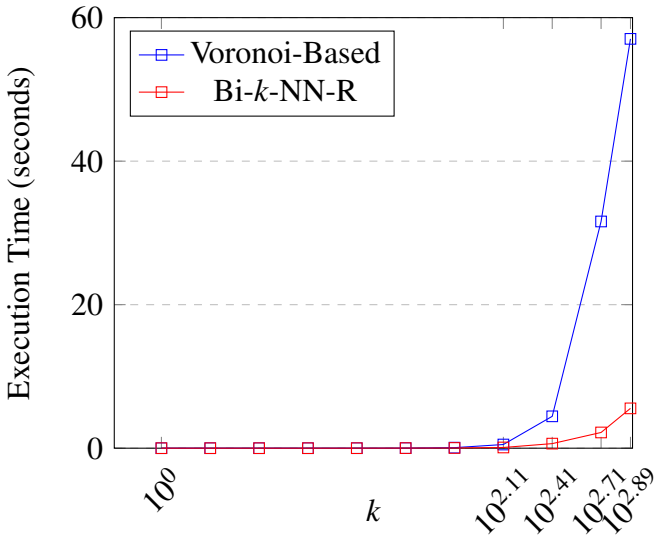


Figure 11 – Execution time for Monaco ($Density = 100\%$).

neighbors are just too fast to notice any difference between the two algorithms. On the other hand, queries that retrieve more than 64 nearest neighbors have a noticeable difference for the final user. For example, the average execution time for $k = 256$ in the Monaco graph with a VPoI density of 100% is 4.4 seconds to the Voronoi approach and 0.6 second to the Bi-k-NN-R algorithm.

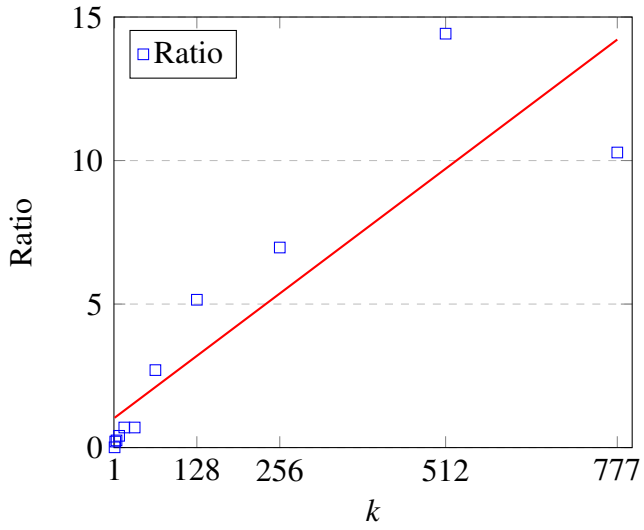


Figure 12 – Curve of the ratio between the two algorithms in Monaco ($Density = 100\%$).

Figure 12 shows the ratio between the results of the Voronoi-based solution and the Bi-k-NN-R algorithm. From the results, we can see that our algorithm achieves speedups lower than 1 when $k < 64$, while in the opposite case it achieves speedups up to 15×.

5.2.3 Evaluation of the network size

For this set of experiments, we considered three different sizes networks: Monaco, Seychelles and Andorra. Setting the density of VPoIs for all graphs in 75% and analyzing the ratio of the execution time between the Voronoi-based solution and the Bi-*k*-NN-R algorithm, Table 4 shows that the ratio decreases when the size of the graph increases.

Table 4 – Network size comparison.

Graph	Maximum number of k	Ratio
Monaco	582	10.78
Seychelles	1713	4.0
Andorra	2622	1.46

6 CONCLUSION

This thesis presents a novel solution, the Bi-*k*-NN-R algorithm, to solve the problem of computing *k*NN queries over road networks with volatile points of interest. The solution incorporates the A* algorithm into the bidirectional search.

We also conduct an experimental evaluation, where we study the performance of our approach with several maps of different countries. The experiments show that the Bi-*k*-NN-R algorithm is up to one order of magnitude faster than the state-of-the-art, i.e., the Voronoi-based approach when considering scenarios where the number and density of VPoIs in the map are high $k \geq 64$.

As a future direction of research, we plan to improve its performance for any number of retrieved VPoIs and density. For example, the new version of the Bi-*k*-NN-R algorithm could also use the A* search to direct the forward search to the closest VPoIs in that iteration. Another possible line of research consists in exploring if some of the techniques used in this work can be incorporated in the resolution of other types of queries – for instance, incorporating the bidirectional search technique when computing *reverse k*-NN queries. Finally, in the future, we plan to extend the experimental evaluation to include larger graphs (e.g., Germany) and compare with the V-Tree approach that is, current, the state of the art for the k nearest neighbors query with moving objects.

REFERENCES

HAKLAY, M.; SINGLETON, A.; PARKER, C. Web mapping 2.0: The neogeography of the geoweb. **Geography Compass**, Wiley Online Library, v. 2, n. 6, p. 2011–2039, 2008.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE transactions on Systems Science and Cybernetics**, IEEE, v. 4, n. 2, p. 100–107, 1968.

INFORMATION., C. O. E. I. T. H. O. G.; CHORLEY, R. **Handling Geographic Information. Report... of the Committee... Chairman: Lord Chorley**. [S.l.]: HM Stationery Office, 1987.

KARICH, P.; SCHRÖDER, S. Graphhopper. **http://www. graphhopper. com, last accessed**, v. 4, n. 2, p. 15, 2014.

LEE, K. C.; LEE, W.-C.; ZHENG, B. Fast object search on road networks. In: ACM. **Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology**. [S.l.], 2009. p. 1018–1029.

LUBY, M.; RAGDE, P. A bidirectional shortest-path algorithm with good average-case behavior. **Algorithmica**, Springer, v. 4, n. 1, p. 551–567, 1989.

MAGALHÃES, R. P.; COUTINHO, G.; MACÊDO, J.; FERREIRA, C.; CRUZ, L.; NASCIMENTO, M. Graphast: an extensible framework for building applications on time-dependent networks. In: ACM. **Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems**. [S.l.], 2015. p. 93.

NANNICINI, G.; DELLING, D.; SCHULTES, D.; LIBERTI, L. Bidirectional a* search on time-dependent road networks. **Networks**, Wiley Online Library, v. 59, n. 2, p. 240–251, 2012.

PAPADIAS, D.; ZHANG, J.; MAMOULIS, N.; TAO, Y. Query processing in spatial network databases. In: VLDB ENDOWMENT. **Proceedings of the 29th international conference on Very large data bases-Volume 29**. [S.l.], 2003. p. 802–813.

SAMET, H.; SANKARANARAYANAN, J.; ALBORZI, H. Scalable network distance browsing in spatial databases. In: ACM. **Proceedings of the 2008 ACM SIGMOD international conference on Management of data**. [S.l.], 2008. p. 43–54.

SHAHABI, C.; KOLAHDOUZAN, M. R.; SHARIFZADEH, M. A road network embedding technique for k-nearest neighbor search in moving object databases. In: ACM. **Proceedings of the 10th ACM international symposium on Advances in geographic information systems**. [S.l.], 2002. p. 94–100.

SHEN, B.; ZHAO, Y.; LI, G.; ZHENG, W.; QIN, Y.; YUAN, B.; RAO, Y. V-tree: Efficient knn search on moving objects with road-network constraints. In: IEEE. **Data Engineering (ICDE), 2017 IEEE 33rd International Conference on**. [S.l.], 2017. p. 609–620.

ZHONG, R.; LI, G.; TAN, K.-L.; ZHOU, L. G-tree: An efficient index for knn search on road networks. In: ACM. **Proceedings of the 22nd ACM international conference on Information & Knowledge Management**. [S.l.], 2013. p. 39–48.