NTNU

Norwegian University of
Science and Technology

Keyword Search on Spatial Network Databases

Road network indexing for efficient query processing

Øystein Egeland Carlsson

Master of Science in Computer Science

Submission date: June 2011

Supervisor: Kjetil Nørvåg, IDI

Co-supervisor: João B. Rocha-Junior, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

With the popularization of geo-tagged data, there is an increasing interest for spatial queries such as nearest neighbor and range. The nearest neighbor query returns the nearest spatial objects from a given query location, while the range query returns the set of spatial objects in a given area. A combination of spatial queries with keyword queries has attracted the interest of the research community recently. In such queries, the user is interested in spatial objects whose description is relevant for a set of query keywords. For example, giving a spatial location and the set of query keywords "italian food", a nearest neighbor keyword query returns the nearest spatial objects relevant for the keywords "italian food". In order to support these queries efficiently, spatial indices such as R-tree and information retrieval indices such as inverted files are employed. In this project, we are interested in a new kind of spatial keyword query that takes in account the road networks. In these queries, the distance between the query location and the spatial objects is constrained by a road network. Each spatial keyword query has a counterpart in the context of road networks. Current solutions that rely on spatial indexes cannot be employed to support these queries because they are restricted to Euclidean distance. Therefore, we plan to investigate how to develop a framework to process spatial keyword queries on road networks efficiently.

Assignment given: 17 January 2011
Supervisor: Kjetil Nørvåg, IDI
Co-supervisor: João B. Rocha-Junior

Keyword Search on Spatial Network Databases

Road network indexing for efficient query processing

Øystein Egeland Carlsson
IDI, NTNU
oysteica@stud.ntnu.no

Abstract

Given a spatial location and a set of keywords, a *spatial keyword query* locates spatio-textual objects based on both the location of the objects, and the textual relevance of the query keywords to the description of the objects. Spatial keyword queries can be used to answer challenging questions such as finding the nearest spatio-textual object relevant for the query keywords "restaurant sushi".

Our focus in this project is on a new type of spatial keyword query that takes a road network into account during query processing. These queries are based on the fact that the distance between two objects in the real-world is constrained by the pre-defined paths that comprise the road network. Different from traditional spatial keyword queries that employ the Euclidean distance, the spatial keyword query on road networks assumes the shortest path between the query location and the objects. Unfortunately, no approach currently exists that supports processing of spatial keyword queries on road networks.

In this thesis we address the challenging problem of locating spatio-textual objects in a road network given a spatial location and a set of keywords. We first propose a baseline framework that combines existing state-of-the-art approaches to support processing of keyword-based spatial queries such as range and k-nearest neighbour on road networks. Then, we present a novel framework termed *Road Network Indexing* (RNI) that permits efficient processing of such queries by indexing the spatio-textual objects in each road segment using inverted files.

Moreover, we present algorithms to evaluate keyword k-nearest neighbour and keyword range queries on both the baseline and the RNI framework. Finally, we show through an experimental evaluation using real-world datasets, that our RNI framework performs nearest neighbour queries on road networks in around one order of magnitude faster than the baseline approach in terms of response time and I/O.

Keywords: spatial keyword queries, spatial databases, road networks, query processing, nearest neighbour queries, and range queries.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The past decade has seen a growth of location-based services (LBSs) on the Web due to the proliferation of mobile devices that come with precise geo-positioning technology, as well as rapid deployment of high-speed wireless communication. In these type of services, location-related information is often retrieved by the use of spatial queries. A *spatial query* finds spatial objects given a location, e.g. objects with a set of latitude and longitude coordinates. Although spatial queries now are gaining popularity due to the increase in geo-tagged data, they have been a part of the database community for a long time. In Geographical Information Systems (GIS), for example, spatial queries can be used to retrieve parts of a map, and in medical imaging applications spatial queries can be used to extract a specific section of an image.

A different query is the spatial keyword query, which is a combination of a keyword query and a spatial query. Keyword queries are intuitive, and familiar for most people through search engines and social networks on the web. On a search engine a user specifies a set of query keywords, and the search engine returns the most relevant results according to the given keywords. A *spatial keyword query* thus finds spatial objects based on both location and the textual relevance of the query keywords to the description of the object. Spatial keyword queries can be used to answer questions such as finding the nearest spatial object relevant to the query keywords "restaurant sushi", and are useful in many applications, including map services and online yellow pages. We call spatial objects annotated with a set of keywords, e.g. a menu of a restaurant, for spatio-textual objects, or just objects for short.

In this thesis we focus on a new type of spatial keyword query that takes into account the road network where the spatio-textual objects reside. These queries are based on the fact that the distance between two objects in real-life applications is constrained by the network of roads and pathways that connect the objects. This is different from traditional spatial keyword queries that employ the Euclidean distance between the objects as the spatial constraint.

For example, Figure 1.1 shows a road network containing spatio-textual objects p annotated with their respective textual descriptions. Consider a tourist visiting Trondheim who wants to find the nearest café from her current location q. The tourist opens a search application on her GPS-enabled mobile phone and poses a spatial keyword

query with *cafe* as the query keyword. The query location q is automatically sent by the mobile phone. In a traditional spatial keyword query, spatio-textual object p_3 is the nearest matching object to q, since its textual description is relevant to the query keyword, and the Euclidean distance from q is the least of all the objects. On the other hand, when we take the road network into consideration, the nearest matching spatio-textual object is p_2. The textual description of object p_2 is relevant to the query keyword, and its network distance to q is the least of all the objects.

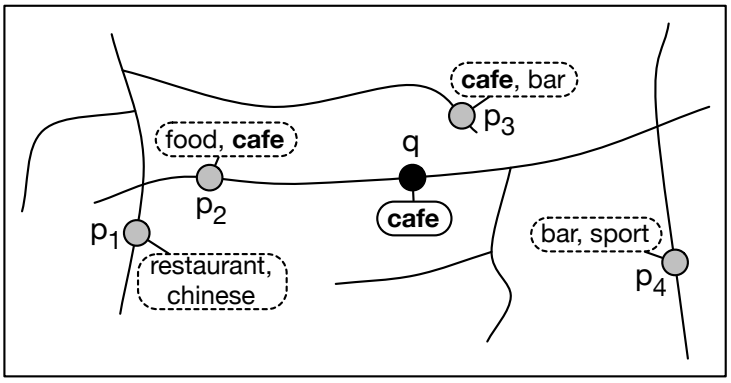


Figure 1.1: Example of a spatial keyword query on a road network.

As presented in the example, spatial keyword queries that utilize the network distance between a query location and the spatial objects returns the best objects according to the constraints of the underlying road network. This is an improvement of the traditional spatial keyword queries where the Euclidean distance is only an approximation of the distance between a query location and the spatial objects.

To the best of our knowledge, no approach currently exists that supports processing of spatial keyword queries on road networks. A solution to this problem is complex, and requires efficient index structures for the spatial and textual part of the spatio-textual objects, as well as an efficient representation of the road network that supports exact computation of the network distance. The current state-of-the-art approach for processing spatial keyword queries is proposed by Cong et al. (2009). The indexing framework they propose supports processing of keyword queries on spatial data by integrating inverted files (Zobel and Moffat, 2006) for text retrieval with an R-tree (Guttman, 1984) for spatial proximity querying, yielding a hybrid index they call the *Inverted file R-tree*, or IR-tree. The IR-tree is an R-tree where each node is augmented with an inverted file for the objects in the children of the node. This means that each node contains a summary of both the textual content and spatial information of the objects in the children of the node, enabling a query to prune the search space using both textual and spatial constraints during tree traversal. The framework of Cong et al. supports spatial keyword queries, and allows indexing of a vast amount of spatio-textual objects, but it does not support spatial keyword queries on road networks.

As for the road network representation, Papadias et al. (2003) proposed a flexible road network framework that supports processing of common spatial queries such as range and k-nearest neighbours (kNN), as well as exact computation of the network distance from a query location to the spatial objects. The framework is based on the network expansion principle, which is similar to how Dijkstra's algorithm operates (Dijkstra,

1959): the edges of a network are gradually expanded from a query point, reporting spatial objects as they are encountered during the expansion. Because of the flexible nature of the framework we consider the approach of Papadias et al. as the state-of-the-art for spatial query processing on road networks. Unfortunately, the framework does not support spatial keyword queries.

So, the primary research question in this thesis is *to study the problem of how to process spatial keyword queries on road networks, and to construct a framework able to do just that*.

In order to address the problem stated in the research question, we first aim to construct a baseline framework that combines the state-of-the-art approaches of Cong et al. (2009) and Papadias et al. (2003), yielding an approach able to evaluate spatial keyword queries on spatial network databases (SNDB) such as common road networks. The road network architecture of Papadias et al. provides support for operations on spatial network databases, while the hybrid index structure of Cong et al., the IR-tree, provides support for processing keyword queries on spatial data.

As a second goal we will address some of the performance issues with the IR-tree of Cong et al. Evaluation of spatial keyword queries on the IR-tree incurs a non-negligible cost since the inverted file of each node has to be accessed during query processing to verify the existence of a keyword in the subtree of a node. In addition, the IR-tree is prone to retrieve false positives. To alleviate these issues, we propose a novel approach for efficient processing of spatial keyword queries on road networks that solely uses inverted files for indexing the spatio-textual objects, yielding a framework we call *Road Network Indexing* (RNI). RNI replaces the IR-tree as the structure for indexing the spatio-textual objects with a set of inverted files, one for each network segment, where each inverted file indexes only the spatio-textual objects that lie on the corresponding network segment. This design allows for fast query processing since we only have to verify the existence of a query keyword in an inverted file to confirm that there exists a matching object on the network segment.

Furthermore, for both the baseline and the RNI framework, we will develop algorithms for processing keyword-based range and k-nearest neighbour queries on the frameworks. Our keyword-based range query retrieves all the spatio-textual objects within range r of a query location q whose textual description matches any of the keywords specified in the query; the keyword-based k-nearest neighbour query retrieves the k closest spatio-textual objects to q that matches any of the specified query keywords.

As part of an experimental evaluation we intend to provide initial performance results of both frameworks, as well as compare the query processing performance of both frameworks, using an implementation of our keyword-based k-nearest neighbour algorithm. The experimental evaluation will use real-world datasets of different size extracted from the OpenStreetMap (OSM) web service. As an additional contribution, we provide procedures and algorithms for extracting these datasets from OSM.

In summary, the main contributions of this thesis are:

- We propose a baseline framework that combines existing state-of-the-art approaches to process spatial keyword queries on road networks.
- We present RNI, a novel framework that permits efficient processing of spatial

keyword queries on road networks, by using inverted files to index the spatio-textual objects on a road segment.

- We propose efficient algorithms for both frameworks for processing keyword-based range and k-nearest neighbour queries.
- Finally, we show through an experimental evaluation that employs real-world data extracted from OpenStreetMap, the efficiency of our RNI framework to process spatial keyword queries on road networks.

The remaining of this thesis is organized as follows. Chapter two, background and related work, establishes a context to the material presented later in the thesis, and provides an insight into already published work on the topic. The third chapter, preliminaries, defines and explains theory and concepts that will be helpful in understanding the problem and our solution. Next, the baseline framework and the RNI framework is presented in detail in chapter four and five, respectively, whereas chapter six introduces OpenStreetMap and the data available from the web service. In chapter seven, initial results on query performance for both frameworks is detailed in an experimental evaluation. Finally, the thesis is concluded in chapter eight, together with a discussion of future work.

Chapter 2

Background and Related Work

In order to establish a context to the material presented later in this thesis, this chapter provides an insight into relevant background information on spatial databases (Section 2.1), road networks (Section 2.2), and information retrieval (Section 2.3). We cover some of the already published literature on these subjects, as well as present a more detailed review on popular index structures such as R-trees and inverted files.

2.1 Spatial databases

When modelling any type of data and subsequently storing the data in a type of database, it is useful to discuss the notion of a data model and a data structure. In the database community a *data model* can be seen as the generic, highly abstract set of concepts, with which a database administrator can describe the data and their relationship, whereas a *data structure* is a generic or specific set of methods or programs used to access data stored in a specific way (Frank, 1992).

The spatial data model thus consists of a set of abstract concepts that can be used to describe any object that has a spatial extent in one or more dimensions. Spatial data, in turn, consist of spatial objects comprised of points, lines, regions, rectangles, surfaces, volumes, and more abstract dimensions such as time (Samet, 1995). Examples of such objects include roads, rivers, cities, forests, mountains, or other geographical landmarks and areas. Each of these objects may have different spatial properties such as length, a defined boundary, latitude and longitude coordinates etc. In addition, it is often interesting to store non-spatial attributes as well, such as the name of a city or a textual description of the object.

In a regular relational database, objects are stored as a collection of tuples, where each tuple consists of several fields belonging to different data types. A spatial object can be stored in such a database by naively creating a field for each of the spatial dimensions of the object, or any of the other spatial properties it is desirable to store. This approach is sufficient for retrieving the data in an uncritical fashion such as a select all query in SQL: SELECT $*$ FROM *table*. The problem arises when a user wants to retrieve a portion of the data based on some given criteria, for example, all the roads that intersect a

specified boundary. In a relational database there are no operations for retrieving data based on the spatial properties of the stored objects, which means that a user manually has to specify all the roads that intersects with other roads or regions.

Spatial databases (SDBs) and multidimensional data structures were introduced to accommodate the need for operations that could support queries using spatial properties. A spatial database is able to efficiently store and query multidimensional data objects such as points, lines, regions, and other objects with a spatial extent, using data structures and indexing methods that takes care of the different spatial properties that each object has.

2.1.1 Spatial indexes

In order to support search operations and efficiently retrieve the data stored in a spatial database we employ a spatial index structure. There exist many different multidimensional access methods and data structures that have been used for indexing spatial data, and an extensive overview has been written by Gaede and Günther (1998).

Among the more popular multidimensional data structures is the *k*-d-tree proposed by Bentley (1975). The *k*-d-tree is a binary search tree where each intermediary node can be seen as generating a $(d-1)$-dimensional hyperplane that splits the space in two parts, alternating the splitting direction among the d possibilities. A related structure is the Quadtree with its many variants (Finkel and Bentley, 1974; Samet, 1984). Similar to the *k*-d-tree, the Quadtree also splits the space into alternating hyperplanes, but an important difference is that the internal nodes of the tree each has 2^d children in a space of d dimensions. Other popular structures are the Grid file (Nievergelt et al., 1984) and the M-tree (Ciaccia et al., 1997).

The most popular index structure for spatial data is the R-tree by Guttman (1984) and its variants, the R^+-tree (Sellis et al., 1987) and the R*-tree (Beckmann et al., 1990). Since the R-tree is widely used among the approaches we discuss in this thesis, we here provide a thorough review of the structure.

R-tree

The R-tree, originally proposed by Guttman (1984), is a dynamic index structure for multidimensional data that supports spatial operations such as intersection, range and nearest search. It is a balanced tree with a structure similar to that of a B-tree, and it is created in a hierarchical fashion by splitting the multidimensional space into nested, and possibly overlapping, bounding boxes or minimum bounding rectangles (MBRs). Moreover, it is completely dynamic, meaning that insertions and deletions can be intermixed with queries, and no periodic reorganizing or re-balancing of the tree is required.

A *leaf node* in an R-tree contains the index records as tuples of the form $(p.id, p.rectangle)$, where $p.id$ is a reference to the data object, and $p.rectangle$ is the n-dimensional rectangle enclosing the spatial object. Although the object reference is usually a pointer to a

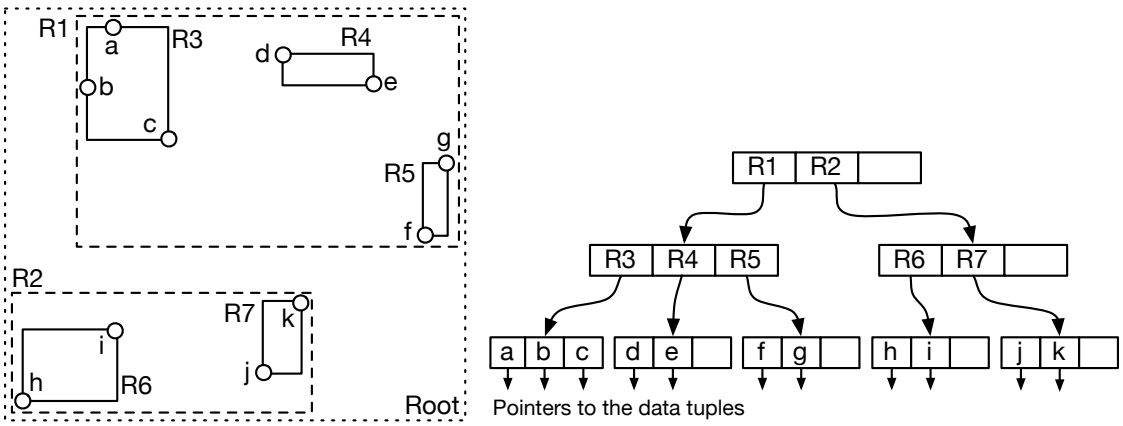
record in a database, the data object itself may be stored in a leaf node entry without affecting the structure of the tree.

Intermediary nodes, on the other hand, contain entries of the form $(node.id, node.rectangle)$, where *node.id* is a pointer to a child node in the R-tree, and *node.rectangle* is the minimum bounding rectangle that encloses all the rectangles of the entries in that child node.

Each node in an R-tree usually corresponds to a disk page, which limits the number of entries in a node to the size of a disk page. Let M be the maximum number of entries that will fit in one node, and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node. An R-tree satisfies the following properties (Guttman, 1984):

- Every leaf node contains between m and M entries unless it is the root.
- Every intermediary node has between m and M children unless it is the root.
- The root has at least two children unless it is a leaf.
- All leaves appear on the same level.

Figure 2.1b presents an R-tree that is created based on the points in Figure 2.1a. The figure shows how the hierarchy of bounding boxes is represented as a tree structure, as well as how the pointers to the child nodes and data tuples are organized. Since we are only using points without any spatial extent for this example the MBRs of the data objects are not shown, and the points are all of equals size, e.g. coordinates representing objects of interest on a map.



(a) The data points and the rectangles enclosing them. **(b)** The structure of the R-tree based on the points in (a).

Figure 2.1: An example of an R-tree for two-dimensional data points.

Searching. Searching in an R-tree uses the MBRs of the intermediary nodes to decide whether an object may be located within a node subtree. A popular query is the intersection query, or window query: given a query rectangle Q_r, retrieve all spatial objects whose MBR intersects with Q_r. The search algorithm for such a query starts at the root and traverses the nodes in the R-tree recursively, visiting only those nodes

whose MBR intersects with the query rectangle; when a leaf node is encountered, the algorithm returns the entries that intersects with Q_r.

Insertion. To insert a new data object in the R-tree, the process is basically to find the best fitting leaf node and insert the new object, splitting a node if it overflows, and propagate this split upwards in the tree if necessary. To select a leaf node for insertion, the algorithm starts at the root and recursively selects the intermediary node whose MBR requires the least expansion to fit the new object, until it reaches a leaf node and subsequently insert the new data object in the node.

Improvements of the original design. There has also been done much work and research on improving the performance and space utilization of the original R-tree as it was proposed by Guttman (1984). One of the variants is the R*-tree of Beckmann et al. (1990), which improves the heuristic optimization of the MBR of each inner node by incorporating a combined optimization of area, margin, and overlap, rather than just minimizing the area as the original R-tree does. This optimization improves the query performance and space utilization by becoming more robust against extreme data distributions and by avoiding node splits, albeit at a slightly higher implementation cost. Another variant is the Priority R-tree (PR-tree) of Arge et al. (2004), which is asymptotically optimal for the worst-case, and at the same time as efficient as the best known R-tree variants.

2.1.2 Spatial queries

In the field of geographic information systems and spatial databases the problem of processing common spatial queries such as range and k-nearest neighbours has retrieved wide attention the last few decades. In order to process such queries efficiently most methods utilize index structures built on the spatial data to aid the search algorithms in retrieval of the data objects. A review of the most important index structure for spatial data, the R-tree by Guttman (1984), can be found in the previous section.

For processing kNN queries on R-trees the first approach, and in retrospect perhaps the most influential, is the branch-and-bound R-tree traversal algorithm proposed by Roussopoulos et al. (1995). The branch-and-bound algorithm traverses the R-tree while keeping a priority queue of the k potential nearest neighbours, as well as applying distance metrics to guide the search and prune unnecessary subtrees. Hjaltason and Samet (1999) improve the kNN query of Roussopoulos et al. with an approach based on an R*-tree (Beckmann et al., 1990) that can retrieve the nearest neighbours to a query point incrementally. This means that while having already retrieved k results, one can retrieve the $k+1$ nearest neighbour without having to compute the $k+1$ nearest neighbours from scratch, thus avoiding redundant computations. However, neither of these approaches account for keyword-based spatial queries, nor do they support spatial queries on road networks, and are thus not applicable to our problem.

2.1.3 Spatial keyword queries

Research on keyword-based spatial queries followed from the increasing popularity of location-based services on the web, giving way to new problems such as finding web pages whose content is related to a particular place or region, and how to represent the spatial part of the web content so as to efficiently process spatial keyword queries on the search engines. As some of the initial contributors to the subject, Zhou et al. (2005) propose three approaches using a hybrid index structure that combines inverted files (Zobel and Moffat, 2006) with R*-trees (Beckmann et al., 1990). The three approaches are: 1) indexing the web pages in both the inverted files and the R*-trees, 2) building an R*-tree for each distinct keyword, and 3) build an R*-tree with the keywords in the leaf nodes of the tree. Through experiments, they found that the second and third approaches performed best, with a slight advantage to the second approach. Similar to Zhou et al., Chen et al. (2006) also focused on improving the performance of spatial keyword queries in search engines. Both approaches have an information retrieval perspective in that they find relevant documents based on pre-defined spatial regions, but they do not support exact location of objects, nor is their work related to query processing on SNDB.

Different from the information retrieval perspective of the work mentioned above are spatial keyword queries that retrieve objects with a specific location. An approach that in some sense builds on the third approach of Zhou et al. (2005) is the KR*-tree proposed by Hariharan et al. (2007). In the KR*-tree, the nodes are augmented with a set of keywords that appear in the objects of the subtrees rooted at the node, with the intention that branches in the tree can be pruned during query processing based on both spatial and textual constraints. Their approach supports retrieval of objects within a region by using a conjunctive keyword query, but it is not applicable to retrieving objects in SNDB.

Top-k spatial keyword queries are queries that, given a location and a set of keywords, returns a ranked set of the k best objects according to a combination of their distance to the query location and the relevance of their textual description to the query keywords. Felipe et al. (2008) address the problem of efficiently processing such queries by integrating R-trees and signature files (Faloutsos and Christodoulakis, 1984) in a hybrid index structure they call the IR^2-tree. Each node in the IR^2-tree contains a superimposed bit signature derived from the keywords of the objects in the subtree, and the idea is that entire subtrees can be pruned during query processing if a node cannot match the query keywords. The limitations with this approach is that it only supports Boolean queries, and due to the bit signatures it is limited to a small amount of keywords per object. The approach is also subject to false positives, since several keywords can be represented by the same signatures. Differently from the work of Felipe et al. (2008), Zhang et al. (2009) address the problem of retrieving the spatially closest objects matching m user-specified keywords. To solve this problem they propose a spatial keyword query they call the m-closest keywords (mCK) query: given m keywords provided by the user, the mCK query finds the closest tuples in space that match these keywords. To index the keywords and the spatial objects, Zhang et al. employ an extended R*-tree they call the bR*-tree. The bR*-tree is in fact very similar to the IR^2-tree of Felipe et al., and is thus subject to the same limitations. Additionally, the mCK query

does not consider objects with a specific location.

To the best of our knowledge, the top approaches for processing top-k spatial keyword queries were developed concurrently by Cong et al. (2009) and Li et al. (2010). Both approaches are very similar in that they both propose to augment each node of an R-tree with a document summary such that each node records a summary of both the textual content and the spatial information of the objects in the subtree of the node. During query processing the document summary enable both approaches to prune parts of the search space based on both textual and spatial constraints, enabling efficient retrieval of the top-k spatio-textual objects, ranked by a function of textual relevance to the query keywords and spatial proximity to the query location. Additionally to the approach described above, Cong et al. extends the R-tree further by incorporating document similarity when constructing the tree, resulting in a structure they call the DIR-tree. In the same work Cong et al. also propose to cluster the nodes of the DIR-tree to further improve query performance, yielding the CIDR-tree. We thus consider the approach of Cong et al. to be the state-of-the-art, and will therefore employ their approach in our baseline framework for spatial keyword processing on spatial network databases. An extended review of the approach of Cong et al. can be found in Section 3.3.2 in the Preliminaries chapter.

A very recent work by Rocha-Junior et al. (2011) proposes a new indexing structure to improve the performance of top-k spatial keyword queries. The approaches of Cong et al. (2009) and Li et al. (2010) incurs a non-negligible cost since the inverted file of each node has to be accessed during query processing to verify the existence of a keyword in the subtree. Rocha-Junior et al. propose to map each keyword in a vocabulary to a distinct aggregated R-tree (aR-tree) (Papadias et al., 2001), effectively replacing the postings lists of an inverted file with an aR-tree that indexes the spatial objects. By running experiments Rocha-Junior et al. found that the query processing performance of their approach outperforms the approach of Cong et al.

2.2 Spatial network databases

Query processing on spatial network databases differs from query processing on spatial databases, since the objects on a road network are restricted to move on the predefined paths of the underlying network. This has the important consequence that the path between two objects depends on the actual connectivity of the transport network, rather than the relative distance between the objects. Consequently, the algorithms for processing the SNDB counterparts of the common spatial queries such as range and k-nearest neighbours have to account for the connectivity of the network in order to provide a correct answer to the queries. Processing spatial queries on SNDB has been the source of much research in the recent years (Papadias et al., 2003; Jensen et al., 2003; Kolahdouzan and Shahabi, 2004; Hu et al., 2006b; Shaw et al., 2007; Lee et al., 2009).

One of the fundamental problems of SNDB is how to model the spatial network in order to provide an efficient structure for query processing. Perhaps the most intuitive approach, and certainly the most common, is to represent the road network as a graph where the vertices of the graph correspond to the road junctions, and the edges of the

graph corresponds to the road segments. An early approach that focuses on how to store and access the graph on disk, in order to minimize I/O during query processing, is the connectivity-clustered access method (CCAM) of Shekhar and Liu (1997). CCAM applies a topological sorting of the nodes of a network and heuristically clusters them in disk pages based on their connectivity; the goal is to allocate them to a common disk page to reduce I/O when later accessing the nodes. The limitation with the CCAM approach, and similar approaches, is that it only preserves the connectivity of the graph, not the spatial location information.

A common paradigm where networks are represented as graphs is the network expansion paradigm. During network expansion, the edges of a network are gradually expanded from a given query location, reporting any matching object discovered during the traversal. The network expansion process is based on the single-source shortest path algorithm of Dijkstra (1959). Dijkstra's algorithm starts at a single node in the network and keeps a priority queue of the visited nodes whose adjacent nodes are not yet visited; the queue is sorted on the distance from the source to the nodes. Related to the proposal of Dijkstra is the A* algorithm (Kung et al., 1986) that employs heuristics to decide which node to expand, also solving the shortest path problem.

Papadias et al. (2003) propose a framework for spatial query processing on SNDB that supports the most common spatial queries such as range, *k*NN, closest-pair, and spatial join. Their approach consists of a three-component network storage scheme that is flexible and separated from the feature sets, i.e. sets containing spatial objects belonging to a certain category such as restaurants or hotels. Unfortunately, the approach does not support spatial keyword queries. A detailed description of the network expansion framework of Papadias et al. can be found in Section 3.3.1 in the Preliminaries chapter. In the same work, Papadias et al. also propose a framework based on Euclidean restriction. Euclidean restriction exploits the Euclidean lower bound property, which says that the network distance between two points is always greater than or equal to the Euclidean distance between the points. They found, by experiments, that the network expansion framework outperforms Euclidean restriction for range and *k*NN queries.

A related approach that solves the problem of answering active, ordered *k*NN queries for moving objects in a road network is the work of Jensen et al. (2003). In their work, Jensen et al. propose a generic framework and data model, and define the abstract functionality necessary to answer nearest neighbour queries on SNDB. In addition, they present detailed algorithms for nearest neighbour search and show initial results from a prototype implementing the algorithms. The road network model is similar to the approach of Papadias et al., in that it takes a road network and constructs a graph representation of the network. Moreover, the limitations are the same as for Papadias et al., i.e. the approach does not support keyword-based spatial queries.

Kolahdouzan and Shahabi (2004) present an approach based on pre-computation of query results that aims to solve the problem of finding the k nearest neighbours to a query point in SNDB, employing first order network Voronoi diagrams (NVD). Their approach, called the Voronoi Based Network Nearest Neighbour (VN^3), is based on reducing the problem of distance computation in a very large network into a problem of computing distances in several smaller regions, together with some additional table

lookups. VN^3 first partitions a large network into smaller and more manageable regions by generating a first order NVD over the points deemed as interesting. Each cell, or network Voronoi polygon (NVP), of the Voronoi diagram is centered by one object, the generator of the polygon, and contains the nodes that are closest in network distance to that object. The second task is to pre-compute the intra and inter distances for each NVP, i.e. pre-computing the distances between all edges, or border points, of the cell to its centre, as well as from the border points to the neighbouring cells. During query processing, the *k*NN query they propose first finds the nearest neighbour of a query point q by locating the NVP that contains q. This can be achieved by searching an R-tree built on the Voronoi cells. Second, they use the intra-cell pre-computed distances to find the distance from q to the border of the Voronoi cell of each candidate. Finally, the inter-cell pre-computed distances are used to compute the actual network distance from q to each candidate, finding the rest of the $k-1$ neighbours. The VN^3 approach is generally fast, however, as commented by de Almeida and Güting (2006), for increasing values of k it gets computationally more expensive, because many paths can be traversed between the Voronoi cells. Voronoi diagrams are also more complex to handle than, for example, a network expansion algorithm (Papadias et al., 2003), and the size of the datasets will also affect the complexity of the computation.

Later, Shaw et al. (2007) present an approach for processing spatial network queries on SNDB that is based on the use of M-trees (Ciaccia et al., 1997) for indexing the spatial objects. The M-tree organizes data according to any arbitrary metric, and in the case of road networks the metric is the network distance between the objects in the network. As a solution for computing the network distance they propose to use a Road Network Embedding (RNE) technique, transforming the coordinates of the network objects into a higher dimensional space (Shahabi et al., 2002). In this higher dimensional space, they use the Chessboard distance metric that is similar to the Euclidean distance. In RNE, this distance is easy to compute, and gives an approximation of the network distance between objects. The embedding technique in RNE requires a substantial amount of pre-computation of all the network distances between all the objects in the network. To reduce the amount of pre-computation, and thus the amount of storage space required, they use the notion of *truncated* RNE, where the shortest path is computed to only a small percentage of the objects in the network. The reason for this is that only a subset of the dimensions of the embedded space are necessary for the calculation of the network distance.

Although the approach of Shaw et al. proves to be fast, in general faster than the VN^3 approach of Kolahdouzan and Shahabi, it has significant drawbacks. The use of the Chessboard metric in the truncated RNE means that the network distance cannot be guaranteed to be correct, it is only an approximation. They show that the solution will actually underestimate the distance to the objects, and range queries will thus not leave out any results because of the inaccurate distance measure, but the result of a *k*NN query may prove to be wrong. Range queries will, however, have to go through a refinement step where the objects that are farther away than the set range will have to be removed from the result.

An approach that reduces the road network to a set of interconnected trees is proposed by Hu et al. (2006b). On each interconnected tree, called a SPIE, an *nd* index is built

which is a pre-computed nearest neighbour result, used to improve performance of a nearest neighbour search. A study with focus on improving the distance computation in SNDB is the Distance Index of Hu et al. (2006a). They propose to construct a distance signature, which on each node stores distance information to all objects in the network, with coarser information for objects far away, and more precise information for nearby objects. The distance signatures subsequently guide range and kNN search during the query process. ROAD is a framework by Lee et al. (2009) to evaluate Location-Dependent Spatial Queries (LDSQ), i.e. a general query defined by a distance condition, and an attribute predicate (e.g. object type = 'restaurant'). ROAD partitions the road network into regional sub-networks (Rnets), and exploits the fact that there are large parts of a road network that do not contain any objects of interest, and can thus be pruned from the search space during query processing. The Rnets consists of a set of border nodes, and the edges between these nodes can be used as a shortcut to avoid traversing the whole network contained within the Rnet. The common limitation for the three aforementioned approaches is that they do not support spatial keyword queries on SNDB, and are thus not applicable to solve our problem.

Although not directly related to our problem, an interesting work done by Yiu and Mamoulis (2004) study the problem of clustering objects that lie on the edges of a large spatial network. They propose a novel transformation of a road network that could potentially be used in an approach for processing spatial queries such as range and kNN. The idea is to transform the weighted graph representation G of the road network, to a new graph G', where each node n in G' is an object p from the original network G. Furthermore, there is an edge between two nodes n and n' in G' if one can move from one object to another in the original network, not passing via any other object. Such a network transformation can likely be applied to the network expansion approaches of Papadias et al. (2003) and Jensen et al. (2003), where all the objects of a feature set is reported as an answer when encountered in the network, making each network node in the transformed graph an answer. In our approach, on the other hand, a spatio-textual object is reported only if it contains any of the keywords specified in the query. Thus, by running a spatial keyword query on the transformed graph, the process runs the risk of expanding a very large part of the network before discovering any matching object.

2.3 Information retrieval

According to Manning et al. (2008):

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Data and information are two terms which in many contexts may be perceived as the same concept, but it is important to separate the field of data retrieval and information retrieval. In general, we can say that the goal of data retrieval is to return an exact answer containing all the documents or objects that matched a specific set of query

terms, given a clearly defined data retrieval language with a specific syntax and semantics. An example is the traditional relational database. The data stored in such a database have a clear structure defined in the database schema, and the language used to retrieve the data, e.g. SQL, has a well defined syntax. If an object that does not match the SQL query is returned, then something has gone wrong and the result is considered incorrect.

The goal of information retrieval, on the other hand, is to present the user with information about a subject or topic that is most relevant to the provided keywords or query phrases, while not returning information that is considered irrelevant. A traditional web search is the prime example here. A user enters a set of query terms, and the search engine returns a set of hits considered most relevant according to the specified terms. Some of the hits may be exactly what the user wanted, and others may be completely irrelevant for the information need of the user. The problem with information retrieval is to actually decide what should be marked as relevant; how can relevancy be quantified.

2.3.1 Inverted files

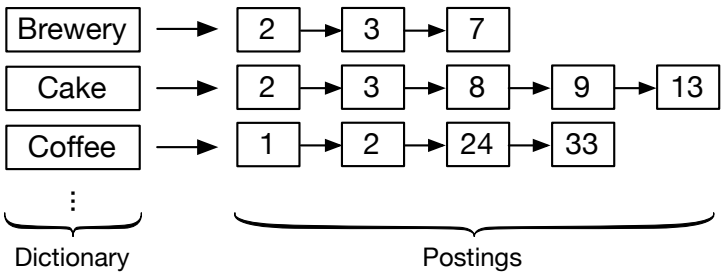


Figure 2.2: The structure of an inverted index.

In order to efficiently retrieve documents and objects that contain the keywords specified in a query we use an index. The *inverted file*, or *inverted index*, is likely one of the most popular and efficient index structures for textual data (Zobel and Moffat, 2006). A basic inverted file[1] is just a data structure mapping terms to the documents they appear in. The name inverted index is actually redundant, since an index always maps back from terms to the documents they appear in Manning et al. (2008).

Figure 2.2 depicts the structure of an inverted file. On the left hand side of the figure we have the dictionary, also called the vocabulary, which is the set of terms in the document collection being indexed. Each term in the dictionary points to a postings list, as seen on the right hand side of the figure. A postings list is a sorted array of postings, where each posting contains the document ID in which the corresponding term can be found. This separation enable us to store the dictionary and the postings lists in different locations, connecting them with a pointer as shown in the figure. Since the dictionary often is much smaller in size than the postings lists, it is often kept in

[1] In this thesis, inverted files and inverted indexes are used interchangeably

memory, while the pointer of a term points to the disk page where the corresponding postings list can be found.

There are several ways to represent a postings list, one way being a sorted array, another as a linked list. In Figure 2.2 the postings lists are represented as a linked list, allowing dynamic updates and additions of documents. In addition to storing just the document ID in a posting, it can be useful to store the position of each occurrence of each term within a document. This of course demands more storage, but it enables us to jump directly to the term instead of searching through the whole document. An effective implementation of the dictionary can be achieved by using a hash table or a search tree.

2.3.2 Retrieval models

In information retrieval, and text databases in general, there are two main models of query, Boolean and ranked.

Boolean query model. The *Boolean query model* consists of the class of queries that can be constructed by the conjunction, disjunction or negation of a set of query terms. That is, a query can be posed as a Boolean expression where each term is combined with either of the operators AND ($\wedge$), OR ($\vee$), or NOT ($\neg$). A conjunction of terms means that every term must be present in the result, whereas a disjunction means that one or more of the terms may be present. For example, in a collection of documents, the document d is the answer to a conjunctive query $t_1 \wedge t_2 \wedge \cdots \wedge t_m$ if it contains every t_i for $1 \leq i \leq m$; similarly, d is the answer to a disjunctive query $t_1 \vee t_2 \vee \cdots \vee t_n$ if it contains any t_i for $1 \leq i \leq n$ (Zobel et al., 1998). In the Boolean query model each document is viewed just as a set of words (Manning et al., 2008).

Ranked query model. In the *ranked query model* the query may consist of a single keyword, a sentence, or a full text. For all purposes the query is viewed as simply a set of words, without any connecting logical operators. A scoring mechanism for this approach can be to calculate a similarity score between the query and each document in the document collection. Similarity is defined by a mathematical function that approximates the likelihood that the document is an answer, and may consist of a large number of parameters. After the similarity score has been calculated for each document against the query, they are ranked according to a given criteria and presented to the user as an answer (Zobel et al., 1998). The simplest approach to achieve a ranking of documents is the *term frequency* weighting scheme, denoted $tf_{t,d}$. In this approach we want to compute a score between a query term t and a document d, based on the weight of t in d. This can be achieved by setting the weight to be equal to the number of occurrences of term t in document d. Thus, the documents can be ranked based on the set of weights determined by the calculated tf weights (Manning et al., 2008).

An additional type of query is the *proximity query* in which answers must contain the specified terms within a specified distance of each other. Such a query may be useful

in phrase searches, and the special case where the proximity is 1 is known as *adjacency*.

In the frameworks we propose later in this thesis we employ a version of the ranked query model. A more detailed description of this can be found in Section 3.2 in the Preliminaries chapter.

Chapter 3

Preliminaries

In this chapter we will define concepts and explain theory that will be helpful in understanding the problem we address in this thesis, as well as provide a better understanding of the two frameworks we present in the following two chapters. In the subsequent section we define road networks and present related notation used throughout this thesis. Section 3.2 formally defines the keyword k-nearest neighbour and keyword range queries, and provides an example showing their operation on a road network. Finally, Section 3.3 provides a detailed review of the road network architecture of Papadias et al. (2003) and the hybrid index for spatial keyword queries of Cong et al. (2009).

3.1 Road network properties

Throughout this thesis we assume a digitization process that takes as input a spatial network and generates a modelling graph that represents the spatial network. The graph is subject to certain properties that are introduced, and formally defined, in this section.

A road network is modelled as a weighted graph $\mathcal{N} = (N, E)$, where N is the set of nodes in the graph, and E is the set of edges. A node $n \in N$ corresponds to a road intersection or an end-point in the road network, whereas an edge $(n, n') \in E$ corresponds to the road segment[1] connecting nodes n and n'. Each edge is assigned a positive real number as the weight, denoted as $|n, n'|$. $|n, n'|$ may represent the actual travel distance, the time used to travel the road segment, or a constraint such as the speed limit. A path $P(u, v)$ in the modelling graph is the set of edges connecting nodes u and v, with the path distance as $|P(u, v)| = \sum_{(n,n') \in P(u,v)} |n, n'|$. Furthermore, the *shortest path* $P_{min}(u, v)$ is defined as the path of shortest distance among all the possible paths connecting the two nodes u and v. Subsequently, $|P_{min}(u, v)|$ is defined as the distance of the shortest path, i.e. the *network distance* between nodes u and v, denoted by $||u, v||$.

S denotes the set of spatio-textual objects present in the road network. A *spatio-textual*

[1] In this thesis, the terms *edge*, *network segment*, and *road segment* are used interchangeably.

object $p \in S$ is an object with a spatial location, e.g. a set of latitude and longitude coordinates, and a textual description *p.doc*. We assume that each spatio-textual object p belongs to one edge $(n, n') \in E$, and that any object p located outside an edge belongs to the nearest edge.

We let $O(n, n')$ represent the set of objects that are present on edge (n, n'), and the *network distance* from an object $p \in O(n, n')$ to the nodes n and n' is represented by $\delta(p, n)$ and $\delta(p, n')$, respectively. Given two arbitrary objects p and q, where p lies on edge (n_a, n_b) and q lies on edge (n_c, n_d), the *network distance* $\Delta(p, q)$ is the distance of the shortest path from p to q. $\Delta(p, q)$ is defined by $min_{x \in \{a,b\}, y \in \{c,d\}}(\delta(p, n_x) + ||n_x, n_y|| + \delta(n_y, q))$.

3.2 Processing spatial keyword queries on road networks

In this thesis we address the challenging problem of searching for spatio-textual objects in a road network given a spatial location and a set of descriptive keywords. We employ a version of the ranked retrieval model, which in this context means that the textual description of each spatio-textual object is seen as a *bag of words*. Under this model, the describing document of each spatio-textual object is represented as a vector, with each vector component corresponding to a unique term in the total collection of describing documents, i.e. the document collection. Each component is assigned a weight based on whether a term is present in the corresponding textual description. For terms that are not present in the description, the weight is zero. By also representing the set of keywords in a query as a vector, the two vectors can be compared to determine whether the spatio-textual object is a match for the query. That is, if the description of a spatio-textual object contains any of the keywords in a query, the object will get a $rank > 0$, indicating that it is a match.

Following is a formal definition of two spatial keyword queries, namely the *keyword k-nearest neighbours* query (K*k*NN) and *keyword range* query (KR).

3.2.1 Keyword *k*-nearest neighbours

$Q_{KkNN} = \langle q, W, k \rangle = \langle q, w_1 \dots w_n, k \rangle$, given a query point q, a set of query keywords W, and a value k. In the set of spatio-textual objects S, the keyword k-nearest neighbours query retrieves the $k \geq 1$ objects closest to q according to the network distance, and whose $rank > 0$, based on the set of keywords W. Observe that the ranking among the spatio-textual objects with $rank > 0$ is defined according to the spatial proximity of the objects with q.

For example, Figure 3.1 shows a road network containing spatio-textual objects annotated with their respective textual descriptions, and can be used as an illustration of what a K*k*NN query retrieves. Assume the keyword k-nearest neighbours query $Q_{KkNN} = \langle q, W, k \rangle$, where $W = \{asian, restaurant\}$ and $k = 1$. In this case, the query point q is the location from where the query starts. When running this query on the

road network in Figure 3.1, object a is returned. The network distance to a from q is 6, and it is the closest object with a description that contains any of the keywords *asian* or *restaurant*.

3.2.2 Keyword range search

$Q_{KR} = \langle q, W, r \rangle = \langle q, w_1 \dots w_n, r \rangle$, given a query point q, a set of query keywords W, and a range r. In the set of spatio-textual objects S, the keyword range search query retrieves the objects p that are within network distance r of q, and whose $rank > 0$, based on the set of keywords W. Hence, $p \in S$ is part of the result set of Q_{KR} if and only if $\Delta(p, q) \leq r$ and $\{\exists w \in W | w \in p.doc\}$.

To illustrate the KR query, let $Q_{KR} = \langle q, W, r \rangle$, where $W = \{asian, restaurant\}$ and $r = 10$. Here q is the point of origin for the query, and r is the range. Using the network and objects in Figure 3.1, the objects that match the query are a and c. The distances from the query point to the objects are 6 and 10, respectively. Even though there is another object within the specified range of 10, object b, this object was not returned as a result of the query since its description does not contain any of the keywords given in W.

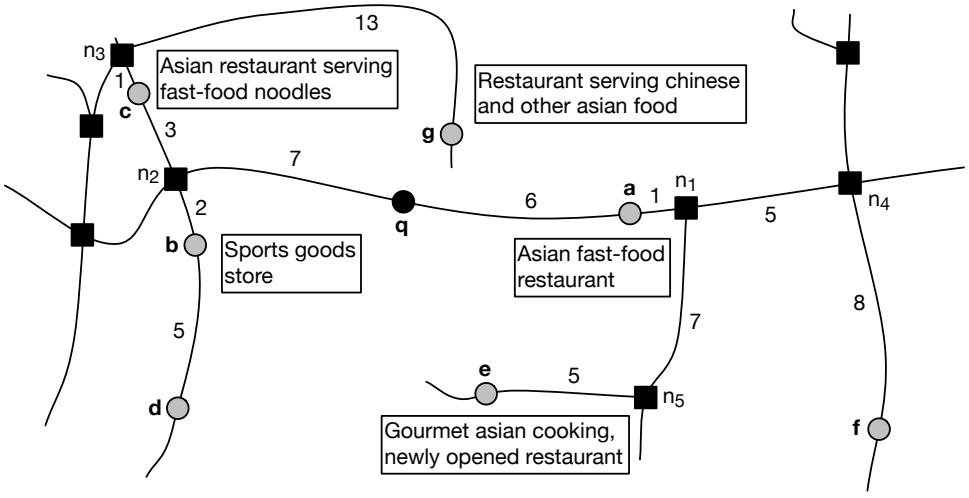


Figure 3.1: An example road network to illustrate what the KkNN and KR queries retrieve. The query point q is located in the middle of the figure as the black point, the spatio-textual objects are spread around the network as the grey points, and the network nodes are the black squares. Each object has a textual description and an identifying label. The labels of the objects, a to g, are in ascending order of the network distance each object has from q; the same for the network nodes, labeled n_1 to n_5. The network distance of some of the segments which do not lead to an object has been omitted for clarity.

3.3 Frameworks for spatial query processing

Currently, there exist frameworks that supports processing of spatial keyword queries on spatial databases, and there exist frameworks that supports processing of spatial queries on road networks. Unfortunately, none of the existing approaches supports processing of spatial keyword queries on road networks. In this section we present two of the current state-of-the-art approaches for spatial query processing. We first introduce the road network framework of Papadias et al. (2003), Section 3.3.1, whereas the hybrid index of Cong et al. (2009) is presented in Section 3.3.2.

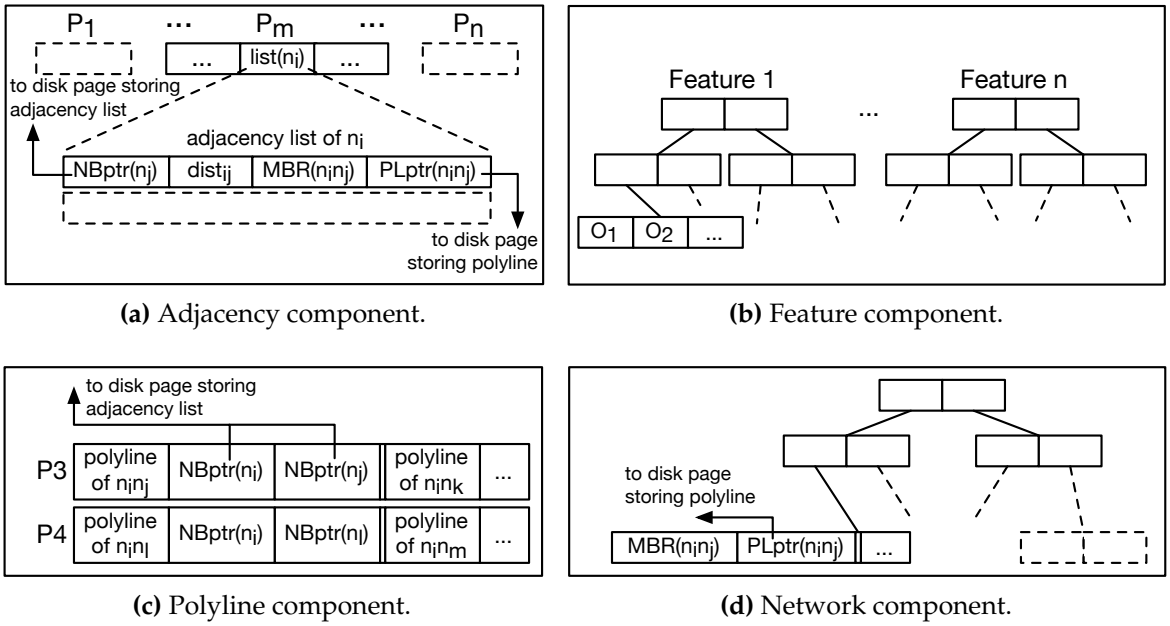
3.3.1 Road network architecture

To the best of our knowledge, the state-of-the-art framework for representing road networks were proposed by Papadias et al. (2003). The framework of Papadias et al. is flexible and supports spatial queries such as range and kNN on road networks. In this architecture the spatial entities are separated from the network to obtain a dynamic framework where feature sets, i.e. categories of objects such as restaurants or bookshops, can be added and updated. In addition, the disk-based network representation preserves connectivity and location information of the underlying road network, and supports exact computation of the network distance between two arbitrary points in the network.

The three components of the network architecture are: (1) an adjacency component that captures the network connectivity, (2) a polyline component that stores the polyline representation of each segment in the network, and (3) a network component that indexes the MBRs of the polylines. In addition, the framework has a fourth component, the feature component, for indexing the feature sets. Figure 3.2 presents the contents of the components and how they are connected.

Adjacency component. The main task of the adjacency component in Figure 3.2a is, as mentioned, to keep track of the network connectivity. To achieve this, each node n_i in the network keeps an adjacency list $list(n_i)$ where the entries have the form $\langle NBptr(n_j), dist_{ij}, MBR(n_in_j), PLptr(n_in_j)\rangle$: $NBptr(n_j)$ is a pointer to the disk page where the adjacency list of n_j can be found, the neighbour to n_i; $dist_{ij}$ is the network distance between nodes n_i and n_j; $MBR(n_in_j)$ is the minimum bounding rectangle enclosing network segment n_in_j; $PLptr(n_in_j)$ is a pointer to the disk page where the polyline representation of the network segment is stored. To improve performance, adjacency lists of nodes that are spatially close to each other are stored on the same disk page in order to benefit from access locality, hence reducing I/O, when fetching data from disk.

Polyline component. The polyline component is depicted in Figure 3.2c. Its main task is to store the polyline representation of each segment in the network. A polyline



(a) Adjacency component.

(b) Feature component.

(c) Polyline component.

(d) Network component.

Figure 3.2: The road network architecture of Papadias et al. (2003) for query processing on spatial network databases.

can be represented as a sequence of points, each with its own coordinates in the Euclidean space, so that a curve consists of the line segments connecting each consecutive point. How accurate this representation of the actual curve is all depends on the number of smaller line segments the polyline is comprised of. Many line segments means higher accuracy, but demands more storage, whereas fewer line segments may give an inaccurate curve representation, but at a lower storage cost. Each entry in this component consists of a detailed polyline representation, as well as pointers to the disk pages in the adjacency component where the adjacency lists of the two end-nodes are stored.

Network component. As for the third component in the architecture, the network component of Figure 3.2d, its task is to index the MBRs of the segments in the network. For this purpose the authors use an R-tree which supports queries exploring the spatial properties of the network. Each leaf node in the R-tree contains the MBR of the indexed network segment, as well as a pointer to the disk page where the corresponding polyline representation of the segment is stored. These pointers to the disk pages enable fast access to the polylines after a segment is located during the course of a spatial query.

Feature component. Separated from the network architecture described above there is the feature component in Figure 3.2b, where each feature set such as hotels, restaurants, and shops are indexed in an R-tree. Such a separation of spatial entities and the network architecture comes with several advantages: (1) conventional Euclidean queries which do not require the network can be processed efficiently using the R-trees, (2) queries combining the network and Euclidean aspects are supported, (3) updates in each feature set can be handled independently, (4) new feature sets can be added to

the system easily, and (5) specific optimizations can be applied to the network and the feature R-trees individually.

Following is a set of primitive operations that the architecture described above supports:

(1) CHECKENTITY(*seg*, *p*) is a Boolean operation that returns *true* if point *p*, i.e. a spatial entity, lies on segment *seg*; *seg* is also said to *cover p*. This is done by first using the MBR of *seg* as a filtering step to see if the coordinates of *p* actually fits within the bounding rectangle. After the filtering step the detailed polyline representation of the segment has to be used to verify that *p* actually lies on the segment. There is also the possibility of errors here. The polyline representation may not be accurate enough, or the coordinates of the point may be off by some small fraction, resulting in the point *p* not being tied to segment *seg*, even if that is actually so. To account for such inaccuracies the authors suggest to use a distance threshold dT, such that if *p* is within distance dT from *seg* it is assumed to lie on it.

(2) FINDSEGMENT(*p*) returns the segment *seg* that covers point *p*. This is done by running a query on the network R-tree using *p* as an input argument and returning the MBR of *seg* if it is located. Should *p* be present in several segments in, for example, a road junction, the first segment that is located is returned. There is also the problem that *p* may not lie on any segment, due to incomplete information in the network. Should this happen, then *p* could be placed on the nearest segment, or discarded, depending on the application.

(3) FINDENTITIES(*seg*): this operation returns all the entities that lies on segment *seg*. The MBR of *seg* is used in a query on the feature R-tree, which retrieves all the spatial entities that are located within the MBR of *seg*. This yields a number of possible false hits, so the polyline of *seg* has to be used in order to check every entity and verify that it does indeed lie on *seg*.

(4) COMPUTEND(p_1, p_2) has the task of computing the actual network distance between two arbitrary points p_1 and p_2 in the network. Papadias et al. have adapted Dijkstra's algorithm (Dijkstra, 1959) for this purpose. It is simple, efficient, and exhibits access locality, reducing the number of page faults during the retrieval of adjacency lists. The problem with this algorithm is that it assumes that the source and the target both are vertices in the network, nodes between two segments, but in this setting an entity, or a point, can be found on an arbitrary location on a segment. Thus, the algorithm has to be adapted to consider this.

A *k*NN query processing example. Here, we present an example to illustrate how a *k*NN query is processed in this framework using the network and the objects of Figure 3.1. Assume that all the objects in the figure belong to one feature set, i.e. any object encountered during query processing is considered an answer to the query. Given a *k*NN query where $k = 3$ and q is the query location, the first step is to retrieve the network segment where q is located, edge (n_1, n_2), from the network R-tree in Figure 3.2d. This is done by executing the FINDSEGMENT operation with q as input. Next, we run the FINDENTITIES operation to find $O(n_1, n_2)$, i.e. the objects on edge (n_1, n_2),

by searching the feature R-tree in Figure 3.2b for objects within the bounding rectangle of (n_1, n_2). As part of this operation we discover object a, and after a refinement step where the polyline representation of segment (n_1, n_2) is used in the CHECKENTITY operation to verify that a lies on the edge, we add the object to a tentative result set $Res = \{a\}$. Onwards, the nearest end-node to q, node n_1, is expanded, and we run successive queries on the feature R-tree to search for objects on edge (n_1, n_4) and edge (n_1, n_5). No objects are found on these two segments, and we continue with the expansion of node n_2. After three more queries on the feature R-tree to search for objects on the neighbouring segments to n_2, we have discovered three additional objects, namely, b, c, and d. The tentative result set now consists of four objects, $Res = \{a, c, d, b\}$. We sort the objects on network distance, get $Res = \{a, b, c, d\}$, and discard object d since the query only specified the three nearest objects, and return $Res = \{a, b, c\}$ as an answer to the kNN query.

Although this framework provides a flexible road network architecture and is able to compute the exact network distance between two arbitrary points, it does, unfortunately, not support keyword-based queries. The framework is thus not directly applicable to our problem.

3.3.2 IR-tree

To the best of our knowledge, the state-of-the-art approach for processing keyword queries on spatial databases were proposed by Cong et al. (2009). The indexing framework they propose supports processing of keyword queries on spatial data by integrating the inverted file for text retrieval and the R-tree for spatial proximity querying in a hybrid structure they call the *Inverted file R-tree*, or IR-tree. The IR-tree is an R-tree where each node has a reference to an inverted file for the objects present in the children of the node. This means that each node in the IR-tree records a summary of both the textual content and the spatial information of the objects in the subtree rooted at any given node, enabling a query to prune the search space using both textual and spatial data during tree traversal.

Each leaf node in the IR-tree contains a number of entries of the form $(O.id, O.rectangle, O.doc)$, where $O.id$ is a pointer to the disk page where the data object is stored, $O.rectangle$ is the minimum bounding rectangle that spatially encloses the object, and $O.docid$ is a reference to the document containing the textual description of object. In addition, each leaf node also contains a pointer to an inverted file for the documents of the objects. This inverted file keeps a vocabulary of distinct terms derived from the document collection, with a pointer from each term to a postings list of tuples; each tuple contains an object id and the frequency of the term in the document describing the object.

The intermediary nodes contain a number of entries of the form *(Node.id, Node.rectangle, Node.pseudodocid)*, where *Node.id* is a pointer to the child node of the entry, *Node.rectangle* is the MBR that spatially encloses the MBRs of all the entries in the child node, and *Node.pseudodocid* is a reference to a pseudo-document that is an aggregated representation of all the documents of the entries in the child node. As the leaf nodes, the

intermediary nodes contain a pointer to an inverted file. The content of this inverted file is slightly different in that it only keeps an index of whether a given term can be found in the subtree.

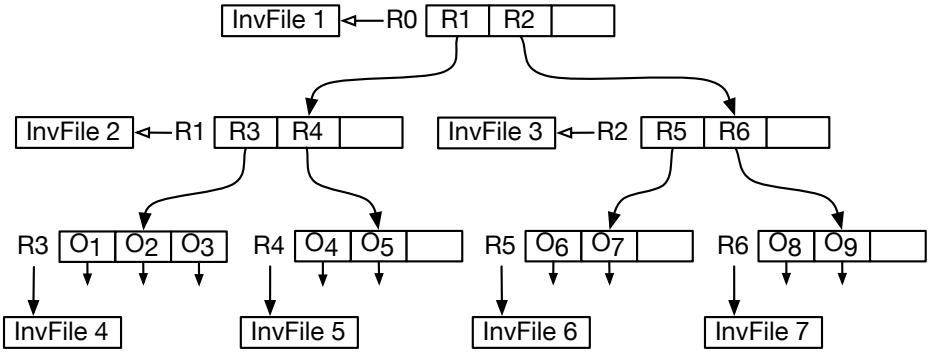


Figure 3.3: An example of how an IR-tree can be structured.

Figure 3.3 shows how an IR-tree can be structured, using an arbitrary set of data objects. Each node in the IR-tree contains a summary of the location information and the textual information of all the objects in the subtree rooted at the node. Thus, *InvFile 4* contains an index of the textual information stored in object O_1, O_2 and O_3, while *InvFile 5* indexes the information stored in object O_4 and O_5. In the intermediary node R_1, on the other hand, *InvFile 2* contains an index of the aggregated textual information of the objects in both nodes R_3 and R_4. The same idea applies to *InvFile 1* and *InvFile 3*, and so on.

With the IR-tree it is possible to filter out objects, and prune whole subtrees, at an early stage in the query processing. If the set of query keywords is not present in the inverted file at any given node in the IR-tree, we know that none of the spatio-textual objects located in the subtree rooted at that node can be an answer to the query, and can thus prune the subtree from the rest of the search space. This is an improvement compared to the naive approach of keeping two separate indexing structures — one for textual information and one for spatial locality — where the objects that match the spatial part of the query are found first, and subsequently filtered based on their textual information and the textual part of the query.

A range query processing example. Assume a range query specifying a range of $r = 5$ from a query point q, and a set of query keywords $W = \{restaurant\}$. In Figure 3.4 we have taken the spatio-textual objects from Figure 3.1 and indexed them in an IR-tree, defined by the hierarchy of MBRs shown in the figure. We have also added a range overlay that extends from q to illustrate which objects are located within the specified range r, and that will be examined as part of the query. The IR-tree constructed based on the spatio-textual objects in the figure consists of a root node, the three leaf nodes R_1, R_2, and R_3, and an inverted file for each of the nodes.

The range query starts at the root node and recursively visits the entries that may contain spatio-textual objects matching the query. First, the query consults the inverted file of the root node to determine which of its children contains objects whose description matches the query keywords. All the entries of the root node, i.e. the rectangles R_1,

R_2, and R_3, encloses at least one object whose description contains the query keyword *restaurant*. Second, the step is to determine which of the rectangles that intersect with the query area. As can be seen from the figure, both rectangles R_1 and R_2 intersect the query area, and are subsequently visited. Rectangle R_3 is outside the query range, and is thus pruned from the rest of the search space. We examine the inverted file of R_1 and discover that spatio-textual object *c* matches the query keyword. Next, we traverse the entries of R_1 to locate object *c*, and we subsequently compare its location with the query area, finding that it is outside the specified query range. The query repeats the process for rectangle R_2; we consult the inverted file and discover that both objects *a* and *g* contains the term *restaurant*. After comparing their location with the query range we find that *a* is outside the range, and *g* is located within. Concluding the range query, spatio-textual object *g* is returned as an answer, since it is within a range $r = 5$ of *q*, and its description contains the keyword *restaurant*.

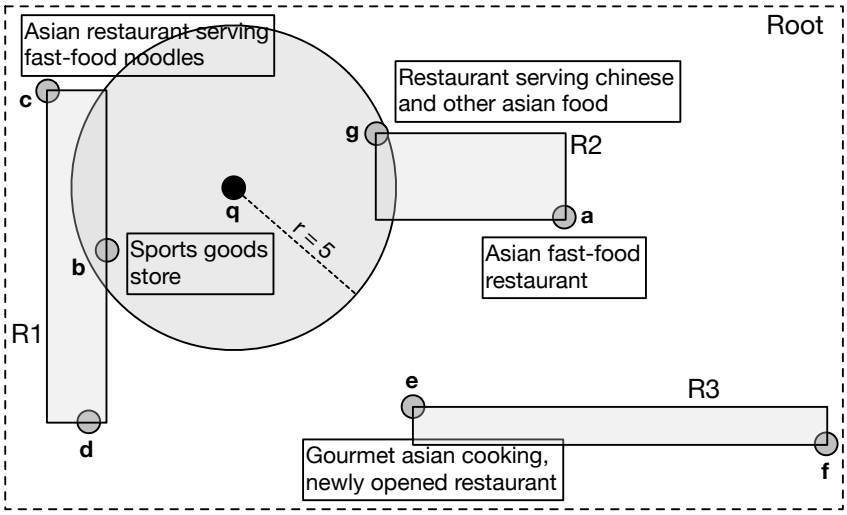


Figure 3.4: A modified version of Figure 3.1 where the network has been removed and the spatio-textual objects have been kept. We have indexed the spatio-textual objects in an IR-tree where the MBRs in the figure indicates how the objects are placed together in the hierarchy. The IR-tree thus consists of a root node and three leaf nodes. A range overlay has also been added to illustrate what a range query with range $r = 5$ retrieves when performed on the IR-tree.

The DIR-tree. Cong et al. have also proposed an extension of the IR-tree that incorporates document similarity when creating the tree, giving an index structure they call the *Document similarity enhanced Inverted file R-tree*, or DIR-tree. Where the IR-tree, like the R-tree, aims to minimize the area of the minimum bounding rectangles of the intermediary nodes when constructing the tree, the DIR-tree takes both the spatial and the textual information into account during tree construction. By minimizing the areas of the bounding rectangles of the intermediary nodes, and by maximizing the text similarity between the documents of the intermediary nodes, the DIR-tree aims to be an index structure that is optimized for both the textual as well as the spatial part of a query (Cong et al., 2009). Subsequently, the DIR-tree can be reduced to an IR-tree if the document similarity factor is set to zero during tree construction, thus optimizing only the area of the enclosing rectangles of the intermediary nodes.

In the rest of this thesis the terms IR-tree and DIR-tree are used interchangeably, though it may be assumed that the DIR-tree is the structure used during later implementation and experiments.

The IR-tree of Cong et al. is a good framework for processing keyword queries on spatial databases, but unfortunately it does not support spatial keyword processing on road networks. Hence, the IR-tree is not applicable to our problem in its original form.

Chapter 4

Baseline Approach

In this chapter we present our baseline solution to the challenging problem of processing keyword-based queries on spatial network databases such as common road networks. By combining two state-of-the-art architectures proposed by Papadias et al. (2003) and Cong et al. (2009), respectively, the baseline framework in this solution supports retrieval of spatio-textual objects using keyword-based *k*-nearest neighbour and range queries. The road network architecture of Papadias et al. provides support for operations on spatial network databases, while the hybrid index structure of Cong et al., the IR-tree, provides support for processing keyword queries on spatial data.

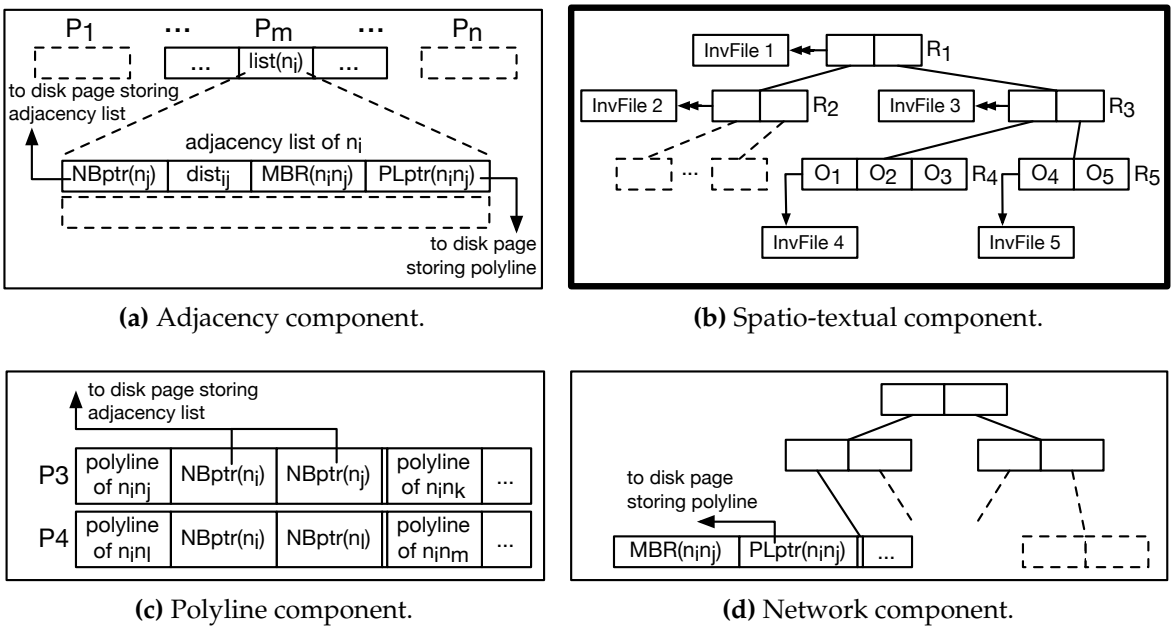
The next section, Section 4.1, will go further into the architecture and the operations of the framework, whereas Section 4.2 will describe the algorithms used to process the keyword-based *k*-nearest neighbour and range queries.

4.1 Basic keyword-based road network architecture

In this section we present a basic approach for evaluating keyword queries on spatial networks. We combine the architecture of Papadias et al. (2003) and the IR-tree of Cong et al. (2009) to obtain our baseline architecture for processing keyword-based queries on spatial network databases.

Figure 4.1 presents the different components in the architecture and how they are combined. The three components of Papadias et al., (a), (c) and (d), constitutes the parts necessary for operations on the road network, and the IR-tree of Cong et al. in (b) indexes the spatio-textual objects and supports spatial operations. As described previously in the Preliminaries chapter, the adjacency component in (a) captures the network connectivity, the polyline component in (c) stores the polyline representation of each network segment, and the network component in (d) indexes the MBRs of the polylines.

The primitive operation FINDENTITIES(*seg*) proposed by Papadias et al. (2003) does not support the operation of retrieving spatio-textual objects. Therefore, we developed a new function, namely the FINDSPATIOTEXTUALOBJECTS(*node,seg,W*) operation (Al-



(a) Adjacency component.

(b) Spatio-textual component.

(c) Polyline component.

(d) Network component.

Figure 4.1: Basic architecture for processing spatial keyword queries on road networks.

gorithm 4.1), which takes text into account when retrieving the objects that a segment covers.

FINDSPATIOTEXTUALOBJECTS(*node*,*seg*,*W*) takes the root of the IR-tree, a segment *seg* and a set of keywords *W* as input, and returns all the spatio-textual objects covered by the segment that matches any of the keywords in *W*. The MBR of *seg* and the keywords in *W* is used in a query on the IR-tree in the spatio-textual component in Figure 4.1b. Thus, the objects located within the MBR of *seg*, and whose description matches any of the keywords in *W*, are returned as a result of the query. This yields a number of false positives, and all the spatio-textual objects have to go through a refinement step where the polyline representation of *seg* is used to find the objects that are actually covered by *seg*, and not just present in the MBR.

Algorithm 4.1 presents the pseudocode of the operation. The root node of the IR-tree is the entry point of the algorithm, and it is sent as input together with *seg* and the set of keywords *W*. The algorithm starts by initializing the result set of spatio-textual objects as empty (line 3). Subsequently, it checks, for each intermediary node in the tree (lines 5-10), whether the MBR of *seg* overlaps the MBR of the entries in the node, simultaneously checking if any of the query keywords is present in the pseudo-document of the node (line 7). If both is the case, it means that there may exist a spatio-textual object in the subtree that matches the query, and the result set is updated with the results from a recursive call to the algorithm, sending the current intermediary node as input (line 8). When encountering a leaf node in the tree (lines 12-18), the algorithm checks, for each spatio-textual object in the leaf node, whether the MBR of the object is overlapping with the MBR of the road segment *seg*, as well as checking if any query keyword in *W* is present in the description of the spatio-textual object (line 13). If this is the case, it must be verified that the spatio-textual object indeed is present on the road segment. This is done by the CHECKENTITY(*seg*, *p*) operation

on line 14, subsequently returning the spatio-textual object as a result if it lies on *seg* (line 15). Finally, the set of spatio-textual objects that lies on the road segment *seg*, matching any keyword in W, is returned as a result, and the algorithm terminates (line 20).

In short, the basic architecture described in this section comprise the road network representation of our baseline framework for processing spatial keyword queries on road networks. In the next section we will discuss the algorithms supporting the KkNN and KR queries.

Algorithm 4.1 FINDSPATIOTEXTUALOBJECTS($node, seg, W$)

```
1: Input: the root node of the IR-tree, a road segment, a set of keywords
2: Output: the set of spatio-textual objects covered by seg, matching any keyword in W
3: result ← ∅
4: if node is an intermediary node then
5:    for each entry E_i in node do
6:       N_i ← node pointed to by E_i
7:       if (seg.MBR ∩ N_i.MBR) ≠ ∅ and W ∈ N_i.pseudodoc then
8:          result ← result ∪ FINDSPATIOTEXTUALOBJECTS(N_i, seg, W)
9:       end if
10:   end for
11: else                                                    //node is a leaf node
12:   for each object O_i in node do
13:      if (seg.MBR ∩ O_i.MBR) ≠ ∅ and W ∈ O_i.doc then
14:         if CHECKENTITY(seg, O_i) then
15:            return O_i
16:         end if
17:      end if
18:   end for
19: end if
20: return result
```

4.2 Query processing

To process the KkNN and KR queries on the baseline framework, we adopted two algorithms proposed by Papadias et al. (2003), namely the *Incremental Network Expansion* (INE*) and *Range Network Expansion* (RNE*) algorithms, both based on the network expansion framework. Network expansion performs query processing directly on the network, starting from one location and examines the nodes in the network as they are encountered.

In the following, Section 4.2.1 presents the INE* algorithm for processing keyword-based k-nearest neighbour queries, while Section 4.2.2 presents the RNE* algorithm for processing keyword-based range queries.

4.2.1 Basic keyword-based nearest neighbour algorithm

INE* (Algorithm 4.2) operates by expanding the network node by node, starting from the query point q, and examining the spatio-textual objects in the order they are encountered until the k nearest neighbours are found.

First, the segment that covers q is located using the primitive operation FINDSEGMENT(p) (line 4), and the nodes terminating the segment are added to a priority queue sorted on their network distance to q. Second, the spatio-textual objects that the segment covers are found using FINDSPATIOTEXTUALOBJECTS(*node*,*seg*,*W*) (line 5). Third, the first node in the priority queue is expanded (line 9), its neighbours are added to the priority queue (line 15), and the segment is examined for spatio-textual objects matching the query (line 12). This process continues until a node is expanded that has a larger network distance to q than the current max distance, d_{Nmax}, the network distance from q to the k-th spatio-textual object, or until the nodes in the network are exhausted.

Algorithm 4.2 shows the pseudocode of Incremental Network Expansion. As input the algorithm takes the query point q, the number of nearest neighbours k, and the set of keywords W specified in the query.

Algorithm 4.2 INE*(q, k, W)

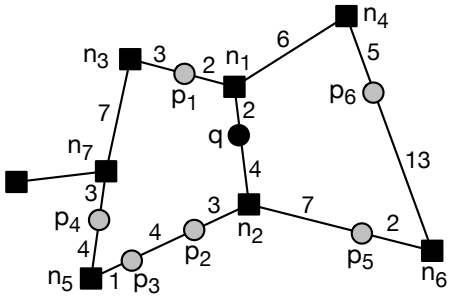
1: **Input:** the query point, the number of nearest neighbours, a set of keywords
2: **Ooutput:** the k nearest spatio-textual objects matching any keyword in W
3: $N_{root} \leftarrow$ root node of the IR-tree
4: $(n_i, n_j) \leftarrow$ FINDSEGMENT(q)
5: $S_{cover} \leftarrow$ FINDSPATIOTEXTUALOBJECTS$(N_{root}, (n_i, n_j), W)$ //S_{cover} is the set of spatio-textual objects covered by (n_i, n_j) whose description matches any keyword in the query
6: $\{p_1, \ldots, p_k\} \leftarrow$ the k nearest spatio-textual objects in S_{cover} sorted in ascending order of their network distance //$p_m, p_{m+1}, \ldots, p_k$ may be $\emptyset$ if S_{cover} contains $< k$ points
7: $d_{Nmax} \leftarrow d_N(q, p_k)$ //if $p_k = \emptyset, d_{Nmax} \leftarrow \infty$
8: $Q \leftarrow <(n_i, d_N(q, n_i)), (n_j, d_N(q, n_j))>$ //sorted on d_N
9: de-queue the node n in Q with the smallest $d_N(q, n)$
10: **while** $d_N(q, n) < d_{Nmax}$ **do**
11: **for** each non-visited adjacent node n_x of n **do**
12: $S_{cover} \leftarrow$ FINDSPATIOTEXTUALOBJECTS$(N_{root}, (n_x, n), W)$
13: update $\{p_1, \ldots, p_k\}$ from $\{p_1, \ldots, p_k\} \cup S_{cover}$
14: $d_{Nmax} \leftarrow d_N(q, p_k)$
15: en-queue $(n_x, d_N(q, n_x))$
16: **end for**
17: de-queue the next node n in Q
18: **end while**

Figure 4.2a shows a modelling graph of a road network that can be used in an example to illustrate the INE* algorithm. The grey points represent the spatio-textual objects, their description listed in Figure 4.2b, the black squares as the nodes in the modelling graph, and the black point as the query point q.

Assume the keyword k-nearest neighbours query $Q_{KkNN} = \langle q, W, k \rangle = \langle q, bakery, 1 \rangle$, i.e. find the nearest spatio-textual object to the query point q that has the keyword *bak-*

ery in its description. The INE* algorithm first retrieves the segment (n_1, n_2) that covers q, using the location of q as input to a query on the network R-tree in Figure 4.1d. A query is subsequently run on the IR-tree with the MBR of (n_1, n_2) as input, but since there are no spatio-textual objects present on this segment, nothing is returned from the IR-tree. Next, the closest node to q, node n_1, is expanded and the other endpoint, node n_2, is added to a priority queue $Q = \{(n_2, 4)\}$. Additional queries are run on the IR-tree for segments (n_1, n_3) and (n_1, n_4), but no matching spatio-textual objects are found, and node n_3 and n_4 are subsequently added to the priority queue, $Q = \{(n_2, 4), (n_3, 7), (n_4, 8)\}$. The algorithm expands node n_2, runs a query on the IR-tree using segment (n_2, n_5), and discovers the matching object p_3 that has a network distance to q of $d_N(q, p_3) = 11$. The two non-visited neighbouring nodes of n_2, node n_5 and n_6, are added to the queue, after which $Q = \{(n_3, 7), (n_4, 8), (n_5, 12), (n_6, 13)\}$. Spatio-textual object p_3 is now a candidate for an answer to the Q_{KkNN} query, and its network distance to q provides a bound to limit the search space, $d_{Nmax} = d_N(q, p_3) = 11$.

In the subsequent steps n_3 and n_4 are expanded, but after successive queries on the IR-tree using segments (n_3, n_7) and (n_4, n_6), no spatio-textual objects matching the query are found, and we get $Q = \{(n_5, 12), (n_6, 13), (n_7, 14), (n_6, 26)\}$. Since the next node in the queue has a larger network distance to q than the current d_{Nmax} the algorithm terminates, and spatio-textual object p_3 is returned as an answer to the keyword k-nearest neighbour query.



(a) The example network.

Object	Description
p1	shop, grocery, flowers
p2	post office
p3	bakery, bread, cake
p4	restaurant, mexican
p5	restaurant, sushi
p6	pub, football

(b) The spatio-textual objects and their descriptions.

Figure 4.2: An example illustrating the INE* and RNE* algorithms. The grey points represent the spatio-textual objects, the black point is the query point q, the black squares indicate the network nodes, and the values on the edges is an arbitrary network distance between the nodes and the objects.

4.2.2 Basic keyword-based range algorithm

The RNE* algorithm (Algorithm 4.3) returns all the spatio-textual objects within a given network distance r from the query point q, that has any of the query keywords in their description. RNE* starts by collecting the qualifying road segments, i.e. the segments that are within network distance r of q, and may contain spatio-textual objects that matches our query. The segments are found using the same network expansion

technique as in INE*, with the difference that this part of the algorithm terminates when we have found all the segments that are within the given distance of the query point, and we are not retrieving the spatio-textual objects during the expansion.

When the qualifying segments have been found, QS is used in a query on the IR-tree in the spatio-textual component. The spatio-textual objects that falls on any of the segments in QS, and whose description matches any of the keywords in W, are subsequently added to the result set (line 9) and returned when the query finishes. Algorithm 4.3 presents the pseudocode for Range Network Expansion. The initial arguments to the algorithm is the root of the IR-tree, the set of qualifying segments QS, and the set of keywords W.

Algorithm 4.3 RNE*($node, QS, W$)

```
1:  Input: the root node of the IR-tree, a set of segments that lies within network distance r of
    the query point q, a set of keywords
2:  Output: the spatio-textual objects within network distance r, matching any keyword in W
3:  result ← ∅
4:  if node is an intermediary node then
5:     compute QS_i for each entry E_i in node
6:     for each entry E_i in node do
7:        N_i ← node pointed to by E_i
8:        if QS_i ≠ ∅ and W ∈ N_i.pseudodoc then
9:           result ← result ∪ RNE(N_i, QS_i, W)
10:       end if
11:    end for
12: else                                                    //node is a leaf node
13:    result_node ← PLANESWEEP(node.entries, QS_i)
14:    sort result_node to remove duplicates
15:    return result_node
16: end if
17: return result
```

Using Figure 4.2 as an example to illustrate the RNE* algorithm, we let a keyword range search query be $Q_{KR} = \langle q, W, r \rangle = \langle q, shop, 7 \rangle$, i.e. find all spatio-textual objects within a network distance of 7 from the query point q that has the keyword *shop* in their description.

The RNE* algorithm starts by collecting the set of qualifying segments QS, i.e. all the network segments that are within a network distance of 7 from q, in the same way as the INE* algorithm expands the network. First, the network segment that covers q is found, its endpoints added to a priority queue $Q = \{(n_1, 2), (n_2, 4)\}$, and the segment itself added to the set of qualifying segments $QS = \{n_1n_2\}$. Node n_1 is then expanded and the two neighbouring nodes are added to the priority queue, making it $Q = \{(n_2, 4), (n_3, 7), (n_4, 8)\}$, and the two network segments connecting the neighbouring nodes are added to $QS = \{n_1n_2, n_1n_3, n_1n_4\}$. Node n_2 is subsequently expanded, adding n_5 and n_6 to the priority queue, $Q = \{(n_3, 7), (n_4, 8), (n_5, 12), (n_6, 13)\}$, and segments n_2n_5 and n_2n_6 to $QS = \{n_1n_2, n_1n_3, n_1n_4, n_2n_5, n_2n_6\}$. The process of generating QS now terminates since any spatio-textual object located on any undiscovered network segment is bound to be outside the given range $r = 7$, because the

network distance to node n_3 is $d_N(q, n_3) = 7 \leq r = 7$.

The next step, after the qualifying segments have been found, is to use QS in a query on the IR-tree, matching the MBR of each segment in QS with the spatio-textual objects indexed in the tree to find all the objects that are located within the specified network distance, and that matches any of the keywords specified in the query. Spatio-textual object p_1 on segment n_1n_3 is the only object within a network distance of 7 that has a textual description that contains the keyword *shop*, and is thus returned as an answer to the keyword range search query.

Chapter 5

Road Network Indexing

For a framework to support efficient retrieval of objects from a spatial network database the underlying structures and algorithms must provide methods to process the network and locate the objects in an efficient manner, yielding low processing costs, and thus low response times for the spatial keyword queries. Unfortunately, the architecture of the baseline approach suffers from different issues which has an impact on the performance of the framework, making query processing inefficient and causing high response times for the keyword-based nearest neighbour and range search queries.

The main issues are: (1) extensive computation to detect and prune false positives when querying the IR-tree searching for spatio-textual objects, and (2) retrieval of an excessive amount of spatio-textual objects during a KkNN query as well as the necessity to sort the objects on network distance. Following is a more detailed description and examples of the issues:

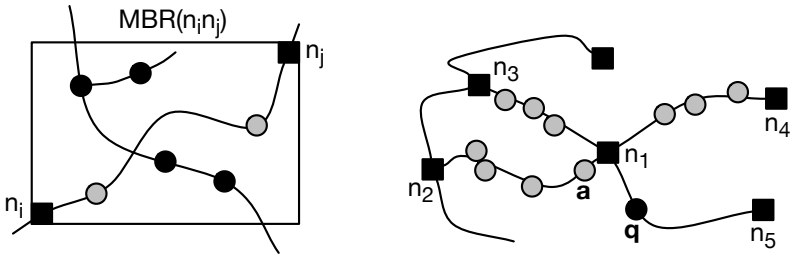
Issue 1. Since the IR-tree is based on indexing the data objects by constructing a hierarchy of possibly overlapping MBRs, the MBR of a road segment may intersect with several intermediary bounding rectangles, which in turn may contain objects that matches the query, but are not present on the road segment in question, i.e. false positives. The IR-tree provides a filtering step to prune non-qualifying objects, where the output of the filtering step has to pass through a refinement step that compares the actual object representation with the exact geometry of the road segment to determine whether a spatio-textual object is actually present on the segment. Such a refinement step incurs an additional computation cost that increases with the density of the spatio-textual objects in the network.

Figure 5.1a illustrates the issue with false positives. For example, assume that we want to retrieve the objects (grey points) on segment (n_i, n_j) in Figure 5.1a. We run a range query on the IR-tree, defined by the MBR that encloses (n_i, n_j), to filter out any point not on the network segment. In this process, the black points will also pass the filtering step as false positives since they intersect with the query MBR, and will thus have to be pruned as part of the refinement step.

Issue 2. A query on the IR-tree retrieves matching spatio-textual objects on a per-segment basis, which means that it has to retrieve all the matching objects present

on a network segment, even though fewer objects are often desired. Furthermore, the objects have to be sorted on network distance since the output from the IR-tree is not ordered, and the excess objects are subsequently discarded to provide a desired answer to the query. For example, if a query specifies k results, and there are n matching objects, $k << n$, then excessive computation is used to sort all the objects on network distance, just to discard the $n - k$ remaining objects.

Figure 5.1b illustrates the excessive retrieval of objects; each grey point represents one spatio-textual object. For example, assume a KkNN query where $k = 1$ and q is the query point, then point a will be the nearest match to q, ignoring the textual descriptions for simplicity. In order to find a, the INE* algorithm requires searching the IR-tree for the objects in segments (n_1, n_2), (n_1, n_3), (n_1, n_4), and (n_1, n_5), filter the objects, and sort them according to network distance. From the sorted list of objects we can extract object a as the answer to the KkNN query, whereas the remaining objects will be discarded.



(a) The MBR that encloses the road segment (n_i, n_j) and the objects located within it.

(b) A simple road network along with spatio-textual objects spread out on the road segments.

Figure 5.1: Examples of the performance related issues with the architecture of the baseline approach.

Due to the issues described above, the baseline approach is costly in terms of both I/Os and execution time. To this end, we propose a novel approach for efficient processing of keyword queries on spatial network databases that solely uses inverted files for indexing the spatio-textual objects, yielding a framework we call *Road Network Indexing* (RNI).

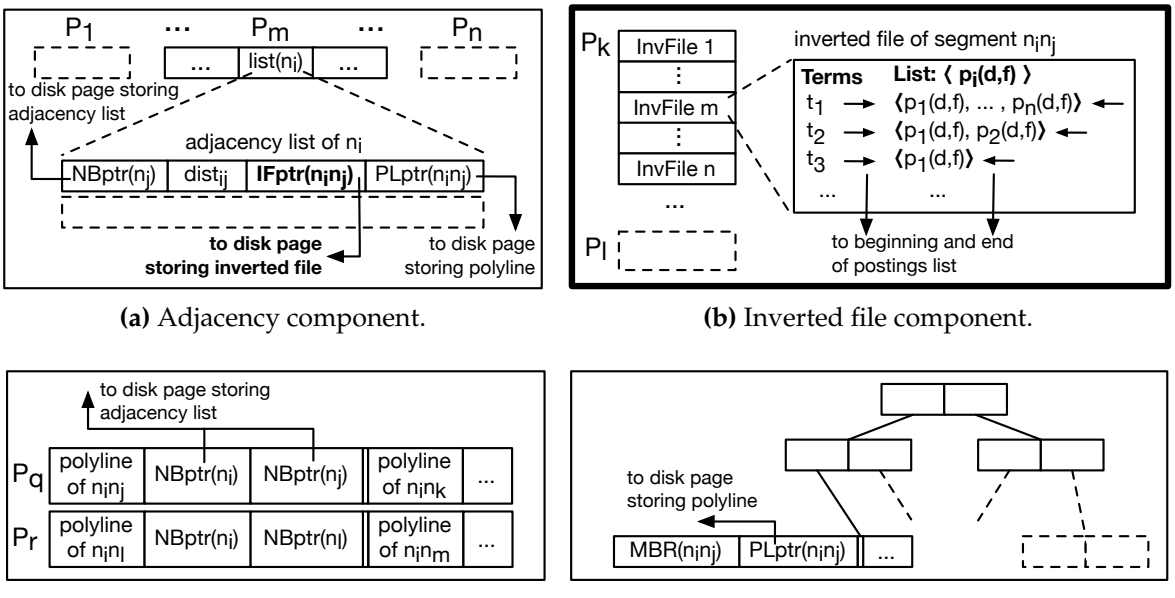
RNI replaces the IR-tree with a set of inverted files, one for each network segment, where each file indexes only the spatio-textual objects covered by the corresponding segment. This design allows for fast query processing since we only have to verify the existence of a query keyword in an inverted file to confirm that there exists a matching object on the network segment. Furthermore, the network distance to an object can be stored in the postings of the inverted file, together with the object reference and term frequency, and can be retrieved efficiently during query processing.

In the rest of the chapter the RNI framework will be explained in more detail, starting with the architecture in Section 5.1, continuing with the query processing and the algorithms in Section 5.2.

5.1 Advanced keyword-based road network architecture

In this section we present the architecture of our proposed RNI framework for efficient evaluation of keyword queries on SNDB. The RNI framework is an improvement of the baseline framework that is modified to support keyword-based spatial queries more efficiently. In RNI we employ a slightly modified version of the baseline network storage scheme, combined with a set of inverted files for indexing the spatio-textual objects, consequently replacing the IR-tree of the baseline framework.

Figure 5.2 presents the components of the RNI architecture, with the changes from the baseline approach marked in bold. The inverted file component in (b) replaces the spatio-textual component as the component used to index the spatio-textual objects. In the adjacency component in (a) we have modified the contents of the entries in the adjacency lists of the network nodes. The MBR of the network segment connecting two neighbouring nodes has been replaced with a pointer to the disk page storing the inverted file that indexes the spatio-textual objects covered by the network segment. Such a pointer allows for fast access to the inverted file so we can retrieve the matching objects, compute their network distance to a query location, and subsequently add them to the answer of a spatial keyword query.



(a) Adjacency component.

(b) Inverted file component.

(c) Polyline component.

(d) Network component.

Figure 5.2: Advanced architecture for processing spatial keyword queries on road networks.

Inverted file component. The inverted file component in Figure 5.2b is responsible for indexing the spatio-textual objects in the network. To accomplish this, we create one inverted file for each segment in the network, where all the objects that are covered by one segment are indexed in the corresponding inverted file. Should an object be covered by several network segments, e.g. in a road intersection, then the object will be indexed in the inverted file of the network segment discovered first during the construction of the component.

In each inverted file, the vocabulary is comprised of the set of terms extracted from the document collection describing the spatio-textual objects on the corresponding network segment. Each term t in the vocabulary points to a postings list of tuples of the form $p(d, f)$, where p is a reference to a spatio-textual object whose textual description contains t, $d = \delta(p, n_i)$ is the network distance from p to node n_i in the segment (n_i, n_j), and f is the frequency of the term t in the document describing p. Furthermore, the tuples in the postings list are sorted on the network distance to node n_i in the segment (n_i, n_j).

As indicated by the two arrows in each entry in the vocabulary, we are storing two pointers from t to the postings list, one to the beginning of the list and one to the end. These pointers allow us to traverse the postings list both ways, starting from either end, and thus having immediate access to the nearest and farthest object per term. This is useful during network traversal when we encounter a network segment where n_j is the first node, i.e. in the segment (n_j, n_i). In this case the last object in a postings lists will be the nearest, and by following the end pointer we can access it directly, without having to traverse the whole postings list first. The network distance to the objects starting from node n_j in segment (n_i, n_j) is easily computed by subtracting the stored distance d from the total distance of the network segment.

5.2 Query processing

In the Preliminaries chapter we outlined two keyword-based spatial queries, namely the KkNN and KR queries, for nearest neighbour and range querying on spatial network databases. In order to process the two keyword-based spatial queries on the RNI framework we have developed two algorithms that exploits the network storage scheme and the inverted files in an efficient manner to quickly locate the spatio-textual objects and in the process lower the amount of I/Os and computation necessary to retrieve the objects.

In this section we present the *Incremental Inverted Index k-Nearest Neighbours* algorithm (I3kNN) and the *Incremental Inverted Index Range Search* algorithm (I3RS). Both algorithms are based on incremental expansion of the network, similarly to Dijkstra's shortest path algorithm (Dijkstra, 1959).

The I3kNN algorithm is explained in more detail in the following section, Section 5.2.1, along with an example to show how it operates, whereas the I3RS algorithm with an example is presented in Section 5.2.2.

5.2.1 Advanced keyword-based nearest neighbour algorithm

The I3kNN algorithm (Algorithm 5.1) supports the KkNN query described in Section 3.2.1 in the Preliminaries chapter, hence, the output of the algorithm is the k nearest spatio-textual objects to a query point q whose textual description contains any of the keywords we specify in the query.

I3kNN operates by expanding the network, segment by segment, while examining the inverted files of the segments as they are encountered to look for matching spatio-textual objects; a process that continues until the desired number of objects have been found, or the network is exhausted. The first step of the expansion process is to retrieve the network segment covering q from the R-tree in the network component. Then, the algorithm follows the pointer to the inverted file of the segment to check whether there exists any matching spatio-textual objects.

In the inverted file we traverse the vocabulary to find the terms that match any of the query keywords. If there are no matching terms we know that there are no matching spatio-textual objects, and we can discard the inverted file and continue the expansion process. If there are matching terms, the algorithm follows a pointer from each matching term to its postings lists to find the nearest matching spatio-textual object p. After having retrieved the nearest end-node n from the adjacency component, we compare the distance from q to p with the distance from q to n.

If the distance is shorter to n there might be spatio-textual objects farther in the network that is closer to q than p, and we put the inverted file on hold by adding to a priority queue a tuple $(p, \Delta(q,p), *IF(n,n'))$ consisting of p, the distance from q to p, and a pointer to the next nearest object in the inverted file. On the other hand, if p is closer to q than n, we add p to a result set together with its network distance to the query location.

Algorithm 5.1 I3kNN(q,k,W)

Input: Query point (q), Number of results (k), Set of keywords (W)
Output: Result set (Res) of spatio-textual objects matching any keyword in W
Priority queue $(\mathcal{H}, \mathcal{Q})$
$\mathcal{H}, \mathcal{Q}, Res \leftarrow \emptyset$
$(n_i, n_j) \leftarrow$ FINDSEGMENT(q)
$\mathcal{H} \leftarrow \langle (n_i, \Delta(q,n_i)), (n_j, \Delta(q,n_j)) \rangle$
$p \leftarrow$ GETCANDIDATE$(IF(n_i,n_j), W)$ *//Get first object matching any keyword in W*
$\mathcal{Q} \leftarrow \langle (p, \Delta(q,p), *IF(n_i,n_j)) \rangle$ *//$*IF$ points to next matching object in $IF(n_i,n_j)$*
while $\mathcal{H} \neq \emptyset$ **and** $|Res| < k$ **do**
  $nDist \leftarrow$ **peek**$(\mathcal{H})$ //Get distance to nearest node
  $pDist \leftarrow$ **peek**$(\mathcal{Q})$ //Get distance to nearest object
  if $nDist < pDist$ **then**
    $(n, \Delta(q,n)) \leftarrow$ **dequeue**$(\mathcal{H})$
    for each non-visited adjacent node n' of n **do**
      enqueue$(\mathcal{H}, (n', \Delta(q,n')))$
      enqueue$(\mathcal{Q}, (p, \Delta(q,p), *IF(n,n')))$
    end for
  else
    $Res \leftarrow Res \cup p$
    update$(\mathcal{Q}, (p, \Delta(q,p), *IF(n,n')))$ *//Update tuple with next object from $*IF$, discard if empty*
  end if
end while
return Res

I3kNN then expands the nearest network node n and visits each adjacent network segment to consult the corresponding inverted files, subsequently adding tuples to the priority queue. The network distance to the spatio-textual objects can easily be computed by adding the distance from q to n to the distance that is stored together with the object reference in each posting. The process is repeated, except that we now compare the distance to the nearest network node with the distance to the object of the first entry in the priority queue, i.e. the currently nearest spatio-textual object, either adding the next nearest object to the result set, or expanding the network further. This process goes on until we find k spatio-textual objects, or until the network is exhausted.

Algorithm 5.1 presents the pseudocode of I3kNN. The input to the algorithm is the query point q, the number of results k, and a set of query keywords W. I3kNN proceeds as follows. After a local initialization of the result set and the two priority queues $\mathcal{H}$ and $\mathcal{Q}$ — $\mathcal{H}$ is keeping track of the network nodes, while $\mathcal{Q}$ is keeping track of the spatio-textual objects — we search the network R-tree looking for edge (n_i, n_j), the road segment where q is located (line 5). Next, we calculate the distance from q to each of the end-nodes of segment (n_i, n_j), and enqueue a tuple to $\mathcal{H}$ consisting of a pointer to the node, and the distance from q, e.g. $(n_i, \Delta(q, n_i))$ (line 6).

In line 7 and 8 we construct a tuple $(p, \Delta(q, p), *IF(n_i, n_j))$ and add it to $\mathcal{Q}$: p is the nearest matching spatio-textual object on the segment; $\Delta(q, p)$ is the distance from the query point q to p; $*IF(n_i, n_j)$ is a pointer to the next matching object in the inverted file. The inverted file is accessed by following the pointer stored in the adjacency list of n_i. Furthermore, the tuples in $\mathcal{Q}$ are ordered on the network distance from q to the spatio-textual object present in each tuple, independent of the road segment each tuple comes from.

The network expansion process and the incremental discovery of spatio-textual objects goes on in line 9 to 22; it loops until the network is exhausted, or until we have found the desired number of results.

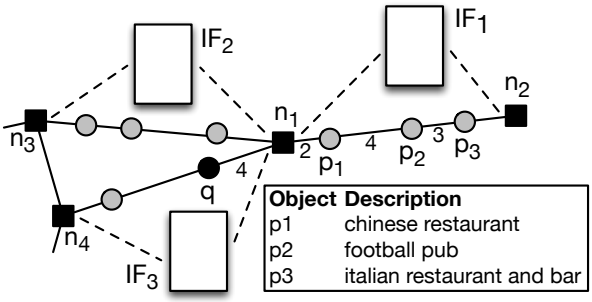
In each round we first get the distance from q to the currently nearest network node n in $\mathcal{H}$, and from q to the nearest spatio-textual object p in $\mathcal{Q}$ (line 10 and 11). If the distance to n is shortest we know that we might find a closer object if we expand the network further, and if it is shortest to p then p is a result candidate. In the latter case we remove p from $\mathcal{Q}$ and add it to the result set *Res*, subsequently updating the tuple with the object pointed to by $*IF(n, n')$, as well as incrementing the pointer and re-ordering the queue (line 19 and 20), alternatively discarding the tuple if there are no more objects in $IF(n, n')$. In the former case we traverse the adjacency list of the network node n, add its adjacent nodes to $\mathcal{H}$, and construct one object-distance-pointer tuple for each adjacent network segment and add the tuples to $\mathcal{Q}$ (lines 14-17).

The I3kNN algorithm terminates in line 23 by returning the result set *Res* which contains the k nearest spatio-textual objects that matches any of the keywords W in the query.

Note that since we always expand the network further if there is a network node with shorter distance to q than the current nearest object pointed to in $\mathcal{Q}$, we are guaranteed that the result set contains the k nearest matching objects. Consequently, it is not necessary to sort the result set before returning it as an answer to the query.

kNN query processing example

Figure 5.3a presents a road network containing spatio-textual objects p as the grey points, with their description listed in the expanded view of inverted file IF_1 in Figure 5.3b, the query point q as the black point, and the distance between the objects annotated on the edges. Assume the keyword k-nearest neighbour query $Q_{KkNN} = \langle q, W, k \rangle = \langle q, restaurant, 1 \rangle$. For simplicity, assume that the objects without a label are without a textual description, and consequently do not match the query. The I3kNN algorithm first retrieves segment (n_1, n_4) using q as input to a query on the network R-tree in Figure 5.2d. Next, the distance from q to both of the end-nodes is computed. The algorithm consults IF_3, traverses the vocabulary of the inverted file only to find that the query keyword *restaurant* does not exist here, and then discards the inverted file since the network segment does not contain any matching objects. Node n_1 is closest to q, and is subsequently expanded. The same process is performed on segment (n_1, n_3), the vocabulary of inverted file IF_2 is searched, but the query keyword is not present here, and we conclude that there do not exist any objects on that segment. Next, the algorithm expands segment (n_1, n_2). In inverted file IF_1 the query keyword *restaurant* is found as a term in the vocabulary, and we subsequently retrieve the postings list of the term. By following the pointer from the term to the beginning of the postings list, we find object p_1. Since p_1 is the first object in the postings list, it is thus the nearest object to q, and we add it to the result set, subsequently terminating the query since we found the nearest neighbour to q.



(a) A graph model of a road network along with spatio-textual objects p spread around the road segments. IF_i indicates the inverted files used to index the objects on each segment (n_i, n_j).

IF_1

Terms	List: ⟨p(d, $f_{d,t}$)⟩
restaurant	p_1(2,1), p_3(9,1)
chinese	p_1(2,1)
italian	p_3(9,1)
bar	p_3(9,1)
pub	p_2(6,1)
football	p_2 (6,1)

(b) An expanded view of inverted file IF_1, displaying the indexed content of the spatio-textual objects present on segment (n_1, n_2).

Figure 5.3: An example network to illustrate the road network indexing approach.

5.2.2 Advanced keyword-based range algorithm

To enable keyword-based range queries on the RNI framework we have developed the I3RS algorithm (Algorithm 5.2). I3RS supports the KR query we presented in Section 3.2.2 the Preliminaries chapter, and followingly outputs the spatio-textual objects that are within range r of the specified query point q, and whose textual description contains any of the given query keywords.

In the same fashion as the I3kNN algorithm, the expansion process successively expands the nearest network node to the query point q, consults the inverted files of the encountered adjacent segments, and adds to the query answer all the spatio-textual objects that are within range r of q, and whose description contains any of the query keywords. The process continues until we have expanded all of the network within the specified query range.

The I3RS algorithm starts from the query location and adds the two end-nodes of the road segment, together with their network distance to q, to a priority queue, but differently from the I3kNN algorithm we do not keep a priority queue for the inverted files and the spatio-textual objects. The reason for this is that we are interested in all objects within range r, and when we consult the inverted file of a road segment during the expansion process, we get all objects from the postings lists of the matching terms, not just the nearest one.

To limit the search to the defined range r we keep a variable *minDist* that holds the distance to the nearest non-visited network node After we have visited all the adjacent road segments of a given node, and subsequently added the adjacent node to the priority queue, we update *minDist* with the distance to the first element of the queue. Consequently, when we assign a value to *minDist* that is larger than r, we know that no other object can be within the query range, and the algorithm can return the query answer and terminate.

Algorithm 5.2 I3RS(q, r, W)

```
1: Input: Query point (q), Range (r), Set of keywords (W)
2: Output: Result set (Res) of spatio-textual objects within range r matching any keyword in W
3: Priority queue (H)
4: H, Res = ∅
5: (n_i, n_j) ← FINDSEGMENT(q)
6: H ← ⟨(n_i, Δ(q, n_i)), (n_j, Δ(q, n_j))⟩
7: minDist ← min(Δ(q, n_i), Δ(q, n_j))
8: Res ← GETCANDIDATES(IF(n_i, n_j), r, W)        //Get objects matching W within range r
9: while H ≠ ∅ and minDist < r do
10:    (n, Δ(q, n)) ← dequeue(H)
11:    for each non-visited adjacent node n' of n do
12:       if Δ(q, n') < r then
13:          enqueue(H, (n', Δ(q, n')))
14:       end if
15:       Res ← Res ∪ GETCANDIDATES(IF(n_i, n_j), r, W)    //Get objects matching W within range r
16:    end for
17:    minDist ← peek(H)                          //Update with distance to head of queue
18: end while
19: return Res
```

The pseudocode of the I3RS algorithm is presented in Algorithm 5.2, and as input the algorithm takes the query point q, the search range r, and the set of query keywords W. Like the I3kNN algorithm we start by initializing a result set *Res*, and a priority queue

$\mathcal{H}$ for the network nodes. Next, the network segment where q is located, edge (n_i, n_j), is found by searching the network R-tree using q as input (line 5). We calculate the network distance from q to each of the end-nodes n_i and n_j, and add a tuple to $\mathcal{H}$ for each node that consists of a reference to the node together with the distance (line 6), e.g. $(n_i, \Delta(q, n_i))$. The minimum of the end-node distances is assigned to a variable *minDist*, and is used as a search boundary to stop the network expansion when the nearest network node is farther away than the query range r (line 7).

Before expanding beyond the first segment we retrieve all spatio-textual objects from the corresponding inverted file, $IF(n_i, n_j)$, whose description contains any of the keywords specified in W and that is within range r to q (line 8). We follow the inverted file pointer stored in the adjacency list entry of n_i to access the inverted file of the segment. In the inverted file we traverse the vocabulary to find the terms matching the keywords in W, and subsequently retrieve the corresponding postings lists. We traverse the lists, comparing the distance to the objects in the postings with the query range r, and adding to the result set the objects within the range.

The main network expansion loop happens in line 9 to 18. Inside the loop we start by dequeuing $\mathcal{H}$ to retrieve the non-visited node n that has the currently shortest distance to q (line 10). Next, we retrieve the adjacency list of n from the adjacency component, and for each entry in the adjacency list, we enqueue the adjacent node n' to $\mathcal{H}$ (line 13). We also access the inverted file of the segment, $IF(n, n')$, subsequently adding to the result set *Res* the matching spatio-textual objects on the segment (line 15). At the end of the loop in line 17, we update *minDist* with the distance to the nearest non-visited network node at the head of the queue $\mathcal{H}$.

Finally, in line 19 the I3RS algorithm terminates by returning the result set *Res* that contains the set of spatio-textual objects within range r that matches any of the query keywords in W.

Range query processing example

To present an example of the I3RS algorithm we employ the road network and the spatio-textual objects of Figure 5.3. Assume the keyword range query $Q_{KR} = \langle q, W, r \rangle = \langle q, restaurant, 13 \rangle$. Like in the preceding kNN query processing example, we assume that the non-labeled objects in the figure are not a match for the query, for simplicity. The I3RS algorithm starts by retrieving the edge where q lies by running a query on the network R-tree in Figure 5.2d, using q as input. As the next step, the algorithm consults the inverted file IF_3 of the retrieved edge, edge (n_1, n_4), to find any spatio-textual object in the file that has a network distance to q less or equal to the specified range r. None are found, and the distance to both end-nodes from q are computed. Continuing, the nearest network node to q is expanded and the adjacent road segments are to be examined for matching spatio-textual objects. In edge (n_1, n_3) there are no terms matching the query keywords in the vocabulary of inverted file IF_2, and the algorithm continues to examine edge (n_1, n_2). On this edge, as we consult the inverted file IF_1, we find the query keyword *restaurant* among the terms in the vocabulary. Consequently, we retrieve the postings list of the term, and traverse the list adding objects that are

within the specified range of $r = 13$ to the result set. The distance to the objects are computed by adding the distance from q to node n_1 with the distance stored in the respective posting, i.e. $\Delta(q, p_i) = \Delta(q, n_1) + \Delta(n_1, p_i)$. After having traversed the postings list of term *restaurant* in IF_1 the result set consists of the objects $\langle p_1, p_3 \rangle$. They are the objects within the specified range r, and their description contains the query keyword *restaurant*.

In conclusion, this chapter presented the RNI framework for efficient processing of spatial keyword queries on road networks. We have described the improved keyword-based road network architecture, and we have presented algorithms for processing keyword-based k-nearest neighbour and keyword-based range queries.

Chapter 6

OpenStreetMap

The purpose of the two frameworks we propose in this thesis is to be used as the bottom index layer in an application that can provide to a user services such as searching for spatio-textual objects in a spatial network database. In order to test and compare the baseline and RNI frameworks properly we employ real-world cartographic data from the web service OpenStreetMap[1] (OSM). OpenStreetMap is a project whose goal is to create a free editable map of the whole world. It is a collaborative project where the contributors are a community of people from all over the world with an interest in mapping. The focus of this chapter will thus be to describe the OSM web service, the data that is available, and some of the processes we have gone through in order to extract and convert the data for use in the different components of our two frameworks.

This chapter first presents, in Section 6.1, the data that is available from OpenStreetMap, explaining different terms and giving examples. Subsequently, Section 6.2 discusses how data can be extracted from OpenStreetMap and presents algorithms for this task.

6.1 Data available from OpenStreetMap

The OpenStreetMap web service allows a user to export a custom map to several different formats, such as the Mapnik[2] image shown in Figure 6.1, as embeddable HTML, or as OpenStreetMap XML Data. For the purpose of creating a spatial network database coupled with spatio-textual objects, we have chosen to export the map to the OSM XML format. XML provides an organized and structured way of storing the data, which makes it easier to retrieve and convert into a useful format for use as the data source in the implementation of our proposed baseline and RNI frameworks. In this map we have data that describes the following:

- The actual roads, their coordinates and other useful information such as driving direction and speed-limit.

[1] `http://www.openstreetmap.org`
[2] `http://mapnik.org`

- The nodes that connect two roads, i.e. the vertices in the graph model of the road network, or what is considered the road intersections.
- Other objects of interest such as shops, cafés, restaurants, and ATMs, with a textual description of what the objects are, as well as the coordinates for their spatial location, i.e. the spatio-textual objects.

Figure 6.1 shows a view over Campus Gløshaugen at NTNU, exported from the OpenStreetMap web service. The map shows, for example, where there exist roads and footpaths, e.g. the dotted lines, some of the parking spots in the area, indicated by the P's, as well as some of the street names. There is a limitation with the Mapnik image format in that it does not show all the objects of interest in the map. When exporting this map to the OSM XML format everything from the roads to staircases, as well as cafés and lecture halls will be available.

A *node* in the OSM scheme is a basic element or building block in a map, and consists of an id, the latitude (lat) and longitude (lon) coordinates, and an optional set of tags used to describe or classify the node. The set of tags are key-value pairs with no key occurring twice in a node, and may for example be used to denote that a node represents a restaurant or a historic attraction (OpenStreetMap contributors, 2011).

As an example, Listing 6.1 is a description of a *node*, and its information, in the OpenStreetMap XML format. The code is extracted from the XML file generated based on the information present in the map in Figure 6.1. Here, a tag with the key *amenity* has the value *cafe*, and another has the key *name* with the value *Cafe-sito Stripa*. With this information we see that the node is used to represent a café with the name "Cafe-sito Stripa", and that it is located at (lat=63.4169296, lon=10.4042830), indicated by the dotted point in the lower middle of Figure 6.1. There are also other attributes in the node header such as the timestamp of the last edit and the user who performed it, as well as a version number and the changeset which the last update belongs to.

```
1 <node id="431038572" lat="63.4169296" lon="10.4042830" user="Espenso" \
2         uid="176415" visible="true" version="4" changeset="2852590" \
3         timestamp="2009-10-15T10:01:32Z">
4  <tag k="amenity" v="cafe"/>
5  <tag k="cuisine" v="coffee_shop"/>
6  <tag k="level" v="1"/>
7  <tag k="name" v="Cafe-sito␣Stripa"/>
8  <tag k="operator" v="Studentsamskipnaden␣i␣Trondheim"/>
9 </node>
```

Listing 6.1: A section from the OSM XML describing a node, © OpenStreetMap contributors, CC-BY-SA.

The OSM scheme describes a *way* as an ordered interconnection of two or more nodes representing a linear feature such as a street, footpath, river, or a bridge (OpenStreetMap contributors, 2011). Listing 6.2 describes a way in the OSM XML format. Similarly to the node in Listing 6.1, the way in Listing 6.2 has attributes for an id, the user who last modified the object, a version number and a timestamp for the last edit. A way is

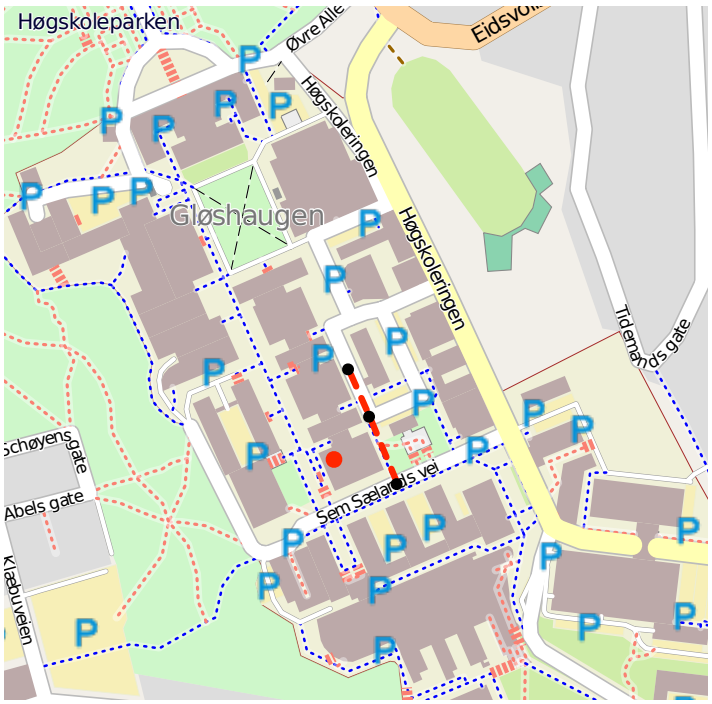


Figure 6.1: An OpenStreetMap view of Campus Gløshaugen, NTNU in Trondheim, Norway, © OpenStreetMap contributors, CC-BY-SA.

composed of the nodes in the list of *nd* attributes, which are ordered according to their latitude and longitude coordinates.

For example, in Listing 6.2 the way consists of three nodes with the ids 436127539, 436153454, and 436127538. When plotting the way composed by these three points, the result is the dashed line presented in the lower middle of Figure 6.1, with the black points indicating the nodes. A way may also contain a set of tags, similarly to a node, used for describing the properties of the feature. In the case of Listing 6.2 the set of tags contain the key-value pairs *foot-yes* and *highway-cycleway*, giving us the information that this way can be walked and bicycled. Other tags may for example describe the direction of a road, the name of a highway, whether a speed limit is enforced, and what the actual speed limit is.

```
1 <way id="37363527" user="dperkis" uid="133533" visible="true" version="2"
      changeset="1760108" timestamp="2009-07-07T08:56:11Z">
2   <nd ref="436127539"/>
3   <nd ref="436153454"/>
4   <nd ref="436127538"/>
5   <tag k="foot" v="yes"/>
6   <tag k="highway" v="cycleway"/>
7 </way>
```

Listing 6.2: A section from the OSM XML describing a way, © OpenStreetMap contributors, CC-BY-SA.

There exists also the notion of a *relation*, which is a group of zero or more data primitives, used for specifying relationships between objects (OpenStreetMap contributors, 2011). By employing the relation data primitive it is possible to combine several ways into one longer path, or specifying a network of roads that has a special relationship. It is also possible to specify an area, such as a lake, by creating a closed way that starts and ends in the same node, subsequently tagging it with a key that is handled as an area.

6.2 Extracting data from the XML file

In order to extract data from the XML file to build a road network, we scan the file two times, using the data from the first scan as a lookup facility in the second scan. More specifically, in the first scan of the file we search for nodes and insert them into a HashMap as they are encountered, using the node id as the key and the spatial coordinates as the value. When the HashMap is filled we have a quick way of looking up the coordinates of a node given the id.

Algorithm 6.1 EXTRACTWAYS($file$)

```
1:  INPUT: the OSM XML file
2:  OUTPUT: a set consisting of the ways from the XML file
3:  nodeSet ← ∅
4:  waySet ← ∅
5:  for each entry in file do                          //first iteration, extracting nodes
6:     if entry is a node then
7:        nodeSet ← nodeSet ∪ node
8:     end if
9:  end for
10: for each entry in file do                          //second iteration, extracting ways
11:    if entry is a way then
12:       nodeList ← ∅
13:       for each nodeId in way.nodeIdList do
14:          nodeList ← nodeList ∪ nodeSet.get(nodeId)
15:          if nodeId is a member of another way then
16:             split the way in two
17:             add a neighbour relationship between the first node of nodeList and nodeId
18:             add a neighbour relationship between nodeId and the last node of nodeList
19:          end if
20:       end for
21:       add a neighbour relationship between the first and last node in nodeList
22:       waySet ← waySet ∪ way
23:    end if
24: end for
25: return waySet
```

During the second scan the goal is to create roads as polylines for later insertion into the polyline component in Figure 5.2c. When a way is encountered in the XML file we

know that it consists of the nodes that compose the way, given by a list of node ids. Thus, for each way we iterate through the corresponding node list, using the HashMap to look up the coordinates for each node, and storing each way as a polyline formed by the series of straight line segments between the nodes. As part of the same process we also construct the adjacency relationship between the nodes. If the intermediate nodes of a polyline is only a member of that one polyline, we create a neighbour relationship between the first and last node of the polyline. On the other hand, if an intermediate node is a member of one or more other ways, we can split the way into two or more shorter polylines, subsequently creating adjacency relationships between the first and last nodes of each of the shorter polylines. Algorithm 6.1 presents the pseudocode for the extraction of nodes and ways from the XML file, and takes as input the file itself.

For extracting the spatio-textual objects from the XML file we have created the EXTRACTSPATIOTEXTUALOBJECTS(*file*,*T*) algorithm presented in Algorithm 6.2. The spatio-textual objects, or nodes in the file, are characterized by having a set of tags as key-value pairs, describing the node in one way or another. By selecting which tag keys shall be considered as a textual description, we can filter the nodes that have these keys and classify them as spatio-textual objects. The set of these tag keys, T, is sent as input to the algorithm, together with the XML file itself. Although it is possible to perform this operation as a part of the first iteration of EXTRACTWAYS(*file*) in Algorithm 6.1, we have separated the two algorithms for clarity.

After the information about the nodes and the ways has been gathered, we can use this to create data for the adjacency component and the polyline component, Figure 5.2a and 5.2c respectively. At this point, there still is some processing left to determine which nodes are adjacent to one another, and computing the distance and MBRs of each way. In addition to this, all the nodes that were classified as spatio-textual objects during the execution of the EXTRACTSPATIOTEXTUALOBJECTS(*file*,*T*) algorithm have been written out to a new file for later insertion into either the IR-tree in the spatio-textual component, Figure 4.1b, or the inverted files in the inverted file component, Figure 5.2b. An excerpt of this file can be seen in Listing 6.3, where each line represents one spatio-textual object. The values are the id of a node, its latitude and longitude coordinates, and the key-value pairs from the set of tags, in this case used as the description of the object.

Algorithm 6.2 EXTRACTSPATIOTEXTUALOBJECTS($file, T$)

```
1:  Input: the OSM XML file, the set of keys used to describe a spatio-textual object
2:  Output: a set consisting of the spatio-textual objects from the XML file
3:  spatiotextualSet ← ∅
4:  for each entry in file do
5:      if entry is a node then
6:          if ∃node.tag.key ∈ T then                    //node is a spatio-textual object
7:              spatiotextualSet ← spatiotextualSet ∪ node
8:          end if
9:          add the tags to the node's description field
10:     end if
11: end for
12: return spatiotextualSet
```

```
1 153142045 63.4313285 10.3928913 amenity pub microbrewery yes name Trondhjem
      Mikrobryggeri
2 956343641 63.4326826 10.3975758 amenity fast_food cuisine sandwich name Big
      Bite
3 383903540 63.4306835 10.3983484 name Mercursenteret shop mall
4 385719944 63.4302163 10.3899054 amenity restaurant cuisine asian name Mien
     Trung Cafe
5 956343651 63.4318762 10.3935699 amenity nightclub name Familien
6 380355308 63.4306906 10.3906721 amenity restaurant cuisine italian name
      Napoli
```

Listing 6.3: Spatio-textual objects, as extracted from an OSM XML file and written to a new file.

As mentioned, when creating a node or a way in OpenStreetMap it is possible to use a set of tags to describe the entity being created. There are no restrictions to which tags can be assigned to an entity, but the OpenStreetMap wiki[3] contains a core set of features and corresponding tags that is recommended to use. At the top level the features are divided into the categories physical, non-physical, naming, annotation, and editor keys, where each of these are further divided into more and more specific categories. Within the physical category there are subcategories such as highway, railway, leisure, amenity, shop, tourism, and natural, whereas the non-physical contain subcategories such as route, boundary, sport, and various restrictions. In general, the categories mentioned here are used as keys in the tags, and the values coupled with the key may again be further subcategorized into, for example, restaurant, pub, bakery, and so on.

Physical	Highway, Barrier, Cycleway, Tracktype, Waterway, Railway, Aeroway, Aerialway, Power, Man Made, **Leisure**, **Amenity**, Office, **Shop**, **Craft**, **Emergency**, **Tourism**, **Historic**, Landuse, Military, Natural, Geological
Non-physical	Route, Boundary, **Sport**, Abutters, Accessories, Properties, Restrictions
Naming	Name, References, Places, Addresses
Annotation	note, fixme, description
Editor keys	created_by, history

Table 6.1: The most common categories of keys used for tagging in OpenStreetMap.

To extract entities from the map to best represent our notion of spatio-textual objects, we have employed the following subcategories of the physical category: leisure, amenity, shop, craft, emergency, tourism, and historic. From the non-physical category we have included the sport subcategory. These eight categories, which all are used as keys in a tag, represent entities that are likely to be used as keywords for an application based on keyword search in spatial network databases. Although there may be some

[3] `http://wiki.openstreetmap.org/wiki/Map_Features`

overlap between the different categories as to which can be considered a description of a spatio-textual object, the selected eight provides a good coverage of such objects when used for testing and evaluation purposes in a project like this. When evaluating the spatio-textual objects that are filtered out as part of Algorithm 6.2, this proves to be a good assumption.

Table 6.1 presents the main categories of keys that can be used to tag nodes and ways, with the more commonly used listed first in each category. The eight categories that are used to classify spatio-textual objects are presented in bold font. Each of these categories may again contain more detailed keys, with the exception of the entries in the *Annotation* and *Editor keys* lists which are actual keys, not just categories.

Chapter 7

Experimental Evaluation

In this thesis we have presented two frameworks that solves the problem of processing spatial keyword queries on road networks. In order to test these frameworks properly we have run a series of experiments on real-world datasets of different size to detail how the frameworks perform under various conditions. We chose to implement the keyword-based k-nearest neighbour algorithm for both frameworks, i.e. the INE* and I3kNN for the baseline framework and RNI framework, respectively. Thus, all the experiments are based on running nearest neighbour queries on the frameworks while varying different parameters such as the number of query results k, and the number of query keywords. The output from the experiments is the average response time a query uses to return the answer, and the average number of I/Os the query uses during the whole process.

In this chapter we will present and discuss the results from these experiments. Since the RNI framework is intended to alleviate some of the performance issues with the baseline approach, we will especially focus on comparing the query performance of the two frameworks, explaining the reasons for differing query performance.

Both of the frameworks, and the two algorithms, were implemented using the Java programming language. All of the experiments were executed on a PC running the Ubuntu 10.04 operating system, with dual Intel Xeon E5520 CPUs and 18GB of RAM.

In the following section we present the datasets used throughout the experiments, along with their characteristics such as size, number of ways, and number of spatio-textual objects. Section 7.2 presents the main parameters and values of the experiments, as well as other details about the experimental setup. In Sections 7.3 and 7.4 we introduce the two different sets of experiments; in the first set we run nearest neighbour queries on the frameworks varying number of results k, whereas the number of keywords are varied in the second set. Results and a discussion follows the introduction of each of the set of experiments.

7.1 Datasets

In this section we describe the different datasets used throughout the experimental evaluation. We employ three different datasets based on real-world data extracted from the OpenStreetMap web service. The advantage of OpenStreetMap is that it gives us actual road networks with latitude and longitude coordinates that can be used to accurately calculate the real-world network distance between two points. OSM also contains spatio-textual objects, which together with the road networks make a good testing platform for the two frameworks we present in this thesis. More details on OpenStreetMap, the data that are available, and a walkthrough of the processes used to extract the data can be found in the previous chapter, Chapter 6.

	Trondheim	**London**	**Netherlands**
Size in MB	38	293	3 188
Tot. no. of ways	27 267	238 080	2 037 817
Tot. no. of objects	1 063	31 860	26 437
Tot. no. of words	7 338	216 001	216 466
Tot. no. of unique words	1 151	11 582	15 838
Avg. no. of unique words per object	6.23	6.34	7.26
Avg. no. of objects per km^2	3.54	20.61	2.01
Area in km^2	299.7	1 546.1	13 210.6
Bounding box (bottom, left)	(63.325, 10.27)	(51.342, -0.449)	(51.77, 4.13)
Bounding box (top, right)	(63.466, 10.654)	(51.649, 0.206)	(52.86, 5.75)

Table 7.1: Characteristics of the datasets.

Table 7.1 presents characteristics such as the size, the number of ways, and the number of spatio-textual objects of each dataset used in the experiments. The first two datasets are extracted from the cities of Trondheim in Norway and London in England. The third dataset is a subset of the Netherlands containing cities such as Amsterdam, Rotterdam, Den Haag and Utrecht. From the table, we see that the three datasets vary about two orders of magnitude in size on disk as well as number of ways, from the smallest to the largest. The different sizes should provide good insight into how the two frameworks perform on small, medium, and large sized datasets. The Trondheim dataset is clearly the smallest of the three in terms of size on disk, number of ways, and number of spatio-textual objects. London is the next step, with a significant increase in both number of ways and number of spatio-textual objects from the Trondheim dataset. When comparing London and the Netherlands datasets, we see from the table that London has a higher amount of spatio-textual objects than the Netherlands, even though the latter has been extracted from a much larger area. This is reflected in the density of the spatio-textual objects, *Avg. no of objects per* km^2, and the Netherlands has the lowest average among the three datasets, with only 2.01 objects per km^2. The difference in object density will give an indication of the effect a low-density road network has on the query performance.

The coordinates of the bounding boxes used when extracting the datasets from OpenStreetMap can be found in Table 7.1. The numbers are given as latitude and longitude

coordinates (lat, lon), and indicate the bottom left and top right of the bounding box of each dataset, respectively.

As part of building the indexes, the datasets go through a pre-processing step where each spatio-textual object is moved to the nearest polyline to ensure that objects lying away from the road network are discovered during network traversal. In the same process we also remove closed circuits from the datasets, i.e. areas of the road network that starts and ends on the same network node, and are thus disconnected from the rest of the road network.

7.2 Experimental setup and parameters

Parameter	Values
Number of results (k)	1, 5, **10**, 15, 20
Number of keywords	1, 2, **3**, 4, 5
Datasets	trondheim, london, netherlands

Table 7.2: Settings used in the experiments. The default values are presented in bold.

Table 7.2 presents the main parameters and values used throughout the experiments. The default values are presented in bold. In the first set of experiments the number of results k were varied from 1 to 20 in increments of 5, while the number of keywords were kept at the default of 3. Fixing the number of keywords to 3 is a reasonable number since the average number of terms per spatio-textual object is between 6 and 8, as shown in Table 7.1. It is also likely that 3 provides a good measure of the number of keywords a user would try when performing such a query in an actual application. In the second set of experiments the number of keywords were varied from 1 to 5, while the number of results k were fixed at 10 throughout the experiment.

In each of the experiments, before the query process starts, an initialization process is executed. The goal of the initialization process is to select the latitude and longitude coordinates of the query point, and to select the query keywords that are used as input to the algorithm. The query point is chosen by obtaining a set of coordinates from a polyline, randomly retrieved from the R-tree in the network component, Figure 5.2d. This is done in order to ensure that the query point is located on an actual segment in the road network. As for the query keywords, these are selected from the term vocabulary based on a probability function that is more likely to pick a commonly used keyword, than a rare one.

During all of the experiments, none of the index structures (IR-tree, network R-tree, inverted files) had any form of caching activated. This means that whenever the index structures are consulted as part of the nearest neighbour algorithm they have to be fetched from disk, and not just from a cache in memory. The results from the experiments should thus give a good indication of the I/O each framework requires to process the queries.

The two sets of experiments are run on all three datasets for successive rounds in order to provide a confident result. The aggregated data from these rounds are then used to calculate the average response time and the average I/O of the query in each experiment, as shown later in the results. All the results are plotted using a logarithmic scale on the y-axis.

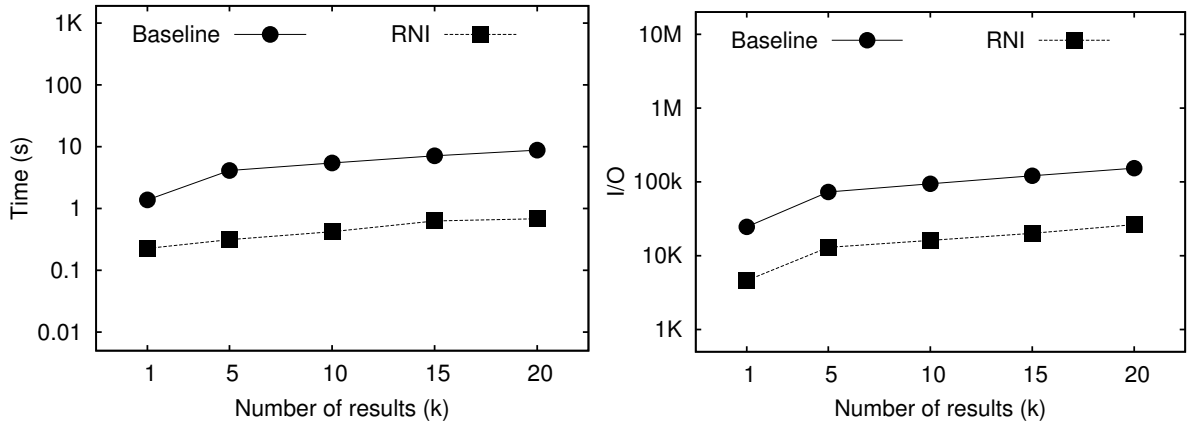
7.3 First set of experiments: varying number of results

The first set of experiments is designed to see how the two frameworks perform when processing nearest neighbour queries on different datasets, for different values of k. The number of query keywords is fixed at the default value of 3 throughout all the experiments.

As output from the experiments we record the average response time of each query, and compute the average number of I/Os each query requires, for each of the values of k.

Before executing each query, an initialization process is conducted to select the query location and the query keywords, as explained previously.

Results and discussion



(a) Response time. **(b)** I/O.

Figure 7.1: Trondheim dataset. Response time and I/O varying the number of results (k).

Figure 7.1 plots the results of running a set of nearest neighbour queries on the baseline and RNI frameworks, using the Trondheim dataset while we vary the number of results k. The RNI framework clearly outperforms the baseline framework with a response time of at least one order of magnitude lower, as seen in Figure 7.1a. Although the response time increases for both frameworks as the number of results increases, the improvement in performance is consistently in favour of RNI for all values of k. The increasing response time is expected though, since we have to expand further in the network for each increasing value of k to find enough matching objects. The main

reason that RNI has a lower response time is that the query has to access fewer disk blocks than baseline during query processing to find the same amount of results. This is reflected in Figure 7.1b which shows the I/Os for both frameworks, also here with one order of magnitude improvement for the RNI framework.

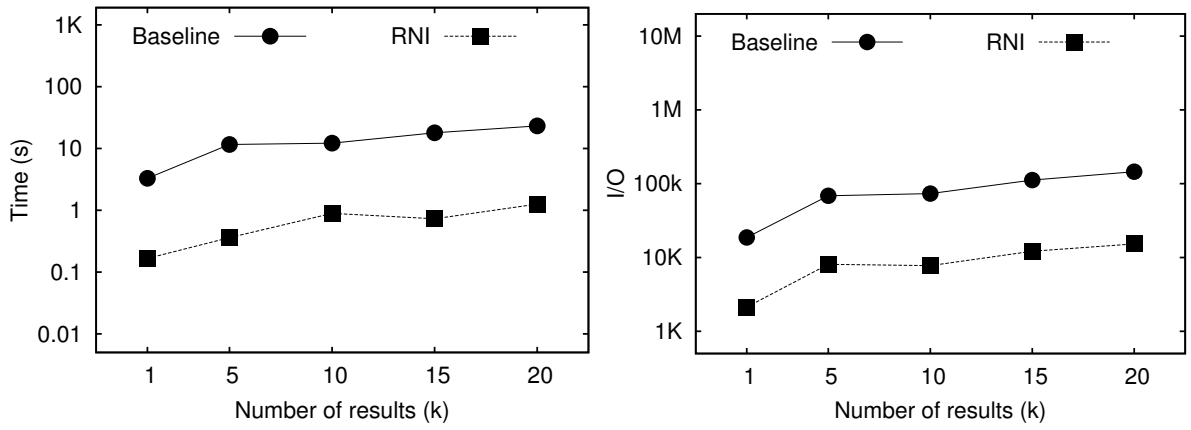
The lower I/O for the RNI framework can be explained by the following. Each time the query in the baseline framework encounters a road segment, we search the IR-tree to find the matching spatio-textual objects on the segment. The query on the IR-tree is defined by the MBR of the road segment and the query keywords. As we traverse nodes in the IR-tree, I/Os occur when we retrieve a node, as well as when we retrieve the inverted file of a node to check which child nodes may contain any matching objects. After the IR-tree has been traversed and the candidate objects have been found, we have to check for false positives, i.e. we have to compare the location of the objects with the exact geometry of the road segment to verify that the objects are present on the road segment. Thus, we retrieve the polyline of the road segment from the polyline component, resulting in more disk accesses. At this step we also compute the distance to each of the spatio-textual objects, and subsequently sort them on network distance to the query, an operation that contributes to an increased response time.

With the RNI framework, on the other hand, we only have to access one inverted file for each encountered road segment, not several nodes and several inverted files as during traversal of the IR-tree, resulting in fewer I/Os. Additionally, we can avoid accessing an inverted file if it does not contain any objects. The reason that the last statement is possible is that we know already during building of the indexes if an inverted file contains any objects, and can thus avoid accessing the file during query processing if it is empty. This can be achieved by, for example, setting the inverted file pointer in the adjacency component to -1, or some other defined value indicating an empty file. I/Os are further avoided since we do not need to retrieve the polyline to check for false positives in the RNI approach, because we know that an object in an inverted file is guaranteed to lie on the corresponding segment. Furthermore, the distance from a spatio-textual object to an end-node of the segment is pre-computed and stored in the posting, and the polyline is not needed to perform this task, as it is in the baseline approach.

When running the same set of nearest neighbour queries on the two frameworks using the London dataset we get the results presented in Figure 7.2. Again, the performance of the RNI framework is consistently better than the baseline. In response time, Figure 7.2a, the improvement is one to two orders of magnitude, whereas in I/Os the improvement is about one order of magnitude.

The reasons for the improvement in both response time and I/Os are the same as for the Trondheim dataset. Query processing on the RNI framework requires less I/Os than the baseline to retrieve the same amount of query results.

Although the London dataset is much larger than the Trondheim dataset in number of ways, that does not seem to have a big impact on the query performance of the frameworks, but this may be attributed to the much higher density in spatio-textual objects. A denser network means that the query is more likely to find matching spatio-textual objects near the query point, avoiding extensive expansion of the network.

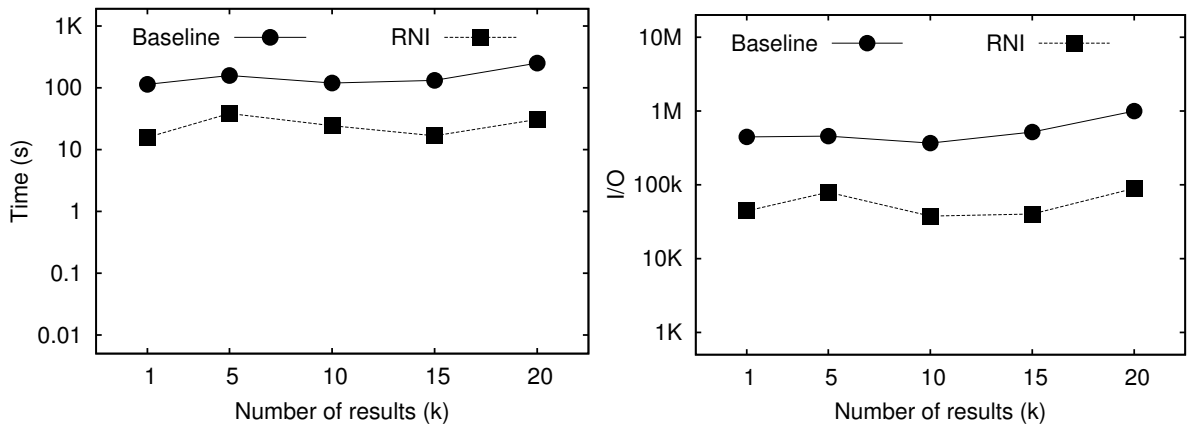


(a) Response time. **(b)** I/O.

Figure 7.2: London dataset. Response time and I/O varying the number of results (k).

Looking at the results of running the queries on the Netherlands dataset, Figure 7.3, we see the effect a large network, sparsely populated with spatio-textual objects, has on the query performance. The RNI framework still performs about one order of magnitude better than the baseline approach, but the average response time and I/Os has increased significantly for both frameworks. This overall increase in response time and I/O is clearly the result of a large part of the network being expanded before finding enough results to satisfy the query.

Another thing to note about the Netherlands dataset is that the distribution of spatio-textual objects are likely to be concentrated around the large cities of the extracted area. Since we have extracted about half a country, there are large areas with few or no spatio-textual objects at all, such as large rural areas. If the query point is selected from a road in such an area during the initialization phase, the query is required to expand a very large part of the network before finding the required amount of results.



(a) Response time. **(b)** I/O.

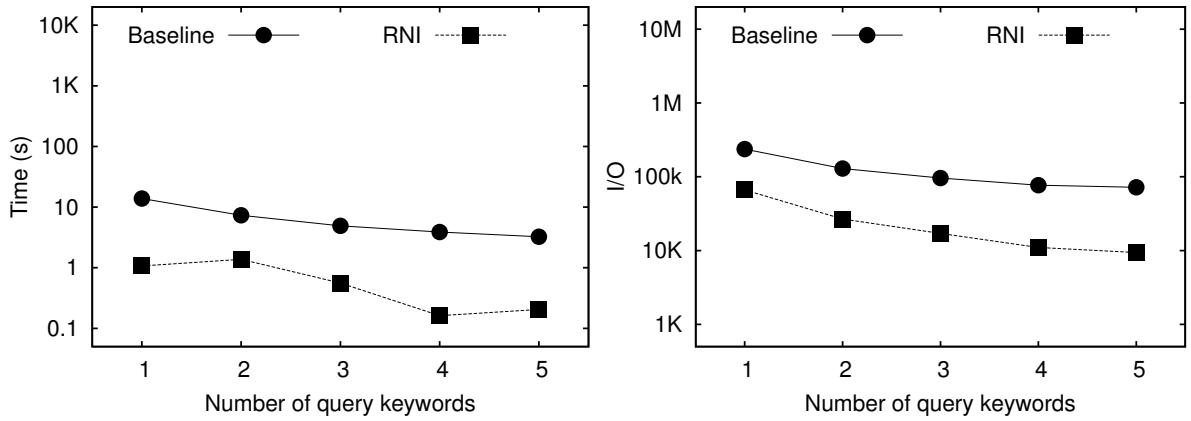
Figure 7.3: Netherlands dataset. Response time and I/O varying the number of results (k).

7.4 Second set of experiments: varying number of keywords

In the second set of experiments we want to see how the two frameworks perform when processing nearest neighbour queries on different datasets while varying the number of query keywords. The number of results k is in these experiments fixed at the default value of 10.

Like in the first set of experiments we first conduct an initialization process to select the query keywords and the query location. The output is the average response time for each query, and the average number of I/Os each query requries, reported as the number of keywords vary.

Results and discussion



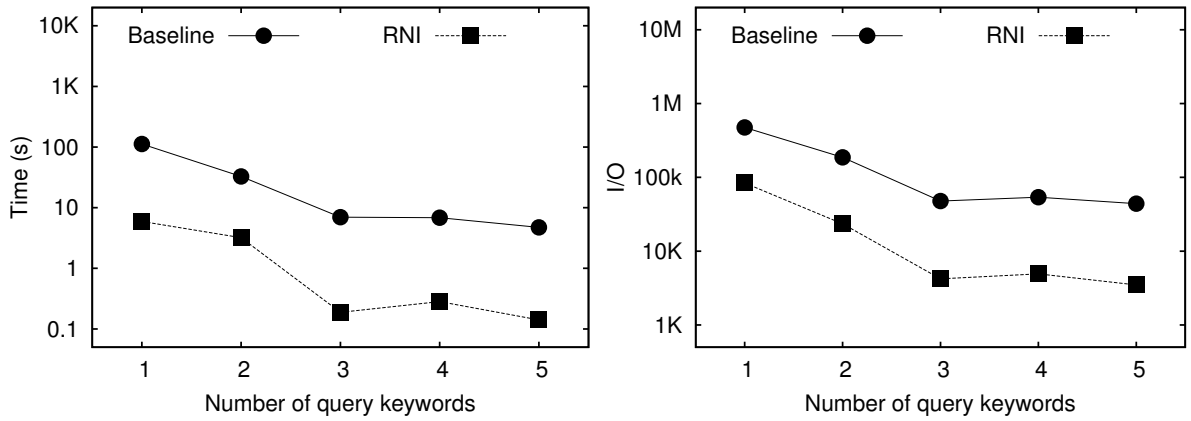
(a) Response time. **(b)** I/O.

Figure 7.4: Trondheim dataset. Response time and I/O varying the number of keywords.

The results of varying the number of keywords when running a set of nearest neighbour queries on the baseline and RNI frameworks is depicted in Figure 7.4; the Trondheim dataset has been used as the source. We see that the trend from the previous set of experiments can also be found in the results here. The RNI outperforms the baseline with about one order of magnitude for both response time and I/O, Figure 7.4a and 7.4b. The reason for the performance improvement is the same as described before. The amount of I/Os are greatly reduced during query processing on the RNI framework since we only retrieve one inverted file for each segment if there are objects present, and subsequently avoid additional costly queries on the IR-tree to find the spatio-textual objects. Also, the polyline component is only visited during the initialization phase to retrieve the first road segment, and not for each encountered road segment as in the baseline approach.

From the results we also see that the response time is highest when we have one query keyword, and subsequently decreases as the number of keywords increases. This is

because we only require that the description of a spatio-textual object contains one of the keywords for it to be reported as a match. Consequently, when we specify just one keyword the query has to expand the network farther to find the desired amount of matching objects, whereas when we specify several keywords it is more likely that we will encounter objects that contain any of the keywords within a shorter distance to the query point.



(a) Response time. **(b)** I/O.

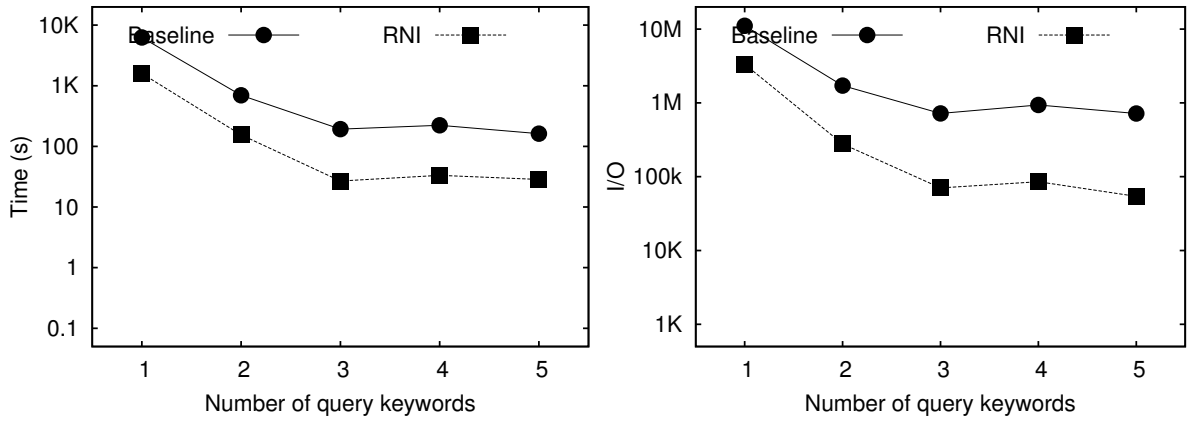
Figure 7.5: London dataset. Response time and I/O varying the number of keywords.

For the larger and more densely populated dataset of London, the results of running the nearest neighbour queries are presented in Figure 7.5. The trend is clearly showing an advantage to the RNI framework for both response time and I/O, like we have seen in the previous results. As before, the improvement in performance is attributed to the number of disk accesses the frameworks require during query processing. The IR-tree of the baseline has to run costly searches for each road segment to retrieve the spatio-textual objects, while the inverted file component of the RNI only have to retrieve a single inverted file if there exists any objects on a road segment.

When comparing the results in Figure 7.5 with the results of the Trondheim dataset in Figure 7.4, it shows that the advantage to RNI has increased slightly. This improvement indicates that the query performance of the RNI framework improves even further when the network is densely populated. It is more likely that the IR-tree will retrieve false positives when there are more objects in the network, and the time spent in the refinement step to discover and discard the false positives will increase.

In Figure 7.6 we present the results of running the second set of experiments on the Netherlands dataset. The results shown here are interesting since the overall response time and I/Os has increased significantly for both frameworks when compared to the London dataset, especially when we only use one query keyword. Same as for the first set of experiments, the increase in response time and I/O for the Netherlands dataset is due to the low density of spatio-textual objects, combined with a very large road network.

The reduced performance when we have few query keywords indicates that there is a need for additional improvements of the RNI framework to account for such cases. One such case is when the query keyword is rare, meaning that few spatio-textual



(a) Response time. **(b)** I/O.

Figure 7.6: Netherlands dataset. Response time and I/O varying the number of keywords.

objects has the term in their description. In this case, if the number of matching spatio-textual objects are fewer than the required number of query results, we run the risk of expanding the whole network on the search for objects. A solution to this problem is to keep a vocabulary of terms together with the total number of objects that is described by the term. Before starting the query process, the vocabulary is consulted to see how many objects contain the term. If the number of query results k is larger than the total number of objects, we can set the latter number as a boundary and terminate the search when all objects have been found. Another approach is to include a range boundary in the nearest neighbour query, that stops the search when the boundary is exceeded.

Chapter 8

Conclusions

We have in this thesis presented a twofold solution to the new and complex problem of answering spatial keyword queries where the distance between the spatio-textual objects are constrained by road networks. Our solution consists of 1) a baseline framework that combines existing state-of-the-art approaches to support processing of keyword-based spatial queries such as range and k-nearest neighbour on road networks, and 2) a novel framework termed Road Network Indexing (RNI) that permits efficient processing of such queries by indexing the spatio-textual objects in each road segment using inverted files.

We have also developed algorithms to process keyword k-nearest neighbour and keyword range queries on both the baseline and the RNI framework. The keyword k-nearest neighbour query retrieves the k closest spatio-textual objects to a query location q that matches any of the keywords specified in the query, whereas the keyword range query retrieves all spatio-textual objects within range r of q that matches any of the query keywords.

Through an experimental evaluation we have presented results from running nearest neighbour queries on the frameworks, using differently sized real-world road networks extracted from the OpenStreetMap web service. From the experiments we show how the average response time and average I/O are affected as the number of results k, and number of query keywords, are varied. The results show that the RNI framework outperforms the baseline framework with around one order of magnitude for all experiments. Furthermore, the results also show that the performance increases in favour of the RNI framework as the density of the spatio-textual objects increases in the road network, indicating potential for scalability with further development of the RNI framework.

The work of this thesis is only the first step towards providing more beneficial spatial keyword queries that employ road networks to find the best objects matching the query. Current and future applications that may benefit from our solution include navigation and trip planning software, tourist portals on the Web, and the many search engines out there in need of a spatial extension to their application.

Future work

There are still much left to cover on the topic of keyword-based spatial queries on road networks. What would be interesting to do as part of a future study is to implement more algorithms to run on the frameworks, such as the keyword-based range algorithm proposed in this thesis. It would also be interesting to do a further extensive performance comparison using synthetic datasets to better control object density, detailing the scalability of the frameworks and the proposed algorithms.

There is also room for improvements of the performance of the RNI framework, especially with regard to sparsely populated datasets. In this context it could be interesting to look at an improvement of the road network representation. In sparsely populated datasets there are large areas without any objects of interest. A way to alleviate this issue could be to truncate the empty areas, creating shortcuts from the start of the area to the end, so that the network expansion did not have to expand all the empty road segments within such an area.

References

Arge, L., de Berg, M., Haverkort, H. J., and Yi, K. (2004). The Priority R-tree: a practically efficient and worst-case optimal R-tree. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 347–358.

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Chen, Y.-Y., Suel, T., and Markowetz, A. (2006). Efficient query processing in geographic web search engines. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 277–288.

Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: an efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 426–435.

Cong, G., Jensen, C. S., and Wu, D. (2009). Efficient retrieval of the top-k most relevant spatial web objects. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 337–348.

de Almeida, V. T. and Güting, R. H. (2006). Using Dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 58–62.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Faloutsos, C. and Christodoulakis, S. (1984). Signature files: an access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems*, 2(4):267–288.

Felipe, I. D., Hristidis, V., and Rishe, N. (2008). Keyword search on spatial databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 656–665.

Finkel, R. A. and Bentley, J. L. (1974). Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9.

Frank, A. U. (1992). Spatial concepts, geometric data models, and geometric data structures. *Computers & Geosciences*, 18(4):409–417.

Gaede, V. and Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 47–57.

Hariharan, R., Hore, B., Li, C., and Mehrotra, S. (2007). Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1–10.

Hjaltason, G. R. and Samet, H. (1999). Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318.

Hu, H., Lee, D. L., and Lee, V. C. S. (2006a). Distance indexing on road networks. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 894–905.

Hu, H., Lee, D. L., and Xu, J. (2006b). Fast nearest neighbor search on road networks. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 186–203.

Jensen, C. S., Kolářvr, J., Pedersen, T. B., and Timko, I. (2003). Nearest neighbor queries in road networks. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems (ACM GIS)*, pages 1–8.

Kolahdouzan, M. and Shahabi, C. (2004). Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 840–851.

Kung, R.-M., Hanson, E., Ioannidis, Y., Sellis, T., Shapiro, L., and Stonebraker, M. (1986). Heuristic search in data base systems. In *Proceedings of the International Workshop on Expert Database Systems*, pages 537–548.

Lee, K. C. K., Lee, W.-C., and Zheng, B. (2009). Fast object search on road networks. In *Proceedings of the International Conference on Extending Database Technology: Advances in Database Technology (EDBT)*, pages 1018–1029.

Li, Z., Lee, K. C. K., Zheng, B., Lee, W.-C., Lee, D., and Wang, X. (2010). IR-tree: an efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering*, 1:1–13.

Manning, C. D., Prabhakar, R., and Hinrich, S. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Nievergelt, J., Hinterberger, H., and Sevcik, K. C. (1984). The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71.

OpenStreetMap contributors (2011). Data primitives - OpenStreetMap wiki. Available at `http://wiki.openstreetmap.org/wiki/Data_Primitives`. Accessed 1 June 2011.

Papadias, D., Kalnis, P., Zhang, J., and Tao, Y. (2001). Efficient OLAP operations in spatial data warehouses. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 443–459.

Papadias, D., Zhang, J., Mamoulis, N., and Tao, Y. (2003). Query processing in spatial network databases. In *Proceedings of the International Conference of Very Large Data Bases (VLDB)*, pages 802–813.

Rocha-Junior, J. B., Gkorgkas, O., Jonassen, S., and Nørvåg, K. (2011). Efficient processing of top-k spatial keyword queries. In *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD) (to appear)*.

Roussopoulos, N., Kelley, S., and Vincent, F. (1995). Nearest neighbor queries. *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 71–79.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260.

Samet, H. (1995). *Spatial data structures*, pages 361–385. ACM Press/Addison-Wesley.

Sellis, T., Roussopoulos, N., and Faloutsos, C. (1987). The R+-tree: a dynamic index for multi-dimensional objects. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 507–518.

Shahabi, C., Kolahdouzan, M. R., and Sharifzadeh, M. (2002). A road network embedding technique for k-nearest neighbor search in moving object databases. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems (ACM GIS)*, pages 94–100.

Shaw, K., Ioup, E., Sample, J., Abdelguerfi, M., and Tabone, O. (2007). Efficient approximation of spatial network queries using the M-tree with road network embedding. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 11–11.

Shekhar, S. and Liu, D.-R. (1997). CCAM: a connectivity-clustered access method for networks and network computations. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):102–119.

Yiu, M. L. and Mamoulis, N. (2004). Clustering objects on a spatial network. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 443–454.

Zhang, D., Chee, Y. M., Mondal, A., Tung, A. K. H., and Kitsuregawa, M. (2009). Keyword search in spatial databases: towards searching by document. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 688–699.

Zhou, Y., Xie, X., Wang, C., Gong, Y., and Ma, W.-Y. (2005). Hybrid index structures for location-based web search. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 155–162.

Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38(2).

Zobel, J., Moffat, A., and Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490.