Top-k Spatial Keyword Queries on Road Networks

João B. Rocha-Junior* and Kjetil Nørvåg
Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU), Trondheim, Norway
{joao,noervaag}@idi.ntnu.no

ABSTRACT

With the popularization of GPS-enabled devices there is an increasing interest for location-based queries. In this context, one interesting problem is processing top-k spatial keyword queries. Given a set of objects with a textual description (e.g., menu of a restaurant), a query location (latitude and longitude), and a set of query keywords, a top-k spatial keyword query returns the k best objects ranked in terms of both distance to the query location and textual relevance to the query keywords. So far, the research on this problem has assumed Euclidean space. In order to process such queries efficiently, spatio-textual indexes combining R-trees and inverted files are employed. However, for most real applications, the distance between the objects and query location is constrained by a road network (shortest path) and cannot be computed efficiently using R-trees. In this paper, we address, for the first time, the challenging problem of processing top-k spatial keyword queries on road networks where the distance between the query location and the spatial object is the shortest path. We formalize the new query type, and present novel indexing structures and algorithms that are able to process such queries efficiently. Finally, we perform an experimental evaluation that shows the efficiency of our approach.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Experimentation, Performance

Keywords

Top-k spatial keyword queries, road networks, indexing

1. INTRODUCTION

Top-k spatial keyword queries return the k best spatio-textual objects ranked in terms of both spatial proximity to the query location and textual relevance to the query keywords. Despite the wide range of location-based applications that can benefit from these queries, the current approaches for processing top-k spatial keyword queries are restricted to the Euclidean distance [3, 12, 18]. In this paper, we address, for the first time, the challenging problem of processing top-k spatial keyword queries on road networks. Given a set of spatio-textual objects (e.g., restaurants annotated with a text), a query location (latitude and longitude), and a set of query keywords, a top-k spatial keyword query on road networks returns the k best objects in terms of both 1) shortest path to the query location, and 2) textual relevance to the query keywords.

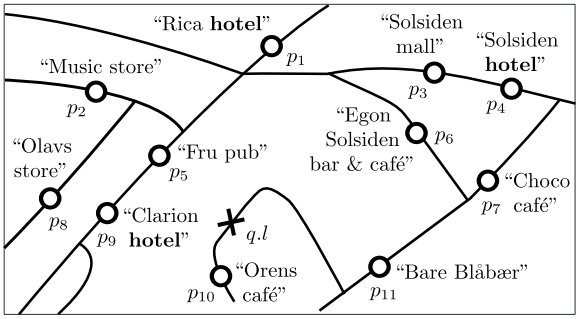


Figure 1: Top-k spatial keyword query on road networks.

For example, Figure 1 illustrates the road networks and spatio-textual objects in a tourist area of Trondheim, Norway. The circles represent spatio-textual objects p with a textual description, and the cross mark $q.l$ represents the query location. Assume a tourist in $q.l$ with a GPS-enabled mobile phone. The tourist poses a top-k spatial keyword query looking for "hotel" (his spatial location is automatically sent by the mobile phone). If a traditional query (Euclidean distance) is considered, the top-1 hotel is p_9 on the left side of the figure. However, when road networks are considered, the top-1 hotel is p_4 on the right side of the figure. In top-k spatial keyword queries on road networks both shortest path and textual relevance are considered. For example, for the query "bar café" posed in $q.l$, the spatio-textual object p_6 may appear better ranked than p_7 because the description of p_6 ("Egon Solsiden bar & café") is more textually relevant to the query keywords than the description of p_7 ("Choco café"), and p_6 is only slightly more distant to $q.l$ than p_7. The top-1 object, however, is p_{10} because it is very near to $q.l$ and is also relevant to the query keywords. Note that p_{11} is not returned as a result of this query, since none of the terms in the description of p_{11} appear in the query keywords.

Top-k spatial keyword queries on road networks can be employed by location-based applications to provide a more precise and realistic result. However, processing these queries is costly, since it

*On leave from the State University of Feira de Santana (UEFS).

EDBT 2012, March 26–30, 2012, Berlin, Germany.

requires computing several shortest paths.

To the best of our knowledge, processing top-k spatial keyword queries on road networks has never been proposed before. In this paper, we formalize the concepts of this new query type and describe how to rank objects considering both the network distance and the textual relevance. We also propose a basic approach for processing top-k spatial keyword queries on road networks combining the state-of-the-art approaches for road network and spatio-textual indexing. Then, we present an enhanced approach that employs inverted files to index the description of the spatio-textual objects lying on the road networks and, therefore, can process queries more efficiently. Finally, we describe how to create and employ an overlay on top of the actual road network to improve the query processing performance even further. The overlay allows to prune the regions of the network that cannot contribute with relevant objects. In order to identify the relevant regions, we compute an upper bound score for any object in the region in terms of both minimum network distance to the query location and maximum textual score. The maximum textual score of any object in a region is obtained through an abstract textual representation that is maintained for each region. We show the efficiency of our approach through an extensive experimental evaluation.

In summary, the main contributions of this paper are:

- We introduce top-k spatial keyword queries on road networks.
- We describe a basic approach for processing top-k spatial keyword queries on road networks combining state-of-the-art techniques.
- We propose an enhanced approach that indexes the description of the objects on a segment of the road network for efficient query processing.
- We employ an overlay network on top of the actual road network to prune regions that cannot contribute with relevant objects improving the query processing performance.
- Finally, we perform an experimental evaluation that demonstrates the efficiency of our approach.

The remainder of this paper is organized as follows. First, we describe the related work in Section 2. Next, we present the preliminaries and problem statement in Section 3. Then, we describe the basic approach in Section 4, the enhanced approach in Section 5, and the overlay approach in Section 6. Finally, we present the experimental evaluation in Section 7, and conclude the paper in Section 8.

2. RELATED WORK

Top-k spatial keyword queries on road network are related to keyword queries on relational databases [17] and data graphs [4]. However, in relational databases and data graphs, the addressed problem is finding rooted trees of connected vertices that are relevant for the query keywords.

There is also related work in the context of preference queries in road networks and relational databases. Mouratidis *et al.* [14] propose processing top-k and skyline queries on road networks assuming additional costs on the edges of the road networks. The result of the queries is the set of facilities that can be achieved from a given query location with minimum cost. Recently, Levandoski *et al.* [10, 11] propose a framework for integrating preference queries in database systems.

In the context of spatial objects on road networks [8, 9, 16], Papadias *et al.* [16] propose a framework to store road networks and spatial objects. They also propose algorithms to process nearest neighbor and range queries. One interesting result of this work is the observation that network expansion algorithms present better performance when compared with algorithms based on Euclidean distance heuristics. Recently, Lee *at al.* [8, 9] propose using route overlays to improve the performance of nearest neighbor and range queries on road networks. In contrast to their approach, we assume keyword and top-k queries.

In the context of spatial keyword queries. Ian de Felipe *et al.* [6] propose a new data structure that integrates signature files and R-trees. Each node of the R-tree employ a signature to indicate the keywords present in the node sub-tree. Zhang *et al.* [21] propose finding the m-closest objects to a given query location that match the set of m query keywords. Cao *et al.* [2] propose finding a group of objects that match the query keywords, minimizing intra-group distance, and the distance among the objects in the group and the query location. Different from our approach, these approaches are restricted to boolean keyword queries and Euclidean distance.

Finally, related work has been proposed in the context of top-k spatial keyword queries on spatial databases [3, 12, 18]. In top-k spatial keyword queries, the spatio-textual objects are ranked in terms of both spatial distance and textual relevance, where the distance between query location and the spatio-textual object is restricted to the Euclidean distance. Cong *et al.* [3] and Li *et al.* [12] propose augmenting the nodes of an R-tree with textual indexes such as inverted files. The inverted files are used to prune nodes that cannot contribute with relevant objects. Recently, Rocha-Junior *et al.* [18] propose an indexing structure that associates each term to a different data structure (block or aggregated R-tree) and can process top-k spatial keyword queries more efficiently. Finally, Wu *et al.* [20] cover the problem of keeping the result set of traditional spatial keyword queries updated, while the user is moving on a road network. Current approaches for processing top-k spatial keyword queries are restricted to Euclidean distance and rely on R-trees to compute the distance between the objects and the query location. Therefore, the techniques proposed cannot be applied in the context of road networks where the distance between the query location and the objects of interest is the shortest path.

3. PRELIMINARIES

In this section, we define the model used to represent the road networks, the set of objects of interest, and the problem statement.

Road networks. We model a road network as a graph $G = (V, E, W)$, where V is the set of vertices, E is the set of edges, and W is the set of weights (network distances) that are associated with each edge. A vertex $v \in V$ represents a road intersection or an end-point in the road network, an edge $(v, v') \in E$ represents a *road segment*, and the weight $w \in W$ associated with each edge (v, v') represents the length (network distance) $|v, v'|$ of the road segment. For simplicity, we assume bidirectional traffic. However, unidirectional traffic is also support by our approach. In this case, $(v, v') \neq (v', v)$ and the distance $|v, v'|$ may be different from $|v', v|$.

Objects of interest. We assume a set of spatio-textual objects $p \in P$ on the edges E of the road network G. Each object p has a spatial location $p.l$ and a textual description $p.d$. The size (cardinality) of the set of objects P is denoted as $|P|$. The network distance between an object p and the *ends of the edge* (vertices) in which it belongs is defined as $|v, p|$ or $|v', p|$, where v and v' are the ends of the edge (v, v') where p lies. The shortest path between two objects p and p' in the road network graph G is defined as $||p, p'||$. The set of objects in the edge (v, v') and (v', v) are the same, and the distance $|v, p|$ is equal to the distance $|v, v'| - |v', p|$. There-

fore, knowing the distance between p and one vertex is sufficient to obtain the distance between p and the other vertex. We denote *reference vertex*, the vertex used to compute the distance to the objects on the edge. Note that if unidirectional traffic is considered, the set of objects lying on edge (v, v') is different from the set of objects lying on the edge (v', v), and the distance $|v, p|$ may be different from $|v, v'| - |v', p|$.

Problem statement. We define a top-k spatial keyword query on road network as $Q_N = \langle q.l, q.d, q.k \rangle$ where $q.l$ is the query spatial location (latitude and longitude), $q.d$ is the set of query keywords, and $q.k$ is the number of results of interest. Without loss of generality, we assume that the query location $q.l$ lies on an edge of the road network. This assumption can be bypassed finding the nearest edge of a given query location. Given a query Q_N, a road network G, and a set of spatio-textual objects P; Q_N returns $q.k$ spatio-textual objects in descending order of score τ. The score $\tau(p)$ is defined as:

$$\tau(p) = \frac{\theta(p.d, q.d)}{1 + \alpha \cdot \delta(p.l, q.l)} \quad (1)$$

where $\delta(p.l, q.l)$ represents the network proximity between the query location $q.l$ and the object location $p.l$, and $\theta(q.d, p.d)$ represents the textual relevance of $p.d$ according to the query keywords $q.d$. The query parameter α is a positive real number ($\alpha \in \mathbb{R}^+$) and defines the importance of one measure over the other. For example, if $\alpha = 0$ only textual relevance is considered, and $\alpha > 1$ increases the importance of the network proximity over textual relevance.

The measure $\tau^*(p) = \alpha \cdot \delta(p.l, q.l) + (1 - \alpha) \cdot \theta(p.d, q.d)$ has been used for top-k spatial keyword queries [3, 12, 18]. We employ $\tau(p)$ instead of $\tau^*(p)$ to avoid normalizing the network distance, which is a requirement for using $\tau^*(p)$. In the context of road networks, normalizing the network distance requires computing the shortest path between any two points in the road networks, which is prohibitively costly to process in practice. However, our approach also supports $\tau^*(p)$.

Network proximity (δ) gives the importance of the location of a spatio-textual object $p.l$ to a query Q_N. Therefore, the network proximity is defined as:

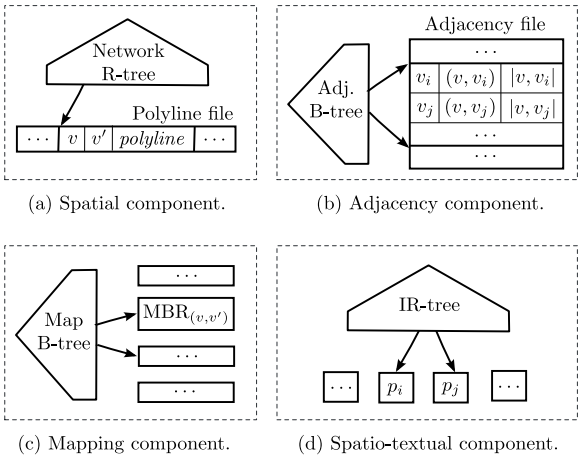
$$\delta(p.l, q.l) = ||p.l, q.l|| \quad (2)$$

which is the shortest path between p and q.

Textual relevance (θ) gives the importance of the textual description of a spatio-textual object $p.d$ to a query Q_N in terms of cosine similarity between $p.d$ and $q.d$ [22]. The textual relevance is defined as:

$$\theta(p.d, q.d) = \frac{\sum_{t \in q.d} w_{t,p.d} \cdot w_{t,q.d}}{\sqrt{\sum_{t \in p.d} (w_{t,p.d})^2 \cdot \sum_{t \in q.d} (w_{t,q.d})^2}} \quad (3)$$

the weight $w_{t,p.d} = 1 + ln(f_{t,p.d})$, where $f_{t,p.d}$ is the number of occurrences (frequency) of term t in $p.d$; and the weight $w_{t,q.d} = ln(1 + \frac{|P|}{df_t})$, where $|P|$ is the number of objects in the collection and df_t is the number of objects that contains t in their description (document frequency). We also define the *impact* $\lambda_{t,d}$ of a term t in d, where d is the *document* (text or description) of an object $p.d$ or the keywords in the query $q.d$. The *length of a document* is $W_d = \sqrt{\sum_{t \in d} (w_{t,d})^2}$. Hence, the impact is the normalized weight of the term in the document [1, 19], $\lambda_{t,d} = \frac{w_{t,d}}{W_d}$. The impact takes into account the length of the document and can be used to compare the relevance of two different documents according to a term t present in both documents. The textual relevance $\theta(p.d, q.d)$ can be rewritten in terms of the impact [19], $\theta(p.d, q.d) = \sum_{t \in q.d} \lambda_{t,q.d} \cdot \lambda_{t,p.d}$.



(a) Spatial component. (b) Adjacency component.

(c) Mapping component. (d) Spatio-textual component.

Figure 2: Basic indexing architecture.

Other relevance measures for textual relevance, such as Okapi BM25 [13], can be supported by our approach. However, in this paper, we focus on the efficient processing of top-k spatial keyword queries on road networks.

4. BASIC APPROACH

In this section, we present the basic approach to support top-k spatial keyword queries on road networks. We start presenting the indexing architecture in Section 4.1. Then, we present the query processing algorithm in Section 4.2.

4.1 Indexing Architecture

The basic indexing architecture combines a *spatio-textual index* [3, 12, 18], such as IR-tree, and the road network framework proposed by Papadias *et al.* [16]. The framework proposed by Papadias *et al.* permits starting a query at any location of the road network, while the spatio-textual index permits finding the objects in a given spatial region relevant for the query keywords.

Figure 2 presents the basic indexing architecture. The *spatial component* is used to identify the road segment in which the query location $q.l$ lies. The *adjacency component* permits retrieving the adjacent vertices of a given vertex, it is used to traverse the network. The *mapping component* is the connection between the network and the objects through the edge of the network. The mapping component maps an edge *id* (identification number) to the *Minimum Bounding Region* (MBR) that encloses the edge. Finally, the *spatio-textual component* stores the objects. In the following, we describe each component in more details.

Spatial component. The spatial component (Figure 2(a)) combines spatial and network connectivity information as proposed by Papadias *et al.* [16], and permits starting queries from any spatial location (latitude and longitude) on the road network. Each *road segment* (edge) is composed by a detailed polyline describing the edge. Each polyline is stored in an R*-tree named *Network R-tree*. In fact, only the MBR of the polyline is stored in the Network R-tree. The detailed polyline is stored in the *polyline file*. The polyline file stores also the *end vertices* of the edge. Locating the edge in which the query $q.l$ lies is executed in two steps: first, a point location query is performed in the Network R-tree for finding all MBRs (polylines) covering $q.l$, then, a filtering process is executed for selecting the exact polyline. This process permits finding

the edge where $q.l$ lies, and starting a network expansion from the *end vertices* of the edge. The Network R-tree is accessed only once during the query processing.

Adjacency component. The adjacency component (Figure 2(b)) points to the adjacent vertices of a given vertex permitting traversing the network from vertex to vertex. The adjacent B-tree points to the block in the adjacency file where the adjacent vertices of a given vertex v_i are. The *adjacency file* stores the *id* (e.g., (v, v_i)) of the edge, and the length of the edge (e.g., $|v, v_i|$). The *id* of the edge is used to retrieve the objects that lie on the edge through the mapping component.

Mapping component. The mapping component employs a B-tree named *Map B-tree* that maps an edge *id* to the MBR of the edge. Although not explicitly shown in the figure, the mapping component also points to the polyline of the edge. The MBR of the edge is used to find the spatio-textual objects lying on the edge through the spatio-textual component.

Spatio-textual component. The spatio-textual component can be any spatio-textual index that stores spatial and textual information of the objects [3, 12, 18]. Given the MBR enclosing the edge and the set of query keywords, the spatio-textual index is employed to retrieve the objects inside the MBR of the edge relevant for the query keywords. First, the spatio-textual index is accessed to retrieve all spatial objects inside the MBR enclosing the edge. Then, the polyline of the edge is used to prune the points that are inside the MBR of the edge, but not lying on the polyline of the edge. This is necessary because the MBR of an edge covers an area of the space that may contain objects belonging to other polylines.

Our basic indexing architecture preserves location and connectivity and can be employed to process top-k spatial keyword queries on road networks. In the next section, we present a query processing algorithm to process top-k spatial keyword queries on road networks employing the basic indexing architecture.

4.2 Query Processing

The basic query processing algorithm expands the adjacencies of a query location similarly to Dijkstra's algorithm [5]. The k best spatio-textual objects are maintained in a heap in decreasing order of score. The algorithm stops when the remaining objects cannot have a better score than the score of the *k-th* object already found, or the entire network has been expanded.

Algorithm 1 presents the basic query processing algorithm. The algorithm receives as parameter a query Q_N and returns the $q.k$ best objects in decreasing order of score $\tau(p)$. First, the Network R-tree is accessed to find the edge (v, v') in which $q.l$ lies (line 6). Then, the polyline of the edge is used to compute the network distances between $q.l$ and the *end vertices* v and v' of the edge (line 7). Next, the vertices v and v' are inserted into N in increasing order of network distance to $q.l$, and marked as visited (line 9). After that, the *findCandidates(ID*$_{(v,v')}$*, q.d, ϵ)* procedure is used to retrieve the spatio-textual *candidate objects* C laying on the edge (v, v') whose score $\tau(p)$ is higher than e (*k-th* score in $H^{q.k}$). ID(v, v') is the *id* of the edge (v, v'). Then, the heap $H^{q.k}$ is updated with the objects in C (line 10). As the insertion of a new object in $H^{q.k}$, the *k-th* object in $H^{q.k}$ is removed and ϵ receives the score of the new *k-th* object in $H^{q.k}$. The algorithm continues accessing the nearest vertex v to $q.l$ in N (line 11), and processing the non-visited adjacent vertices of v (lines 13-17). The algorithm terminates when the entire network is expanded ($N = \emptyset$), or the minimum network distance to any remaining object produces an *aggregated score* smaller or equals the score of the *k-th* object already found (line 12). The *aggregated score* $\tau^-(v)$ of a vertex v is obtained through the distance between v and $q.l$, and the maximum textual relevance ($\theta = 1$),

Algorithm 1 *BasicQueryProcessingAlgorithm(Query Q_N)*

1: **INPUT:** Top-k spatial keyword query on road networks, $Q_N = \langle q.l, q.d, q.k \rangle$.
2: **OUTPUT:** Reports the top-k objects found.
3: MaxHeap $H^{q.k} \leftarrow \emptyset$ *//$q.k$ best objects in decreasing order of τ.*
4: $\epsilon \leftarrow 0$ *//k-th score in $H^{q.k}$; While $|H^{q.k}| < q.k$, $\epsilon = 0$.*
5: MinHeap $N \leftarrow \emptyset$ *//vertices v in increasing order of $|v, q.l|$.*
6: $(v, v') \leftarrow$ network edge in which $q.l$ lies
7: compute $|v, q.l|$ and $|v', q.l|$ using the polyline of (v, v')
8: insert v and v' into N, mark (v, v') as visited
9: $C \leftarrow$ *findCandidates(ID*$_{(v,v')}$*, q.d, ϵ)*
10: update $H^{q.k}$ (and ϵ) with $p \in C$
11: $v \leftarrow N.pop()$ *//Vertice v in N with minimum $|v, q.l|$.*
12: **while** $v \neq \emptyset$ **and** $(\frac{1}{1+\alpha \cdot \delta(v.l, q.l)} \leq \epsilon)$ **do**
13: **for each** non-visited adjacent edge (v, v') of v **do**
14: $C \leftarrow$ *findCandidates(ID*$_{(v,v')}$*, q.d, ϵ)*
15: update $H^{q.k}$ (and ϵ) with $p \in C$
16: insert v' into N, mark (v, v') as visited
17: **end for**
18: $v \leftarrow N.pop()$
19: **end while**
20: **return** $H^{q.k}$

$\tau^-(v) = \frac{1}{1+\alpha \cdot \delta(v.l, q.l)}$. Therefore, if the aggregated score of a vertex is smaller or equals ϵ, it means that even if there is a non-visited object m with maximal textual relevance ($\theta(m.d, q.d) = 1$); the score of m is smaller or equal the score of the *k-th* object already found because $\delta(m.l, q.l) \geq \delta(v.l, q.l)$. This is guaranteed by the fact that the algorithm always expands the vertex v with minimum distance to $q.l$ (line 11 and 18). Consequently, the algorithm can safely terminate reporting the top-k objects found so far.

In order to find the candidate spatio-textual objects lying on the edge (v, v') that are relevant for the query keywords $q.d$, the *findCandidates(ID*$_{(v,v')}$*, q.d, ϵ)* procedure starts accessing the mapping component (Figure 2(c)) to obtain the MBR$_{(v,v')}$ that covers the edge (v, v'). The MBR$_{(v,v')}$ is used to perform a query in the spatio-textual index to obtain the objects inside the MBR$_{(v,v')}$ that are relevant for $q.d$. These relevant objects are, then, filtered by the polyline of the edge (v, v') to obtain the exact set of relevant objects that lie on the polyline of the edge. The polyline is also used to compute the network distance between the objects lying on the edge, and the *end vertices* of the edge. Finally, the score of the objects p lying on the edge are computed and compared with ϵ. Only the objects whose score is higher than ϵ are returned.

In order to simplify the presentation, we assume, in all examples, that the textual relevance θ is the number of occurrences of the query keywords in the description of an object $p.d$ divided by the number of keywords in the document. For example, the score of p_{10} ($p_{10}.d$ ={Orens, café}) for the query starting at location $q.l$ in Figure 3 with keywords $q.d$ ={café} is $\tau(p_{10}) = \frac{\theta(q.d, p_{10}.d)}{1+\delta(p_{10}.l, q.l)} = \frac{0.5}{2} = 0.25$. In the experimental evaluation section, we remove this assumption and employ the definitions presented in Section 3.

EXAMPLE. In this example, we show how to employ the basic query processing algorithm on the road network depicted in Figure 3 using the data presented in Figure 1. Assume a top-1 spatial keyword query on road networks at $q.l$ with $q.d$ ={café}. The algorithm starts accessing the Network R-tree and finding the edge (v_{11}, v_{12}) where $q.l$ lies (line 6). Then, the algorithm computes the distance $|v_{11}, q.l| = 2$ and $|v_{12}, q.l| = 8$ using the polyline of (v_{11}, v_{12}) (line 7). Next, v_{11} and v_{12} are added into N in increasing order of network distance to $q.l$ (line 8). After that, the *findCandidates* procedure is used to get the object lying on (v_{11}, v_{12}) with the keywords in $q.d$ (line 9) whose score is higher than ϵ. The pro-

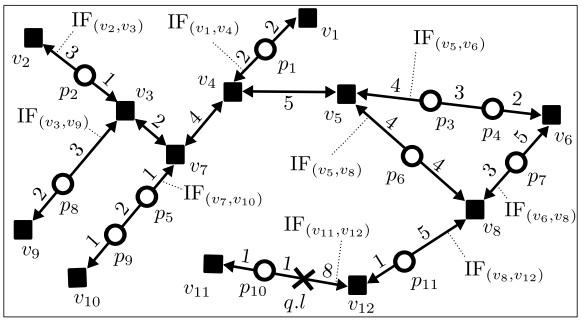


Figure 3: Graph representing the network and objects depicted in Figure 1.

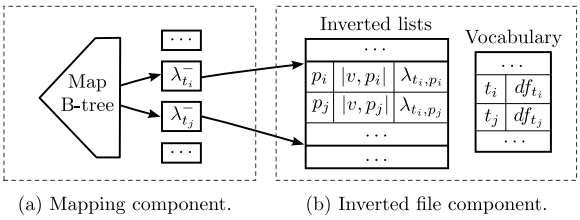


Figure 4: Enhanced Indexing architecture.

cedure returns p_{10} that is added into $H^{q.k}$ and ϵ is updated ($\epsilon = \frac{1}{4}$). The algorithm continues expanding the vertex v_{11} whose aggregate score $\tau^-(v_{11}) = \frac{1}{2}$ is higher than ϵ. Since there is no adjacent edge starting at v_{11} (line 13), the algorithm continues expanding the vertex v_{12} (line 18). The aggregated score of v_{12} ($\tau^-(v_{12}) = \frac{1}{9}$) is lower than ϵ (line 12), which causes the algorithm to finish (line 20) reporting p_{10} as top-1.

The basic query processing algorithm can be employed to process top-k spatial keyword queries on road networks. The main problem of this algorithm lies on the *findCandidates* procedure that is repeated for each adjacent edge. This procedure is expensive because it requires performing a search on a spatio-textual index, a filtering process for finding the relevant spatio-textual objects lying on a given polyline, and computing the network distance between the objects and the *end vertices* of the polyline.

5. ENHANCED APPROACH

In this section, we present an enhanced approach that indexes the objects lying on the edges of the road network for improving the query processing performance. First, we present the indexing architecture in Section 5.1. Next, we present the query processing algorithm in Section 5.2.

5.1 Enhanced Indexing Architecture

Figure 4 presents the new components employed in the enhanced indexing architecture. The mapping component depicted in Figure 4(a) replaces the mapping component presented in Figure 2(c), and the inverted file component shown in Figure 4(b) replaces the spatio-textual component shown in Figure 2(d). Similarly to the basic indexing architecture, the new mapping component is the connection between the network and the objects through the edge of the network. The inverted lists and vocabulary compose the *inverted file component*. In the following, we describe the new components.

Mapping component. The mapping component employs a B-tree named *map B-tree* that maps a key composed by the pair *edge id* and *term id* to the inverted list that contains the objects lying on the edge with term t in their description, see Figure 4(a). The mapping component contains also the *maximum impact* λ_t^- of a given term t among the description of the objects lying on a given edge. The *maximum impact* λ_t^- is an upper-bound impact for any object on the edge that contains t. Therefore, the inverted list of a term t on an edge is accessed only if the upper bound score composed by minimum distance and λ_t^- may turn an object, present in the edge, inside the top-k objects found so far.

Inverted file component. The inverted file component (Figure 4(b) is composed by inverted lists and vocabulary. The inverted file contains inverted lists identified by a key composed by *edge id* and *term id*. Each inverted list stores the objects lying on the edge (v, v') that have a term t in their description. For each object, the inverted list stores: 1) the network distance between the object and the reference vertex of the edge (e.g., $|v, p_i|$), and 2) the impact of the term t_i in the description of the object (e.g., λ_{t_i,p_i}). Figure 3 shows the inverted file $IF_{(v,v')}$ associated with each *populated edge* (edge that has at least one object). $IF_{(v,v')}$ is the set of inverted lists containing objects lying on the edge (v, v'). One list per different keyword in the description of the objects. The inverted lists are stored together for efficiency [22]. The Vocabulary file stores general information about each term t such as the document frequency df_t of each term. This information is used to compute the textual relevance of the object for a given query.

In the enhanced indexing, the objects lying on a given edge are stored in inverted files. Therefore, it does not require performing a search on a spatio-textual index for finding the spatio-textual objects relevant for the query that lie on the index. For each pair (term *id*, edge *id*), the inverted file keeps the objects lying on the edge with the given term. The network distance between the reference vertex of the edge and the object location is also stored in the inverted file. Consequently, the objects lying on a given edge relevant for a query keyword can be accessed directly boosting the performance of top-k spatial keyword queries on road networks.

Furthermore, the enhanced indexing keeps an upper-bound score for each pair (term id, edge id) that permits pruning edges whose the upper-bound score is smaller than the score of the k-*th* object found so far. This verification, thus, permits processing fewer edges terminating the query processing earlier.

5.2 Enhanced Query Processing

Processing top-k spatial keyword queries on road network employing the enhanced indexing architecture can be performed using the basic algorithm (Algorithm 1). The only, but significant, change lies on the *findCandidates* procedure.

The new *findCandidates* procedure employed in the enhanced approach works as follows. First, the mapping component (Figure 4(a)) is accessed to compute an upper-bound score using the maximum impact λ_t^- of a given term $t \in q.d$ and the minimum network distance between the edge and the query location. Second, if the upper-bound score is higher than ϵ, the inverted lists (one list per query keyword) are accessed. The lists that contain objects are retrieved and the objects whose scores are higher than ϵ are returned.

The *findCandidates* procedure on the enhanced approach is more efficient. First, only the lists that can produce relevant objects are accessed. Second, retrieving the inverted lists is faster than processing a query location for retrieving all objects inside a given MBR. Only relevant objects are retrieved since the key of the mapping component incorporates the query keyword. Third, it does not require a filtering process to remove the objects that are inside the MBR of the edge, but not on the polyline of the edge. Finally, it does not require computing the network distance between the ob-

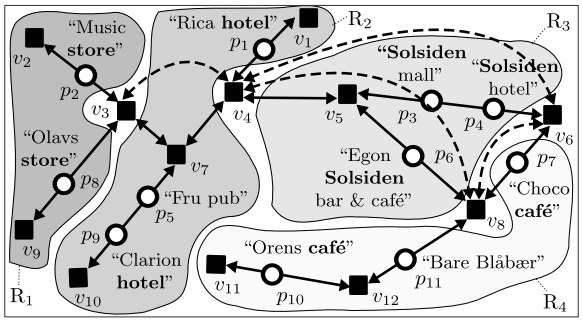


Figure 5: Single-layer overlay network.

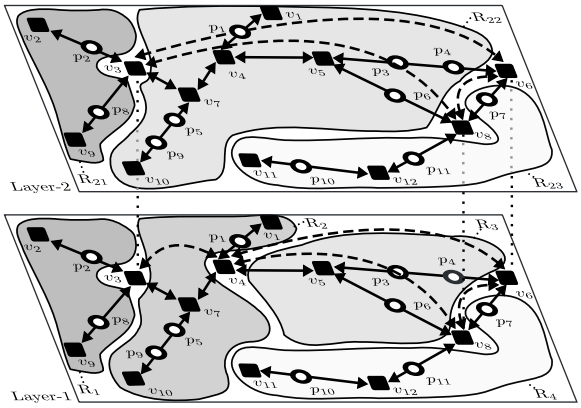


Figure 6: Multi-layer overlay network.

jects and the end vertices of the edges. The distances are computed and stored in the inverted lists during the index construction.

The enhanced query processing algorithm performs well when the network is populated, the query keywords are frequent, or the query preference parameter gives more weight to the network distance. In these cases, k objects with good scores are found rapidly, which permits the algorithm to terminate earlier. On the other hand, it can perform poorly if the k objects cannot be found rapidly, which can be common in top-k spatial keyword queries on road networks due to queries with non-popular terms, a large number of distinct terms in the datasets, or a sparse network. In the next section, we present how to overcome these problems employing an overlay network.

6. OVERLAY APPROACH

In this section, we describe how to employ an overlay network to improve the performance of the enhanced query processing algorithm. The main idea is to prune regions of the network that have a score smaller or equal to the score of the *k-th* object already found. Hence, fewer network regions are visited (expanded) and the algorithm terminates faster.

6.1 Overlay Network

Employing an overlay network to reduce the shortest path computation is not new [7–9]. In this paper, we propose a novel algorithm for creating an overlay network that takes into account the textual description of the objects lying on the segments of the road network. We construct the overlay network bottom-up aggregating edges into regions. The decision about which edges to aggregate is taken based on an aggregation cost. The aggregation cost takes into account the textual similarity between the region and the candidate edge, and the number of border vertices of the region.

For each region, we employ a vector term model of a pseudo-document representing the documents of all objects in the region. The *pseudo-document* is an abstract representation that permits summarizing the description of a group of documents [3, 12]. The pseudo-document of a region is composed by a set of tuples (term *id*, weight), where the weight is the maximum impact of the term among all objects in the region. The pseudo-document can be stored efficiently (one entry per term), and permits computing an upper bound score (term impact) for a term in the region.

The overlay network employs a direct connection among border vertices of the same region. A *border vertex* is a vertex that is in the border of two or more partitions. For road networks, the number of border vertices generated by an effective construction algorithm is far smaller than that of *regular vertices* (non-border vertices) [7]. A *direct connection* is a virtual edge connecting two border vertices in the same region. There is a direct connection when there is a path within the region between the two border vertices. The weight of the virtual edge is the shortest path, within the region, between the two border vertices. Therefore, once a region cannot contribute to the query in terms of textual relevance, the direct connection among the border vertices can be used to prune the region.

Figure 5 depicts a single-layer overlay network on top of the road network used in the Figures 1 and 3. The overlay contains four regions R_1, R_2, R_3, and R_4 whose the border vertices are v_3, v_4, v_6, and v_8. The pseudo-document representing the region R_1 is composed by the terms $R_1.d$ ={Music, store, Olavs}. The term $t =$ "store" appears in two documents p_2 and p_8. The vector representing R_1 keeps only the occurrence of t with highest impact. The directed connections are represented with dashed lines.

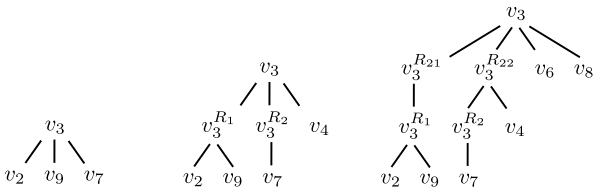
EXAMPLE. The overlay network improves the performance of top-k spatial keyword queries on road networks. Assume a query Q_N on the Figure 5 whose $q.l$ is on the edge (v_{11}, v_{12}) and $q.d =$ {pub}. From the border vertex v_8, it is possible to prune the entire region R_3 going directly to vertex v_4, since the term "pub" is not present in the pseudo-document (vector) of R_3.

Employing document similarity. We also propose an overlay construction method that employs document similarity. Hence, documents with similar content are stored in the same region, which increases the probability of pruning more regions during the query processing. The textual similarity among the objects in the same region improves the query processing performance even further. If objects with similar textual descriptions are put together in the same region, fewer regions will have occurrences of the same keyword.

EXAMPLE. In Figure 5, the terms in bold are those that appear in more than one object in the same region. Hence, if a top-k spatial query on road network looking for "Solsiden" is issued, all other regions except R_3 are pruned because R_3 is the only region with objects that have the keyword "Solsiden" in their description.

Multi-overlay network. The overlay network may be composed of multiple layers, where each layer corresponds to an abstract representation of the subsequent layer. Neighboring regions with similar content are grouped, which permits pruning even larger regions of the road network. The pseudo-document representing a grouped region is a superset of the pseudo-documents of the sub-regions.

The direct connections, border vertices, and pseudo-documents of one layer are sufficient to produce the subsequent layer [7, 9]. Figure 6 shows a multi-layer overlay network. The layer-2 is created using the information of layer-1. In this case, the regions R_2 and R_3 of the layer-1 are grouped to compose a new region R_{22} on layer-2. Consequently, the border vertices of the region R_{22} is a subset of the union of the border vertices of the regions R_2 and



(a) No overlay. (b) Single-overlay. (c) Multi-overlay.

Figure 7: Adjacency hierarchy of the border vertex v_3 for different overlay networks.

R_3, and the pseudo-document of R_{22} is a superset of the pseudo-document of regions R_2 and R_3. The direct connections used in layer-2 are derived from the direct connection in layer-1.

A multi-layer overlay network permits improving the processing of top-k spatial keyword queries on road networks. Assume a query Q_N on Figure 6 whose $q.l$ is on the edge (v_{11}, v_{12}) and $q.d$ ={Music}. From the border vertex v_8 is possible to prune the entire region R_{22} going directly to vertex v_3, since the term "Music" is not present in $R_{22}.d$.

6.2 Adjacency Hierarchy

The overlay network employs a hierarchy model to represent the adjacencies of the border vertices. The *adjacency hierarchy* of a border vertex v represents the adjacent vertices of v through an hierarchy so that it is possible to avoid regions that do not contain objects relevant for the query in terms of textual relevance. The adjacency hierarchy is obtained through *intermediary vertices* v^R labelled with the region R in which v is a border vertex. The region R is pruned if the best score produced by any object in R is smaller than the score of the *k-th* object already found. Therefore, the children entries of an intermediary vertex v^R are accessed only if R can contain relevant objects.

Figure 7 depicts the adjacency hierarchy of the border vertex v_3 for different overlay networks presented in Figure 6. Figure 7(a) shows the adjacency hierarchy of v_3 when no overlay network is considered which corresponds to the *actual adjacencies* of v_3. Figure 7(b) depicts the adjacency hierarchy of v_3 when a single-overlay network is considered (layer-1). In this case, the intermediary vertices $v_3^{R_1}$ and $v_3^{R_2}$ act as filtering vertices giving access to the children vertices only when the regions associated with the vertices are relevant for the query. For example, the children vertices v_2 and v_9 are accessed only when R_1 is relevant for the query. Finally, Figure 7(c) presents the adjacency hierarchy of v_3 for the multi-overlay network (Figure 6). In this case, the vertices $v_3^{R_{21}}$ and $v_3^{R_{22}}$ are able to prune more vertices. Each level in the adjacency hierarchy is built adding vertices on top of the previous hierarchy.

The adjacent vertices that are accessed through a direct connection are stored outside of the region because they may be accessed in the case where a region is not relevant. For example, in Figure 7(b), the adjacent vertex v_4 is accessed through a direct connection and is not stored as a child of $v_3^{R_2}$.

If one vertex v is adjacent to another vertex u through more than one direct connection, only the shortest direct connection is stored. This happens because the vertices are accessed in increasing order of distance to the query location during the query processing. For example, in Figure 5, v_8 is connected to v_6 through two direct connections (beyond the actual connection): one through region R_3 and another through region R_4. Since the direct connection is not part of a region, it is sufficient to keep the shortest direct connection through region R_4.

Algorithm 2 *OverlayQueryProcessingAlgorithm(Query Q_N)*

```
1: INPUT: Top-k spatial keyword query on road networks Q_N.
2: OUTPUT: Reports the top-k objects found.
3: Lines 3-10 of Algorithm 1 (BasicQueryProcessingAlgorithm)
4: v ← N.pop()                          //Vertex v in N with minimum |v, q.l|.
5: while v ≠ ∅ and ε < 1/(1+α·δ(v.l,q.l)) do
6:    for each non-visited adjacent vertex u of v do
7:       if u is an intermediary vertex then
8:          R ← region associated with the intermediary vertex u
9:          if ε < θ(R.d,q.d)/(1+α·δ(u.l,q.l)) then
10:            insert u into N
11:         end if
12:         mark u as visited
13:      else                                   //u is a regular vertex.
14:         Lines 14-16 of Algorithm 1
15:      end if
16:   end for
17:   v ← N.pop()
18: end while
19: return H^k
```

6.3 Overlay Query Processing Algorithm

The overlay query processing algorithm takes advantage of the overlay network to improve the performance of top-k spatial keyword queries on road networks. The idea is to avoid expanding the intermediary vertices that are not relevant for the query. Consequently, fewer vertices are expanded and the query processing terminates earlier.

Algorithm 2 presents the overlay query processing algorithm. The algorithm receives as parameter a query Q_N and returns the k best objects in decreasing order of score $\tau(p)$. The beginning of the algorithm is identical to the beginning of the basic query processing algorithm (Algorithm 1). If u is a regular vertex (line 7), the algorithm performs the same operation as Algorithm 1 (line 14). On the contrary, the pseudo-document $R.d$ of the region R associated with the intermediary vertex u is accessed (line 8), and the relevance of u is evaluated. The intermediary vertex u is relevant only if it can produce an aggregated score higher than the score of the *k-th* object already found (ϵ), otherwise it can be discarded (line 12). This is evaluated through the upper-bound score of u $\tau^-(u) = \frac{\theta(R.d,q.d)}{1+\alpha\cdot\delta(u.l,q.l)}$ that gives the best score that any object in the region R may have when R is accessed from the border vertex u. The network distance $\delta(u.l, q.l)$ is the minimum network distance between $q.l$ and any non-visited object in R; and the textual relevance $\theta(R.d, q.d)$ is the maximum impact for a term t among any object p in R (including the visited objects). This is guaranteed by the fact that the vertices are accessed in increasing order of distance to $q.l$. Therefore, when a border vertex of a region is accessed, it represents the minimum distance among any non-visited object in the region. Note that the region loses proximity relevance as new border vertices of the region are visited, since the distance between the query location and a new border vertex visited is longer or equal the distance between the query location and any previously visited border vertex of the region.

EXAMPLE. Assume a query Q_N whose $q.l$ is on the edge (v_{11}, v_{12}) and $q.d$ ={Music} (Figure 6). The algorithm starts expanding the edge (v_{11}, v_{12}) adding into N the vertices v_{11} and v_{12}. These vertices are expanded leading to the border vertex v_8. The direct adjacent vertices in the adjacency hierarchy of v_8 are $v_8^{R_{22}}, v_8^{R_{23}}, v_3$, and v_6. The vertices $v_8^{R_{22}}$ and $v_8^{R_{23}}$ are identified as intermediary vertices, and are not added into N because they are not relevant for the query keywords. The algorithm continues retrieving from N the nearest vertex v_6 whose adjacent vertices in

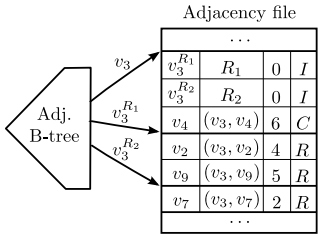


Figure 8: Example of the modified adjacency component storing the adjacency hierarchy of v_3 shown in Figure 7(b).

its adjacency hierarchy are $v_6^{R_{22}}, v_6^{R_{23}}, v_3$, and v_8. However, none of these vertices are added into N because $v_6^{R_{22}}$ and $v_6^{R_{23}}$ are not relevant, v_8 has been visited, and the nearest occurrence of v_3 to $q.l$ is already in N. The algorithm continues expanding the intermediary vertex v_3 whose adjacent vertices are $v_3^{R_{21}}, v_3^{R_{22}}, v_6$, and v_8. The vertices v_6 and v_8 have been visited and $v_3^{R_{22}}$ is not relevant because $\theta(R_{22}.d, q.d) = 0$. Hence, the vertex $v_3^{R_{21}}$ is added into N, which leads to expanding the vertices v_2 and v_9. The vertex v_2 is accessed and the relevant object p_2 is added in the top-k. The edge (v_3, v_9) is also accessed, but it does not have any relevant objects. Since N is empty, the algorithm terminates.

6.4 Overlay Indexing

We build the overlay indexing architecture on top of the enhanced indexing architecture (Section 5.1) with two modifications. The first modification is adding a B-tree named *region mapping B-tree* to map the edge or sub-region to the region it is part of. The region mapping B-tree is required to propagate updates in the edge (Section 6.4.2). The second modification is adding a column in the adjacency file (Figure 2(b)) to indicate the type of the adjacent vertex. For example, Figure 8 shows the adjacency hierarchy of the vertex v_3 (Figure 7(b)) stored in the new adjacency component. The *flag column* indicates the type of the relationship of the adjacent vertex that can be intermediary (*I*), direct connection (*C*), or regular vertex (R).

For an intermediary vertex, instead of storing the *id* of the edge, we employ the *id* of the region. The *id* of the region is created in the same universe of the edge *id*. Therefore, the vector of the pseudo-document associated with the region can be obtained through the mapping component (Figure 2(c)).

Intermediary vertices are not stored in the spatial component (Figure 2(a)) and the distance stored in the new adjacency file (Figure 2(b)) is zero. The intermediary vertices are virtual vertices whose the main role is creating the adjacency hierarchy structure. Therefore, they are not stored in the spatial component, but only in the adjacency component. Furthermore, the distance from the root vertex in the adjacency hierarchy and the intermediary vertex is zero, because the intermediary vertex is a representative (surrogate) of the root vertex in the hierarchy (Figure 8).

Similar to intermediary vertices, the direct connection vertices are not stored in the spatial component, but only in the new adjacency file. However, different from the intermediary vertices, the direct connection vertices have a weight (distance) that represents the shortest path between the two vertices.

In the following, we describe how to create the overlay index (Section 6.4.1), and briefly sketch how to perform updates (Section 6.4.2).

6.4.1 Index Construction

Our overlay index construction algorithm has three main objectives: 1) creating regions with similar number of objects, 2) creating regions with small number of border vertices, and 3) obtaining textual similarity among the objects in the same region.

Algorithm 3 *Overlay Construction(MinHeap H)*

```
1: INPUT: MinHeap H initialized with |R| seed regions in increasing order of |R.P|.
2: OUTPUT: Updates the adjacency and mapping components to reflect the new layer created.
3: while H ≠ ∅ do
4:   R_i ← H.pop()                    //Region R_i in H with smallest |R_i.P|.
5:   R_j ← ∅                          //Region R_j with smallest aggregation cost.
6:   for each v ∈ R_i.V do
7:     for each regular or intermediary adjacent vertex u of v do
8:       R_t ← region associated with u
9:       if R_t is not assigned to any other region then
10:        if R_j = ∅ or γ(R_i, R_t) < γ(R_i, R_j) then
11:          R_j ← R_t
12:        end if
13:      end if
14:    end for
15:  end for
16:  if R_j ≠ ∅ then                  //Expands R_i aggregating R_j.
17:    assign region R_j to R_i in the region mapping B-tree
18:    update R_i.d⃗ with R_j.d⃗
19:    update border vertices of R_i.V
20:    |R_i.P| ← |R_i.P| + |R_j.P|.
21:    insert R_i into H
22:  else                             //Store the region persistently.
23:    for each v ∈ R_i.V do
24:      U ← current adjacent vertices of v
25:      T ← ∅                        //New adjacent vertices of v.
26:      for each u in U do
27:        R_j ← region associated with u
28:        if v^{R_j} ∉ T then
29:          create v^{R_j}, and add v^{R_j} into T
30:        end if
31:        set u as child of v^{R_j}
32:        if v ∈ R_j.V then          //Computes direct edges.
33:          for each border vertex w in R_j.V do
34:            compute ||v, w|| and insert w into T
35:          end for
36:        end if
37:      end for
38:      store T as the new adjacent vertices of v
39:    end for
40:  end if
41: end while
```

The construction algorithm starts with a set of *seed regions* R_i that are expanded aggregating regions of the previous layer. In the case of layer=1 (first layer), the regions aggregated (layer=0) are the actual edges of the road network. The algorithm maintains the current border vertices $R_i.V$, the current number of objects in the region $|R_i.P|$, and the pseudo-document of the region $R_i.d$. The regions R_i are maintained in a MinHeap H in increasing order of $|R_i.P|$. The region R_i with smallest number of objects is expanded. All neighboring regions of $R_i.V$ not assigned to any other region are evaluated. The region with smallest *aggregation cost* is added into R_i. Once a region R_i cannot be further expanded, it is stored persistently and removed from H. The algorithm terminates when all regions have been fully expanded ($H = \emptyset$).

We define the aggregation cost $\gamma(R_i, R_j)$ of adding the region R_j into R_i as:

$$\gamma(R_i, R_j) = \frac{|R_i.V \cup R_j.V| - |R_i.V|}{1 + \beta \cdot \theta(R_i.d, R_j.d)} \quad (4)$$

where $|R_i.V \cup R_j.V|$ is the number of border vertices of R_i after adding R_j and $\theta(R_i.d, R_j.d)$ is the textual similarity between the

two regions. The smaller the aggregation score the better. The overlay construction parameter β permits tuning the trade-off between textual similarity between the regions, and the number of border vertices. The larger the number of border vertices, the higher the storage and construction cost. On the other hand, the higher the value of β, the higher the textual similarity between the regions.

Algorithm 3 presents the construction algorithm. It receives as parameter a heap with $|R|$ seed regions R_i in increasing order of number of objects $|R_i.P|$. For the construction of the first layer, each edge in the actual network corresponds to a region. The end vertices of the edge are the border vertices $R_i.V$, the number of object in the edge is $|R_i.P|$, and the pseudo-document of the edge is $R_i.d$. The seed regions are obtained from the Network R-tree. In this paper, we pick $|R|$ random edges from the Network R-tree to be the seed regions.

The construction algorithm is divided in three main parts. In the first part (lines 5-15), the algorithm finds the neighboring region R_j of R_i with smallest aggregation cost. In the second part (lines 17-21), the region R_i is expanded adding R_j. Finally, in the third part (lines 23-39), the region R_i is stored in the adjacency and mapping components (Figure 2).

First part. In order to find the neighboring region R_j of R_i with smallest aggregations cost, the border vertices $v \in R_i.V$ are visited (line 6). The intermediary or regular adjacent vertices u of v that are not assigned to any other region are evaluated (line 7). The temporary region R_t associated with u is obtained in the adjacency file through the region *id* or edge *id* for intermediary and regular vertices respectively. R_j keeps the region with smallest aggregation cost (lines 10-12).

Second part. If R_i can be expanded ($R_j \neq \emptyset$), R_j is added into R_i (lines 17-21). R_j is marked as part of R_i (line 17). Then, the *vector of the region* R_i ($\vec{R_i.d}$) is updated with the tuples in $\vec{R_j.d}$ (line 18). The tuples (term *id*, term impact) in $\vec{R_j.d}$ that are not present in $\vec{R_i.d}$ are added into $\vec{R_i.d}$, and the highest impact of a term that appears in both vectors is maintained in $\vec{R_i.d}$. The border vertices $R_i.V$ are updated (line 19). The border vertices in $R_j.V$ that are not in $R_i.V$ are added into $R_i.V$, and the border vertices in $R_i.V$ whose the adjacent vertices are assigned to R_i are removed from $R_i.V$, since they are no longer a border vertex of R_i (they are in the middle of the region). The number of objects in R_i is updated to include the objects in R_j (line 20), and R_i is added into H to be further expanded (line 21).

Third part. The regions that cannot be expanded (line 15) are persistently stored in the adjacency and mapping components (lines 23-41). The algorithm accesses the border vertices v of R_i (line 23) and gets the current adjacent vertices U of v (line 24). The current adjacent vertices of v are the vertex stored in layer immediately bellow the layer that is being constructed. Then, for each adjacent vertex $u \in U$, the region R_j associated with u is obtained (line 27). The intermediary vertex v^{R_j} is created if not in the new set of adjacent vertices T (line 29), and u is set as children of v^{R_j} (line 31). The direct connections from v to the other border vertices w in $R_j.V$ are computed and the vertex w is added into T (lines 32-36). Finally, T is stored persistently in the adjacency component (line 38). Although not explicitly shown in the algorithm, the borders of each region are also stored. This information is require on the construction of the sub-sequent layer.

One important parameter of the algorithm is the number of seed regions (starting regions) that should be used during the creation of each layer. We employ the following equation to compute the number of seed regions used at each layer l: $\lceil {}^{|P|}/_{b^l} \rceil$, where b is the average number of items (objects or sub-regions) that are stored at each region. The maximum number of layers $l_{max} = \lfloor log_b(|P|) \rfloor$. The layer l is a number in the range $[1, l_{max}]$. For example, assuming a dataset with 1M objects ($|P| = 1M$) and an average of 100 items per region. Thus, $l_{max} = log_{100}(1M) = 3$, and the number of regions at layer $l = 1$ is $\lceil {}^{1M}/_{100^1} \rceil = 10K$ and for $l = 2$ the number of regions is 100.

6.4.2 Index Maintenance

For most applications that can benefit from top-k spatial keyword queries on road network, updates in the textual description of the objects are, in general, more frequent than updates in the road network and spatial location of the objects. Due to lack of space, we focus on updates in the textual description of the objects.

Given an object p whose previous text $p.d$ is to be updated with the new text $p.d^-$. The update procedure starts locating the edge e in which p lies accessing the Network R-tree. Then, the *id* of e is used to obtain the vector $\vec{e.d}$ (or $e.d$) in the mapping component. Next, for terms t_i that are in $p.d$ but not in $p.d^-$, p is removed from the inverted list of t_i. For terms t_j that are in $p.d^-$ but not in $p.d$, p is inserted into the inverted list of t_j. For terms t that are in both $p.d$ and $p.d^-$, the impact of t in $p.d$ ($\lambda_{t,p.d}$) is updated receiving $\lambda_{t,p.d^-}$. The impact of t in $e.d$ is also updated to the impact of $\lambda_{t,p.d^-}$ when $\lambda_{t,p.d^-} > \lambda_{t,e.d}$. The impact of t in $e.d$ is recomputed using the inverted list of t, when $\lambda_{t,e.d} = \lambda_{t,p.d}$ and p was removed from the inverted list of t or $\lambda_{t_j,p.d^-} < \lambda_{t_j,p.d}$.

In case of multi-layer overlay networks, the updates in the impact of the vector $\vec{e.d}$ are propagated to the regions that encloses e. Therefore, the same procedure to update the vector $\vec{e.d}$ of the edge is performed and the changes in $\lambda_{t,e.d}$ are propagated to the regions that enclose e. The regions that enclose e are obtained through the region mapping B-tree.

7. EXPERIMENTAL EVALUATION

In this section, we compare the three approaches proposed Basic (Section 4), Enhanced (Section 5), and Overlay (Section 6). The three approaches return the same set of top-k objects for a given query. We employed real and synthetic datasets in the experimental evaluation. All approaches were implemented in Java. We employ the R*-tree and B-tree from XXL[1] and JDBM[2] libraries respectively. The nodes of the Network R-tree employ a block size of 4kB. Each node stores between 34 and 102 entries (polyline MBRs). We employ the spatio-textual index S2I [18] in the basic approach. The nodes of the aggregated R-tree [15], that store the objects containing frequent terms, employ a block size of 4kB. Each node stores between 42 and 85 entries. The blocks that store the objects containing non-frequent terms employ a block size of 4kB. Each block stores up to 146 entries.

Setup. The experiments were executed on a PC with a 2.6GHz AMD processor and 32GB RAM. During the query processing, we fixed the java maximum memory parameter in 4GB (-Xmx4G). In the query processing experiments, we execute 100 queries in order to warm-up the buffers, and collect the average results of the next 400 queries. The queries are randomly generated. The query location is a random coordinate on a random edge (polyline) of the road network; while the query keywords are random terms extracted from the vocabulary of the dataset. We employed a fixed buffer of 8MB in all approaches. In the experiments, we measured 1) *response time* (total execution time); 2) *edges expanded*, i.e., the number of edges expanded by the query processing algorithm before finding the top-k results (includes also the number of regions

[1]http://dbs.mathematik.uni-marburg.de/Home/Research/Projects
[2]http://jdbm.sourceforge.net

Table 1: Parameters evaluated in the experiments. The default values are presented in bold.

Parameter	Values
Number of results (k)	**10**, 20, 30, 40, 50
Number of keywords	1, 2, **3**, 4, 5
Query parameter α	0.01, 0.1, **1**, 10, 100
Construction parameter β	0.01, 0.1, 1, **10**, 100
Average region cardinality	10, **20**, 30, 40, 50
Number of layers	1, **2**, 3
Real datasets	London, **Australia**, British Isles
Synthetic datasets	K1, K2, K3, K4, C1, C2, C3, C4

Table 2: Characteristics of the datasets.

Attribute	London	Australia	British Isles
Total size	51MB	560MB	1.3GB
Total no. of vertices	203,383	1,181,142	3,556,460
Total no. of edges	274,947	1,631,421	4,618,215
Avg. no. of lines per edge	5.79	13.65	10.00
Avg. edge length (m)	105.12	740.47	227.3
Total no. of objects	34,162	69,884	298,368
Avg. no. of objects per edge	0.12	0.04	0.06
Total no. of words	121,049	225,865	1,035,857
Total no. of distinct words	12,551	18,875	60,154
Avg. no. of distinct words per object	3.35	3.04	3.26

expanded in case of overlay approach); 3) *edges processed*, i.e., the number of edges whose objects (lying on the edge) are retrieved for identifying their relevance to the query; 4) *index construction time* (time to build the index); and 5) *index size*. Table 1 shows the main parameters and values used through the experiments. The default values are presented in bold.

Real datasets. We employ three real datasets extracted from OpenStreetMap[3]. The datasets are London, Australia, and British Isles. Each dataset was extracted using a rectangle enclosing the region of interest. For the London dataset, we employ the rectangle [(-0.449,51.342),(0.206, 51.649)], where the first coordinate (latitude, longitude) represents the bottom-left corner, and the second coordinate represents the up-right corner. The rectangles used to extract Australia and British Isles datasets are respectively [(112.2, -44.2),(154.8, -9.4)] and [(-11.1, 49.6),(2.1,62.5)]. The datasets extracted from OpenStreetMap have several partitions; most of them are regions in the map describing a building or a spatial area. For simplicity, we consider only the road network formed by the largest partition of each dataset. Furthermore, we allocate the objects that do not lie on a edge to the nearest edge in the road network. Table 2 presents some characteristics of each dataset[4].

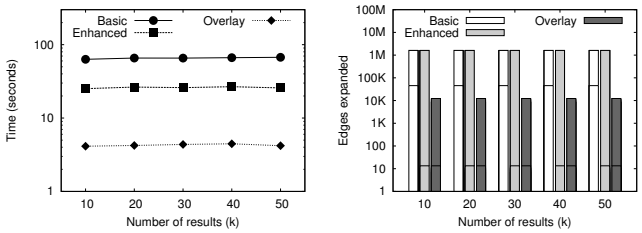
Synthetic datasets. The synthetic datasets were obtained combining the Australia dataset with Twitter[5] messages (tweets). We created two sets of synthetic datasets. The first set was obtained replacing the text of the objects in Australia dataset with tweets from Twitter[6]. We preserved the road network and the location of the objects in the Australia dataset to create four datasets named K1, K2, K3, and K4. The description of the objects in K1, K2, K3, and K4 datasets is composed by 1, 2, 3, and 4 tweets respectively. The average number of distinct words per object in K1, K2, K3, and K4 datasets is respectively 12.3, 23.6, 33.7 and 42.5. The second set of synthetic datasets was obtained populating the Australia network with objects. The description of the objects is obtained from Twitter, one tweet per object. Each object is randomly positioned on a random edge of the network. We created four datasets named C1, C2, C3, and C4. The *cardinality* (number of objects) in C1, C2, C3, and C4 datasets is respectively 250k, 500k, 750k, and 1M. We employed distinct tweets to construct the datasets.

[3]http://www.openstreetmap.org/

[4]The real datasets employed in the experiments are available in: http://www.idi.ntnu.no/~joao/publications/EDBT2012/

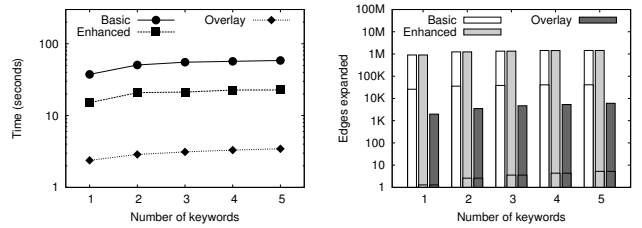
[5]http://twitter.com

[6]http://snap.stanford.edu/data/twitter7.html



(a) Response time. (b) Edges expanded.

Figure 9: Response time and number of edges expanded varying the number of results (k).



(a) Response time. (b) Edges expanded.

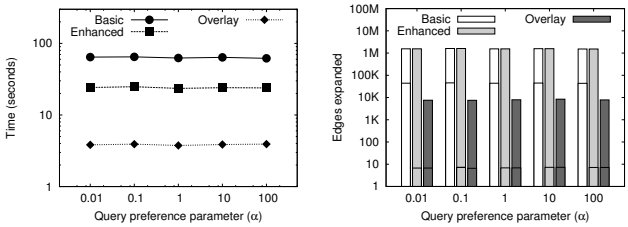
Figure 10: Response time and number of edges expanded varying the number of keywords.

7.1 Experiments on Real Datasets

In this section, we evaluate the query processing performance of *basic* (Section 4), *enhanced* (Section 5), and *overlay* (Section 6) approaches on real datasets. The y-axis is in logarithmic scale, except in Figure 12(a).

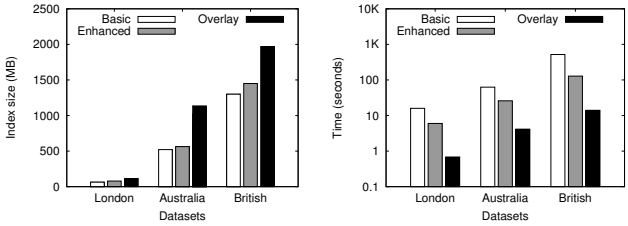
Varying the number of results (k). Figure 9 depicts the response time and the number of edges expanded, while varying k. Both enhanced and overlay have better response time than the basic approach (Figure 9(a)). The enhanced and basic approaches expand the same number of edges (Figure 9(b)), since they do not have a global view of the network. However, the number of edges processed by the enhanced approach is much smaller than the number of edges processed by the basic approach. The *mark on the bar* indicates the number of edges processed that is a percentage of the total number of edges expanded. In Figure 9(b), for example, the number of edges processed for $k = 10$ is approximately 15 for the enhanced and 70K for the basic approach. The enhanced approach only processes edges that have objects that can be in the top-k; while the basic processes all the edges that have objects, even if the objects are not relevant for the query. The overlay approach, on the other hand, expands much fewer edges than basic and enhanced approaches. The number of edges processed by overlay and enhanced approaches is similar. Consequently, the overlay approach is more than one order of magnitude better than the basic approach in response time, edges expanded, and edges processed.

Varying the number of keywords. Figure 10 presents the response time and number of edges expanded, while varying the num-



(a) Response time. (b) Edges expanded.

Figure 11: Response time and number of edges expanded varying the query preference parameter (α).



(a) Index size. (b) Response time.

Figure 12: Index size and response time varying the datasets.

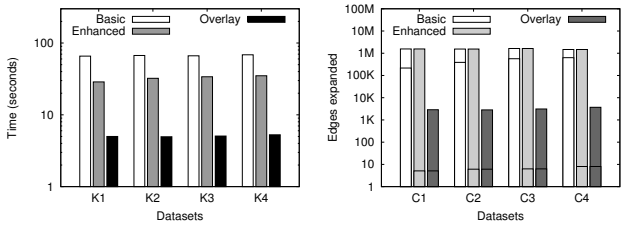
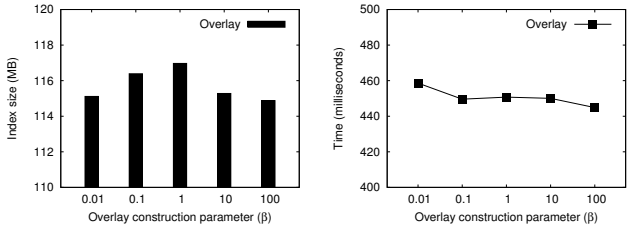


Figure 13: Response time *keyword datasets*.

Figure 14: Edges expanded for *cardinality datasets*.



(a) Index size. (b) Response time.

Figure 15: Index size and response time varying the overlay construction parameter (β).

ber of keywords in the query. The larger the number of keywords in the query, the larger the number of objects that may be relevant for the query. Again, the overlay approach is around one order of magnitude better in response time than enhanced and basic (Figure 10(a)). Figure 10(b) depicts the number of edges expanded and the number of edges processed (mark on the bar). The number of edges processed by overlay and enhanced is very small for queries with few keywords, and increases for queries with more keywords. However, it is much smaller than the number of edges processed by the basic approach. Note that one single edge of the road network may contain several objects.

Varying the query preference parameter (α). In this experiment, we evaluate the impact of the query preference parameter as illustrated in Figure 11. A small value of α gives more preference to the textual description of the objects, while a high value of α gives more preference to the network proximity. The query preference parameter does not present a significant impact on response time (Figure 11(a)) and number of edges expanded (Figure 11(b)). All approaches are slight better for higher values of α, which means more preference to network proximity.

Varying the datasets. In this experiment, we study the index size and response time for different real datasets, as shown in Figure 12. Figure 12(a) presents the index size for the different approaches. The index sizes of the enhanced and basic approaches are similar, with a small advantage for the basic approach. However, the index size of the overlay approach is larger. The additional size required by the overlay approach comes from the additional number of vertices (border vertices), edges (direct connections), and the pseudo-document of the regions. The additional size required by the overlay approach is compensated by a significant improvement in response time (Figure 12(b)).

7.2 Experiments on Synthetic Datasets

In this section, we employ synthetic datasets to evaluate the impact of increasing the number of keywords per object and the number of objects (cardinality) on the road network.

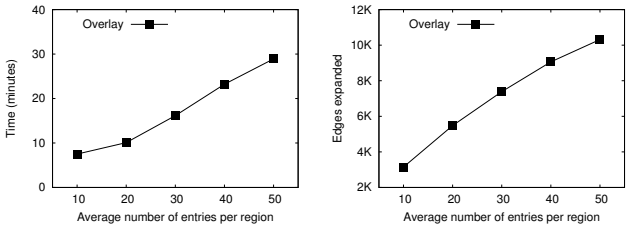
Varying the number of keywords per object. Figure 13 shows the response time for varying the number of keywords in the description of the objects (textual description length). The basic approach is not affected by increasing the description of the objects. The high cost of the basic approach is in the processing of the edges. Since the number of objects in the datasets does not vary, the number of edges processed is the same for all datasets. However, increasing the description of the objects has an impact on the response time of the enhanced and overlay approaches. The main reason is that it becomes more costly to identify the edges that have relevant objects, since more objects can be textually relevant for the query. Consequently, more edges are processed. Despite this, the response time of the overlay approach is more than one order of magnitude better.

Cardinality. Figure 14 shows the number of edges expanded and processed (mark on the bar) for varying the cardinality of the datasets. Increasing the number of objects has a significant impact on the basic approach, since it increases the number of populated edges. All expanded populated edges are processed by the basic approach. In the case of enhanced and overlay approaches, only the expanded populated edges that have objects that can be in the top-k are processed.

7.3 Overlay Network

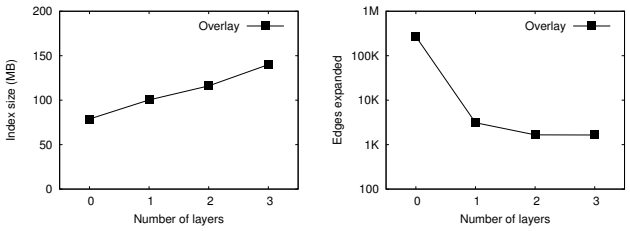
In this section we evaluate the trade-off between the overlay construction cost and the gain in query performance for different setups. The evaluation is performed employing the London dataset.

Varying overlay construction parameter (β). In this experiment, we study the advantage of employing text similarity in the overlay construction. Figure 15 shows the index size and the response time while varying β. The higher the value of β, the higher the impact of textual similarity in the grouping of regions during the overlay index construction. Figure 15(a) shows the overlay index size for varying β. The index is small for small and high values of β. It means that incorporating textual similarity or distance reduces the index size because the regions created have smaller number of borders. However, only the textual similarity has a positive impact on the response time (Figure 15(b)). For small values of β, the in-



(a) Overlay construction time. (b) Edges expanded.

Figure 16: Overlay construction time and number of edges expanded during the query processing while varying the average number of entries per region.



(a) Overlay index size. (b) Edges expanded.

Figure 17: Overlay store cost and the number of edges expanded varying the average number of layers.

dex construction gives more priority to network proximity instead of the textual similarity among the objects. Therefore, the regions created when β is small contain objects whose text description is dissimilar, which impacts on the response time. This experiments demonstrates the advantage of incorporating text similarity during the index construction.

Varying the number of entries per region. Figure 16 shows the overlay index construction time and the response time when varying the number of entries (edges or other regions) per region. If the average number of entries per region is 10, it means that each region at level one has approximately 10 edges; and each region at level two has approximately 10 other regions (around 100 edges); and so on. Figure 16(a) shows that increasing the number of entries per region has a high impact on the index construction time. When the number of entries increases, the number of border vertices also increases. The number of border vertices has a direct impact on the index construction, because more border vertices require storing and processing more direct edges. Figure 16(b) presents the impact of the number of entries per region in the number of edges expanded during the query processing. The smaller the number of entries, the smaller the number of edges expanded during the query processing. This experiment shows the efficiency of the overlay approach. Smaller number of entries indicates that the query processing finds faster an intermediary vertex that permits pruning a large number of irrelevant regions reducing the response time.

Varying the number of layers. In this experiment, we study the impact of the number of layers on the index size and number of edges expanded, Figure 17. The number of layers equals zero indicates that no overlay index is employed (enhanced index). The larger the number of layers, the larger the index size, Figure 17(a). The experiment shows that employing a single layer is sufficient to reduce the number of edges expanded significantly, Figure 17(b).

8. CONCLUSION

In this paper, we introduced top-k spatial keyword queries on road networks. Given a spatial location and a set of query keywords; a top-k spatial keyword query on road networks returns the k best spatio-textual objects ranked in terms of both textual similarity to the query keywords and shortest path to the query location. We presented a straight-forward approach (basic approach) to process these queries combining state-of-the-art techniques. Then, we presented an enhanced approach that indexes the edges of the road network, and permits identifying and retrieving the objects relevant to the query efficiently. Finally, we proposed an overlay approach that groups objects in regions, taking in account the textual similarity among the objects, and permits computing an upper-bound score for all objects in the region. Consequently, regions whose the upper-bound score is smaller or equal the score of the *k-th* object already found can be pruned, improving the performance. Finally, we demonstrated the efficiency of our approach through an extensive experimental evaluation.

9. REFERENCES

[1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *SIGIR*, 2001.

[2] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, 2011.

[3] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *VLDB*, 2009.

[4] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. In *VLDB*, 2008.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 1959.

[6] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, 2008.

[7] N. Jing, Y. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Trans. on Knowledge and Data Engineering*, 10(3), 1998.

[8] K. C. Lee, W. Lee, B. Zheng, and Y. Tian. ROAD: a new spatial object search framework for road networks. *IEEE Trans. on Knowledge and Data Engineering*, to appear.

[9] K. C. K. Lee, W. Lee, and B. Zheng. Fast object search on road networks. In *EDBT*, 2009.

[10] J. J. Levandoski, M. E. Khalefa, and M. F. Mokbel. An overview of the caredb context and preference-aware database system. *IEEE Data Eng. Bull.*, 34(2), 2011.

[11] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa. Flexpref: A framework for extensible preference evaluation in database systems. In *ICDE*, 2010.

[12] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *IEEE Trans. on Knowledge and Data Engineering*, 99(4), 2010.

[13] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[14] K. Mouratidis, Y. Lin, and M. L. Yiu. Preference queries in large multi-cost transportation networks. In *ICDE*, 2010.

[15] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *SSTD*, 2001.

[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.

[17] J. Park and S. Lee. Keyword search in relational databases. *Knowledge and Information Systems*, 26(2), 2010.

[18] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, 2011.

[19] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5), 1988.

[20] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, 2011.

[21] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, 2009.

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