National Technical University of Athens

School of Electrical and
Computer Engineering

Knowledge and Database Systems Laboratory

Geospatial Dataspaces: Location-based services and integration of User-Generated Content

PhD Thesis

of

Christodoulos Efstathiades

BSc in Computer Science (UCY 2009)
MSc in Networked Computer Systems (UCL 2010)

Athens, May 2016

National Technical University of Athens
School of Electrical and Computer Engineering
Knowledge and Database Systems Laboratory

Geospatial Dataspaces: Location-based services and integration of User-Generated Content

PhD Thesis

of

Christodoulos Efstathiades

BSc in Computer Science (UCY 2009)

MSc in Networked Computer Systems (UCL 2010)

Supervising Committee: I. Vassiliou
T. Sellis
D. Pfoser

Approved by the Examination Committee, 18th May 2016.

...	...	...
I. Vassiliou	T. Sellis	D. Pfoser
Prof. NTUA	Prof. RMIT	Assoc. Prof. GMU
...	...	...
K. Kontogiannis	I. Stavrakas	N. Koziris
Assoc. Prof. NTUA	Researcher Athena RC	Prof. NTUA
	...	
	Y. Theodoridis	
	Prof. UNIPI	

Athens, May 2016

. . .

Christodoulos Efstathiades
BSc in Computer Science (UCY 2009)
MSc in Networked Computer Systems (UCL 2010)

ABSTRACT

The technological advances during the past years have resulted in the change of how people use and interact with the internet. Novel Web technologies and resulting applications have lead to a participatory data ecosystem that when utilized properly will lead to more rewarding services. People have been using cell phones for social interaction, willingly producing vast amounts of information. This, in addition to the technological ability of cell phones to track locations have lead to the consideration of advanced location-based services that take into account not only the location but all of the user-generated data that is produced implicitly or explicitly. In this PhD thesis, we investigate the case of this kind of Location-based Services and specifically of how to improve the typical location-based queries with user-generated content taken from sources in the web in the form of texts. Towards this goal we have focused our research on three main directions: (a)Proposal of access methods that take advantage of the additional information taken from user-generated data, (b) proposal of new location-based queries on top of the proposed access methods that leverage the use of crowd-sourced information, (c) efficiency of the execution of the queries leveraging the state of the art in geo-spatial algorithms, graph algorithms and research on set similarity. An extensive literature review has been made in for an in-depth research on the areas of interest towards this PhD thesis. New access methods and query processing algorithms have been proposed in order to support location-based services that are enriched with user-generated content that is extracted from web platforms. Experimental evaluation of the proposed algorithms and techniques prove their efficiency. A framework that works entirely within the database in order to support distance queries on large-scale graphs is proposed. Finally, we present a framework for the collection, analysis, querying and visualisation of social network data, that aims in the support for the proposed enhanced location-based services.

Key words: User generated content, spatial databases, spatio-textual queries, knn, similarity search, shortest paths, graph theory

ΠΕΡΙΛΗΨΗ

Οι τεχνολογικές εξελίξεις κατά τη διάρκεια των τελευταίων χρόνων έχουν οδηγήσει στην αλλαγή στον τρόπο με τον οποίο οι άνθρωποι χρησιμοποιούν και να αλληλεπιδρούν με το διαδίκτυο. Οι νέες τεχνολογίες ιστού καθώς και οι εφαρμογές που έχουν προκύψει, οδηγούν σε ένα οικοσύστημα το οποίο όταν χρησιμοποιηθεί σωστά, θα οδηγεί σε χρήσιμες στον άνθρωπο υπηρεσίες. Η χρήση των κινητών τηλεφώνων για την κοινωνική αλληλεπίδραση, οδηγεί στην παραγωγή τεράστιων ποσοτήτων δεδομένων. Αυτό, σε συνδυασμό με την ικανότητα των κινητών τηλεφώνων να ανιχνεύουν τη γεωγραφική τους θέση, οδηγεί στην αναζήτηση πιο προηγμένων υπηρεσιών που λαμβάνουν υπόψη όχι μόνο τη γεωγραφική θέση αλλά και το σύνολο των δεδομένων που δημιουργούνται από τους χρήστες. Σε αυτή την διδακτορική διατριβή, μελετούμε την περίπτωση αυτού του είδους των Υπηρεσιών Θέσης και συγκεκριμένα το πώς μπορούν να βελτιωθούν τα τυπικά ερωτήματα υπηρεσιών θέσης με περιεχόμενο που προέρχεται από πηγές στο διαδίκτυο μέσω των χρηστών, με τη μορφή των κειμένων. Προς την κατεύθυνση αυτή έχουμε εστιάσει την έρευνά μας σε τρεις κύριες κατευθύνσεις: (α) Πρόταση μεθόδων ευρετηρίασης που επωφελούνται από τις πρόσθετες πληροφορίες μέσω του περιεχομένου, (β) πρόταση νέων ερωτημάτων και αλγορίθμων για βελτιωμένες υπηρεσίες θέσης, που εκμεταλλεύονται τις νέες μεθόδους ευρετηρίασης, και (γ) την αποτελεσματικότητα της εκτέλεσης των ερωτημάτων αξιοποιώντας την πιο πρόσφατη έρευνα σε αλγόριθμους εγγύτητας, αλγόριθμους για την εύρεση κοντινότερων μονοπατιών σε γράφους και την έρευνα σχετικά με ομοιότητα συνόλων. Μια εκτενής ανασκόπηση της βιβλιογραφίας έχει πραγματοποιηθεί για μια σε βάθος έρευνα σχετικά με τις περιοχές ενδιαφέροντος που αφορούν αυτή τη διδακτορική διατριβή. Νέες μέθοδοι πρόσβασης και αλγόριθμοι επεξεργασίας επερωτήσεων έχουν προταθεί για να υποστηρίξουν υπηρεσίες θέσης που είναι εμπλουτισμένες με περιεχόμενο που παράγεται από χρήστες και εξάγεται από διαδικτυακές πλατφόρμες. Η πειραματική αξιολόγηση των προτεινόμενων αλγορίθμων και τεχνικών αποδεικνύουν την αποτελεσματικότητά τους. Προτείνεται επίσης ένα σύστημα που λειτουργεί εξ΄ ολοκλήρου εντός της βάσης δεδομένων, προκειμένου να στηρίξει ερωτήματα απόστασης σε γράφους μεγάλης κλίμακας. Τέλος, παρουσιάζουμε ένα σύστημα για τη συλλογή, ανάλυση, υποβολή επερωτήσεων και την οπτικοποίηση των δεδομένων των κοινωνικών δικτύων, που στοχεύει στην υποστήριξη των προτεινόμενων από αυτή τη διατριβή υπηρεσιών θέσης.

ACKNOWLEDGMENTS

I offer my sincerest gratitude to my supervising PhD committee Yannis Vassiliou, Timos Sellis and Dieter Pfoser. They have provided me with the appropriate guidance and support throughout the years of my research, especially during times when things seemed to have come to a dead end. Dieter Pfoser, with whose guidance I have conducted this PhD Thesis has been an excellent supervisor and has helped me tremendously with his experience, trust and patience. The thesis would not have been possible without his help and guidance. I thank him for always finding time in reviewing my work and giving me valuable advices. Timos Sellis has provided me with the appropriate funding in order for me to be able to conduct this thesis. His willingness to help with any problems and concerns that I had is admirable and I thank him for being an inspiration to me throughout all the years of my PhD. Yannis Vassiliou gave me the chance to teach for three consecutive years at NTUA, providing me with the trigger to pursue a work that involves teaching. In addition, especially during the final year of my PhD he was always willing to help when I needed him.

I would like to thank Prof. Kostas Kontogiannis, Dr. Yiannis Stavrakas, Prof. Nectarios Koziris and Prof. Yiannis Theodoridis for doing me the honor of being members to my PhD defence committee. Special thanks go to Dimitrios Skoutas for his useful advices and collaboration in research, especially in the last year of my PhD. I would also like to thank Alexandros Belesiotis a great researcher and friend, with whom we have collaborated during the final year of my PhD. I would also like thank Alexandros Efentakis for the collaboration in completing large parts of this thesis.

At this point I would like to thank Prof. Marios Dikaiakos from the University of Cyprus, who was the first to believe in me during my undergraduate studies and initiated me in research by employing me as a member of his lab. I offer to him my sincere gratitude, since he is also the one that suggested I come to Athens and NTUA to conduct my PhD research.

I feel lucky to have as lab mates Georgios Skoumas and Sophia Karagiorgou. The endless discussions we had throughout the years, as well as the pints of beer we had together were oases in times when this PhD seemed a never ending desert.

I would like to thank my fiancé and companion in life Irene Ioannidou. She was always there at difficult times, showing love and patience, since I have been away all these years. I thank her for her dedication and love. I dedicate this thesis to her, as well as to my parents Andreas and Eleftheria. My father Andreas is a true inspiration and a representative sample that hard work pays off. I thank my mother Eleftheria for her love and caring and for raising me with proper values.

Finally I would like to thank my brother Hariton, my sister Theophano and all my friends for always being there for me when I needed them.

Athens, May 2016 C. E.

To my family…

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General

The technological advances during the past years have resulted in the change of how people use and interact with the internet. Novel Web technologies and resulting applications have lead to a participatory data ecosystem that when utilized properly will lead to more rewarding services. People have been using cell phones for social interaction, willingly producing vast amounts of information. This, in addition to the technological ability of cell phones to track locations have lead to the consideration of advanced location-based services that take into account not only the location but all of the user-generated data that is produced implicitly or explicitly. In this PhD thesis, we investigate the case of this kind of Location-based Services and specifically of how to improve the typical location-based queries with user-generated content taken from sources in the web in the form of texts.

Research in the area of dataspaces [FHM05] regarding the integration of diverse data sources has mainly been focused on general subjects such as entity recognition and implicit handling of the uncertainty around vague objects. The main focus of this PhD project is to investigate as to how these concepts can be applied to geospatial and spatio-textual data where high volumes are coupled with highly particular data. Specifically, we will focus on how spatial data can be coupled with non-spatial characteristics of data that are a result of integrating various user-contributed data sources.

Location-based Services have been at the forefront of mobile computing as they provide an answer to the simple question as to what is around me. A lot of effort has been dedicated to improving such services typically by improving the selectivity of each request. Rating sites augment POIs with quality criteria. Preferences, when available, add further, user specific parameters to a request. Context limits the available information based on situational choices. However, what has not been captured yet are user experiences per se, i.e., assessing what people want in terms of what people in the same situation have done in the past.

User experiences can be extracted from various sources. With the proliferation of the Internet as the primary medium for data publishing and information exchange, we have seen an explosion in the amount of online content available on the Web. Thus, in addition to professionally-produced material being offered free on the Internet, everybody is able to make its content available online to everyone.

An example of such User-Generated Geospatial Content (UGC) is travel blogs, which are web sites where people share their experiences from trips they make around the world. A large amount of useful information can be extracted from the description of a journey, such as spatial content referring to geographic locations people have visited during their journey and a lot of semantically rich information which are closely related to that spatial content.

Another source rich of crowd-sourced data are location-based social networks (LBSNs) and micro-blogging platforms such as Twitter. Micro-blogging platforms, have become a very popular communication tool, where millions of users share opinions on different aspects of everyday life. Twitter reports over 100 million active users and 500 million tweets exchanged every day. This huge amount of data and the fact that they are offered publicly in real time make their management and analysis challenging. This information, along with the spatial characteristics of each entity in the social network can be used in order to construct interesting and useful queries.

This PhD thesis proposes access methods and queries that use the user-generated content found on the web as well as data taken from micro-blogs. New querying algorithms are proposed in order to support location-based services. The efficiency of the querying algorithms is achieved by the combination of state of the art shortest path algorithms on graphs, set similarity searching algorithms and spatial querying algorithms.

1.2 Contributions

The general framework for the contributions of this thesis lies in proposing enhanced location-based services and algorithms that combine spatial information with other types of user-generated content. In addition, we propose a database framework that can support distance queries on large-scale graphs and provides off-the-shelf tools to be used by applications. Below, we briefly each one of the contributions of the thesis per section.

1.2.1 Relevant Nearest Neighbor Queries

The first contribution deals with proposing a new type of enhanced location-based query namely k-Relevant Nearest Neighbor query (kRNN). We propose indexing schemes and algorithms for searching in a combined environment for close by points of interest (POIs), also taking into account the relevance between POIs.

In order to find relevant POIs, we leverage the use of travel blogs. Travel blogs contain rich user-generated geo-spatial content that allows us to extract meaningful information. Specifically, we introduce the concept of a Link-Of-Interest (LOI) between two POIs to express the respective relevance, i.e., find related nearest POIs to my location. Relevance is inferred by extracting pairs of POIs that are frequently mentioned together in the same context by users. In our work, we extract this information (co-occurrence of POI pairs) by parsing travel blogs and using the page structure to derive relevance. This relevance information can be represented by means of a graph in which POIs represent nodes and LOIs are links. At the same time we need to capture the spatial properties of POIs. The challenge in this work is as to how we efficiently combine relevance captured by a graph and the spatial properties captured by means of a spatial index.

We introduce the k-RNN query as follows. Given a query point that is represented as a POI, find the k relevant nearest neighbors by taking into account both spatial proximity and relevance. Essentially, the query takes into account how relevant a close by POI is to the query point, and combining this information with the spatial proximity between the two POIs, computes a combined RNN score denoting the Spatial-Relevance Distance between two points. We provide indexing schemes as well as variations of the searching algorithms in order to establish the efficiency during the search.

Extending this first work, we deal with optimizing the current approach. A major drawback in the efficiency of our algorithms is the search on the Relevance Graph. A step to the optimization of the proposed algorithms, is to facilitate the current research in graph theory, in order to use current state-of-the art approaches on computing shortest paths and tune them to the context of our work. During the search on the Relevance Graph, we currently compute the score of a node compared to the query POI by searching naively on the graph. We propose new algorithms that leverage the use of Landmarks on the graph, by conducting a pre-computation step. Using landmarks, in combination with the A* search, we achieve high efficiency compared to our previously proposed methods.

Our work on proposing the kRNN query, indexing methods and algorithms are presented in [EP13b]. The work on optimizing the kRNN querying algorithms leveraging the use of shortest path algorithms is presented in [EEP].

1.2.2 Similarity Search on Spatio-Textual Point Sets

Spatio-Textual Search has attracted attention in the past years due to the variety of practical applications. Several location-based services available on the web offer simple spatio-textual queries in order to return to a user close by POIs also considering textual descriptions during query time. In the second contribution, we address the problem of similarity search for spatio-textual entities, where each entity is represented by a set of spatio-textual objects. In particular, we introduce the *Spatio-Textual Point-Set Similarity Join* (`STPSJoin`) query. Given sets of spatio-textual objects, each one belonging to a specific entity, this query seeks pairs of entities that contain similar spatio-textual objects.

The spatio-textual point set similarity join can naturally model, for example, the search for users, which exhibit similar behaviour according to the spatio-textual objects they generate. For instance, given the locations and nature of store chains as spatio-textual objects, we can formalize the problem of identifying similar franchises and support site analysis for a franchise expansion.

To the best of our knowledge, current work treats such documents independently of the entity responsible for their generation. Different documents are treated as independent objects, even though they may have been created by the same user. In this paper, we consider each user to be associated with a set of documents, each one of which has spatial and textual properties, and focus on queries that require the identification of the users whose documents better satisfy some textual and spatial properties or pairs of users who are associated with similar documents.

To efficiently process spatio-textual point-set similarity joins, we adapt and extend the state-of-the-art algorithms for processing similarity joins for single points. The proposed algorithms make use of spatio-textual indexes in conjunction with an early termination and a filter-and-refinement strategy to effectively prune the search space, reducing the execution time by orders of magnitude. In addition, we present an alternative version of the best performing algorithm, which relies on an R-tree instead of a grid for the spatial indexing. The experimental evaluation is conducted using three large, real-world datasets. The results of the experimental evaluation demonstrate that the proposed algorithms achieve an order of magnitude and above improvement in terms of execution time when compared to the baseline method. The results of this work are presented in [EBSP].

1.2.3 Hub Labels on the Database for Large-Scale Graphs

Answering distance queries on graphs is one of the most well-studied problems on algorithmic theory, mainly due to its wide range of applications. Although a lot of recent research focused exclusively on transportation networks, the emergence of social networks has generated massive un-weighted graphs of interconnected entities. On such networks, the distance between two vertices is an indication of their closeness, i.e., for finding users closely related to each other or extracting information about existing communities within the social media users. Although we may always use a breadth first search (BFS) to calculate the distance between any two vertices on such graphs, that approach cannot facilitate fast-enough queries on main memory or be easily adapted to secondary storage solutions.

Considering the drawbacks of previous methods, our third contribution presents a database framework that may service multiple distance queries on massive large-scale graphs. Our pure-SQL *COLD* framework (COmpressed Labels on the Database) may answer multiple distance queries (point-to-point, kNN, RkNN and *one-to-many* queries), rendering it a complete database solution for a variety of practical applications on massive, large-scale graphs. Furthermore, it may be used in order to support for more advanced location-based services such as spatio-textual queries or geo-social queries. Our extensive experimentation will show that COLD outperforms previous solutions, including specialized graph databases, on all aspects (including query performance and memory requirements) while servicing a larger variety of distance queries, making it the best overall solution for servicing scalable, real-time applications operating on large-scale graphs. In addition, COLD is implemented entirely on a popular, open-source database engine with no third-party extensions and thus our results are easily reproducible by anyone. The results of this work are presented in [EEP15].

1.2.4 Visualizing and Exploring the Twittersphere

Micro-blogging platforms and social networks are a rich source of spatio-temporal information, which, together with additional information that can be mined from the social network's structure, makes them extremely valuable for monitoring users' opinions, sentiments and behaviour, and, consequently, making more timely and effective decisions. In this fourth contribution, we present *TwitterViz*, a complete solution for the visualization and analysis of spatio-temporal Twitter data, in combination with the analysis of the Twitter graph, by leveraging the use of a popular graph database and using state of the art visualization tools that aim at providing insights to the non-expert user.

The main motivation for the development of the *TwitterViz* framework stems from the fact that, although there exists a variety of applications that use data from twitter in order to provide useful analytics, no such a tool exists to provide a wide range of functionalities in a unified environment. *TwitterViz* manages to leverage both the spatio-temporal characteristics as well as the social aspects of the twitter data and is successful in assisting the non-expert in exploring the Twittersphere. *TwitterViz* can be extended to support more complex spatio-temporal and geo-social queries on the twitter multi-modal graph in order to allow for a more rich analysis. The results of this work are presented in [EASV15].

1.3 Thesis Outline

This PhD thesis is stuctured as follows:

Chapter 2 presents a literature review of the most recent advances in location-based services, as well as a review of shortest path techniques and algorithms on graphs.

Chapter 3 presents our initial work towards proposing a new type of query, namely the k-Relevant Nearest Neighbors query, that leverages the use of user-generated data from travel blogs in order to suggest places to visitors in specific areas. The second part of this work deals with optimizing the proposed querying algorithms for the k-RNN query and proposes additional algorithms that leverage the use of state of the art graph searching methods.

Chapter 4 proposes access methods and new types of queries that leverage the use of data from micro-blogs (such as Twitter). It deals with the area of Spatio-Textual Similarity Joins and proposes a new type of join query that deals with finding pairs of point sets.

Chapter 5 presents the *COLD* framework that allows for scalable and efficient distance queries on graphs, entirely within the database, allowing the user to use off-the-shelf tools to support distance queries on applications.

Chapter 6 proposes an application that could support all of the proposed location-based services in this thesis. It presents a user-friendly tool that visualizes and allows for the exploration and analysis of data from twitter. It is a framework for the non-expert user and allows exploring and analyzing huge streams of micro-blogging data using simple, unobtrusive yet powerful tools and can essentially support our proposed queries.

Chapter 7 concludes and provides directions for future research.

Chapter 2

Related Work

In the following we discuss the related to our work research in order to lay the foundation for the contributions of this thesis. Works that relate to this thesis are of two main categories: (a) Research that aims in combining spatial data with other forms of data such as textual descriptions of points in space or social connectivity, (b) research related to the computation of the shortest path on large-scale graphs. These two research areas are used in combination in the contributions of this thesis in order to propose advanced location-based services that combine different aspects of the characteristics of points in space. We discuss the recent advances and how they relate to each contribution.

We first describe spatial and spatio-textual search in the form of kNN, top-k and *join* queries. We then discuss the recent advances in geo-social query processing and user recommendation systems. In the last part, we discuss the recent advances in the computation of the shortest path on large scale graphs. When it is applicable, we discuss how this works relate to our contributions and how they differ.

2.1 Spatial and Spatio-Textual Search

Spatial data management has been in the forefront of database research for many years. The database community has given much attention in providing indexing methods so that a database can quickly and efficiently present spatially close areas or points. An index is a data structure that allows for the fast and efficient retrieval of information located in a data source [Clo05]. Different database applications require different kinds of indexes. As mentioned in [EN06], Spatial Databases provide concepts for databases that keep track of objects in a multidimensional space. Typical queries for a spatial database include:

Range Queries Find all the objects within a given area or region.

Nearest Neighbor Queries Find the objects that are closer to a given point or area.

Spatial Joins Determines for two sets of spatial objects all objects in a relationship described by a spatial predicate.

For these queries to be answered and to retrieve the required information efficiently, Spatial Index structures need to be built, in order to support this kinds of queries. A spatial access method organizes spatial objects in a particular space in a way so that, given a query, only specific subsets of all of the objects stored in the spatial database will be considered as a possible answer [G̈94]. Without a spatial index, all objects in a database need to be checked whether they meet a spatial selection criterion (for example whether they are located within a given distance from a given query point). Typical spatial index structures include the R-tree [Gut84a] and its variations (such as the R*-tree [BKSS90]).

2.1.1 Spatio-Textual Indexes

Nowadays, the Internet is comprised of a large amount of information in the form of text documents that are generated mostly by every day users and are therefore unstructured. Additionally, spatial characteristics of these web documents are available, such as the location where a specific document refers to. Also, a lot of spatial information is indirectly contained into a document, such as place names and postcodes. Spatio-Textual Search is a recent subject studied in the past years that aims in combining spatial search with textual descriptions. According to [VJJS05], studies in this area try to make web search geographically-aware, thus enabling it to index and retrieve documents according to their geographic context. Current research focuses on combining spatial and textual indexes and proposes hybrid methods to support geographical awareness.

In order to facilitate spatio-textual search, several indexes have been proposed in the recent literature. Current approaches are dealing with merging R-trees [Gut84a], regular grids, and space-filling curves with a textual index such as inverted files or signature files. The most common combination of indexes used are R-tree based structures with inverted files. On the one hand, R-trees, such as the R*-tree [BKSS90] provide a very efficient way of indexing and searching for spatial objects, and on the other hand inverted files an efficient way of indexing text documents. One of the first works in Spatio-Textual indexing was conducted in the context of the SPIRIT search engine [VJJS05], that takes as input a set of keywords and a set of spatial predicates. The engine returns documents that satisfy both sets (keywords and spatial predicates). SPIRIT facilitates the use of regular grids as spatial indexes and inverted files for the indexing of the documents. Following this idea, Zhou et al. [ZXW$^+$05] present hybrid indexing approaches comprising of R*-trees [BKSS90] for spatial indexing and inverted files for text indexing, suggesting the use of R*-trees instead of regular grids. Several approaches leveraging the combination of R*-trees and inverted files followed ([ZXW$^+$05, CJW09, RJGJN11]). Chen et al. [CSM06] used space-filling curves for spatial indexing, whereas in [DFHR08], R-trees are combined with signature files in the internal nodes of the tree, so that a combined index is created. The algorithm in [HS99] is used for incremental nearest neighbor queries. Cong et al. [CJW09] propose the IR-tree index, creating an inverted file for each node of an R-tree, again using the incremental k-NN algorithm from [HS99]. Here, linear interpolation is used [MSA05], in order for the spatial distance to be combined with the textual distance and therefore produce a combined score. In our work, we use a similar approach when combining the relevance score with spatial distance. A very related approach to [CJW09] is presented in [LLZ$^+$11]. Rocha-Junior et al. [RJGJN11] propose the hybrid index S2I, which uses aR-Trees [PKZT01]. This work takes into consideration the fact that the document frequency of terms in a corpus follows the Zipf's law, and therefore most of the terms occur infrequently while there exist some terms that occur frequently. Consequently, terms with different document frequency should be stored differently. The proposed hybrid index is called *S2I*. The authors claim to outperform all other proposed approaches in this problem domain.

Our work in Chapter 3 differs significantly in that we do not consider textual similarity, but use the co-occurrence of POIs in text as an indicator for relevance. This relevance information (and no actual textual information) is then combined with the spatial aspect of the POIs to retrieve relevant nearest neighbors. To the best of our knowledge, the most related work that combines user experiences and spatial data is [CCJ10], which introduces the notion of prestige to denote the textual relevance between nearby to the query point objects. The prestige from a given query point is propagated through the graph, an information that is later used to extend the IR-tree [CJW09]. The prestige is calculated by a personalized PageRank algorithm [JW03]. Compared to our work in Chapter 3, our index does not include a factor of randomness in the relevance score calculation as this is the case with PageRank-based algorithms. Additionally, Cao et al. assume that similar objects often co-locate, so they create sub-graphs based on this restriction. In the case that a query is so specific in the extend that it limits the query results (based on the text terms given), then this makes sense. However, in our case, the problem is to aid a traveller in visiting important places throughout a city or country. In this case, spatial limitations cannot be strictly provided, a fact that renders the creation of relevance sub-graphs based primarily on the spatial distance inadequate. Additionally, the algorithms they propose do not support updates on the insertion of new points. In general, the notion of relevance in [CCJ10] is based on different measures that are query dependent (textual relevance), a fact that is not being considered in our work. Our query considers only the location of a query point. Relevance is only derived from the collected dataset.

In the present work, relevance is computed by counting the co-occurrences of POIs in the same paragraphs of texts. In spite of the simplicity of this metric, recent results show that POIs co-occur in documents when they are spatially close, have similar properties, or interact with each other [LWK$^+$14]. On a similar note, Skoumas et al. [SPK13] show that textual narratives can be used to derive spatial relationships between POI pairs. In their analysis, they go even further and use this textual information as a means to perform qualitative positioning, i.e., discovering the location of POIs only using spatial relationships mentioned in texts.

2.1.2 Spatial group keyword queries.

Another type of spatio-textual queries is spatial group keyword queries such as [ZCM$^+$09, ZOT10, CCJO11, LWWF13]. These aim at finding groups of spatio-textual objects that collectively satisfy a number of given keywords, while minimizing the collective distances between points in the group and the given query point. [ZCM$^+$09, ZOT10] introduce the bR*-tree structure in order to solve the *mCK* query. [CCJO11] introduces two spatial group keyword queries. Both retrieve a group of objects that cover a given set of keywords; the first requires that the total distance between the objects in the group and the query point is minimized, while the second seeks to minimize the sum of the maximum distance between a point in the group and the query point and the maximum distance between any objects in the group. [LWWF13] revisits the spatial group keyword query with the maximum sum cost and also proposes a variation based on minimizing the maximal distance between any two objects of the group or any object of the group and the query point.

Although the aforementioned queries involve searching for groups of objects, they differ from the problem addressed in our work in Chapter 4. Our proposed STPSJoin query is not constrained by an input query point nor a given textual description. In addition, groups of objects are predefined according to the user they are associated with. STPSJoin considers spatial and textual distances of objects across groups, rather than within groups. Finally, STPSJoin deals with the problem of spatio-textual join, which is fundamentally different from range and kNN queries.

2.1.3 Nearest Neighbor and Top-k Queries

In the context of efficient kNN query processing, [NZTK08] propose the V*-Diagram to compute moving kNN queries, i.e., assessing the kNNs of a moving query point). This approach provides incremental evaluation of the kNN set, but differs significantly from our problem, since we do not take moving objects into consideration. [KOTZ04] investigate approximate kNN queries for data streams and propose the DISC technique to answer the e-approximate kNN problem (ekNN) using space-filling curves. e refers to an error by which the result set is bounded. This work differs from ours, since we do not take into neither data streams or approximate results. [SQZZ15] propose the *k-nearest neighbor temporal aggregate query* (k-NNTA), which computes the nearest POIs to a query point based on spatial distance and the number of check-ins for query point in a given time interval. A weighted sum is used compute a combined ranking score, which is similar to our work, but used in a different problem setting.

The problem of combining relevance and spatial distance in Chapter 3 is closely related to the problem of computing top-k queries that are based on different subsystems such as studied by Fagin et al. [Fag99, FLN03]. The difference to our approach is that we have a specific focus on spatial data and on how to combine the result sets. Also, in our final k-RNN list we have the exact scores, compared to the approximate scores of the NRA algorithm.

2.1.4 Spatial, Spatio-Textual and Similarity Joins

This section overviews works on set similarity joins, spatial joins, and spatio-textual joins. These works form the basis for our proposed algorithms on spatio-textual point set joins.

2.1.4.1 Similarity Joins

Similarity joins seek to identify pairs of objects from given sets that satisfy a predefined similarity threshold. The main application for textual set similarity join is duplicate detection across sets of documents. The set similarity join task is computationally challenging; a naive approach requires the consideration of the similarity between every possible pair of objects across sets. Set similarity joins have been extensively studied, especially with respect to textual characteristics, and multiple optimizations have been proposed. [SK04] uses an inverted index based probing method to reduce the number of potential candidates. [CGK06] observes that the prefixes of potential candidates must satisfy a minimal overlap. The `ALL-PAIRS` algorithm proposed by [BMS07] further optimizes the size of the inverted index, by visiting the tokens associated with every object with respect to their precomputed frequencies as well as reducing the size of the indexing prefix by ordering the objects with respect to their size. `PPJOIN+` [XWL$^+$11] is the state-of-the-art algorithm for set similarity joins. It builds on `ALL-PAIRS` and introduces a *positional filtering principle* which exploits the ordering of tokens, and operates both on the prefix and the suffix of the tokens of objects. `PPJOIN+` is internally used as the final step of our algorithms in order to efficiently compute textual similarity joins.

2.1.4.2 Spatial Joins

Data structures and algorithms for spatial joins have been widely studied in the literature. A relevant survey can be found in [JS07]. Spatial joins have been used in combination both with space partitioning as well as with data partitioning structures. The state of the art algorithm for spatial joins has been proposed in [BKS93]. The algorithm traverses the R-Tree starting from the root node and checks for pairs of child nodes with intersecting ϵ-extended minimum bounding rectangles (MBRs). An ϵ-extended rectangle is obtained by extending an MBR in every dimension by a given spatial threshold ϵ. Intersecting nodes are discovered following the plane sweeping method. We utilize this algorithm to prune the search space when searching for spatially relevant users using an R-Tree in Chapter 4.

The problem of spatial joins over point sets has not received much attention. Adelfio et al. [ANS11b, ANS11a] focus on similarity search for a collection of spatial point set objects based on the Hausdorff distance. The motivation behind their work is highly relevant to our proposed `STPSJoin` query. However, there are important differences. We consider web objects with spatio-textual characteristics and measure the distance among point sets using a different similarity measure. The Hausdorff distance measures the maximum discrepancy between two point sets, whereas in our work we use a measure inspired by the Jaccard coefficient which focuses on the amount of objects from different point sets that are similar.

2.1.4.3 Spatio-Textual Joins

Spatio-textual joins have attracted some attention recently with a specific focus on joins for spatio-textual points. This process is primarily executed for the purpose of duplicate detection. The work in [BCR11] is one of the first examples of spatio-textual join methods. They propose the SpSJoin query that follows the MapReduce paradigm for scalable computation of spatio-textual join queries. The spatio-textual join query has been also studied in the form of spatial regions associated with textual descriptions [LLF12, FLZ$^+$12]. Pruning strategies, based on spatial and textual signatures of objects, are employed to filter the number of candidates. [RLS14] presents grid and quad tree based indexes in order to efficiently partition the database either in a local or global fashion. They also explore different dimensions of the problem, including the use of `PPJOIN+` and `All-Pairs` for text similarity joins, as well as single and multi-threaded approaches.

Bouros et al. [BGM12] propose the state of the art spatio-textual join algorithms. Their work builds on top of `PPJ`, a baseline method that extends `PPJOIN+` to account for objects with spatio-textual characteristics and a given spatial distance threshold. The algorithms `PPJ-C` and `PPJ-R` extend `PPJ` by leveraging a grid and an R-Tree based index respectively. Their `PPJ-C` algorithm utilises a dynamic grid partitioning in order to focus the search for matching objects. A dynamic grid is constructed at runtime, with cell size the exact size of the spatial threshold parameter of the join query. The cells in the grid are traversed in a sequential order, starting from the left column in the lower row. When the right-most cell in a row is traversed, the search continues with the left-most cell of the higher row. The objects within each cell are examined against the objects of the same cell, as well as every adjacent cell that has already been traversed. This process is achieved using `PPJ`. This strategy ensures that objects with distant location are never evaluated, and at the same time no duplicate evaluations are performed. `PPJ-R` is built on top of an R-Tree index. This algorithm traverses the nodes of the R-Tree structure utilising the spatial join algorithm presented in [BKS93] in order to prune the search space. Finally, when `PPJ-R` reaches the leaf nodes of the tree,`PPJ` is used in order to identify pairs of objects that exceed a given spatio-textual similarity score.

Work on spatio-textual joins is highly relevant to our approach. However, the focus is different. To the best of our knowledge, current research in the field has focused on spatio-textual similarity joins among points. On the contrary, our work introduces spatio-textual similarity joins among point sets. Point sets are relevant when objects are grouped with respect to a common characteristic. In this case, the focus is on identifying similarities among groups, rather than single elements. For instance, in the case of web objects, a group consists of objects associated with the same user. In this case, point-set joins identify user similarity instead of object similarity.

2.2 Geo-Social Query Processing

Our work is also related to the area of Geo-Social query processing. Geo-Social networks are graphs that consist of users (nodes) and friendships with other users (edges). The nodes of the graph have spatial properties (location). A first work towards Geo-Social query processing [APP13] introduces a general framework to query a geo-social network. The social and geographical aspect are considered separately and as a result the answers to their proposed queries are not ranked based on spatial and social dimensions simultaneously. On a similar note, [AAP15] propose ranking functions to assign scores to geo-social nodes given a query location.

In combining spatial and social proximity, [MLTM15] compute top-k users based on a minimized weight function combining Euclidean distance and distance in a social graph. Somewhat similar to this work, spatial and graph distance are combined in a scoring function. However, the actual graph is different in that we use a weighted graph. Related to weights is also how the data is updated. While new POIs (nodes) can be added as well to the graph, the bulk of the updates will come from newly discovered relevance information, which requires frequently updating the weights of the graphs. Further, our scoring function combines edge weights with graph proximity, which makes it considerably harder to add such relative graph distance information to (the nodes of) a spatial index such as proposed in their work.

2.3 User Recommendation Systems

Matching users based on their location history is one of the main tasks of recommendation engines in location-based social networks [BZWM15]. User location histories have been used for identifying local experts, recommending friends, and extracting local communities. It has been revealed by several studies that location information plays a vital role in determining such relationships [CTH$^+$10, DKKLN11]. Typically, these approaches take into consideration additional information, such as location ratings, semantics of location descriptions and tags, sequence of visit or duration of stay. [LZX$^+$08] computes user similarity based on matching sequences of locations visited by the users, to find users with similar travelling patterns. Similar works

([LZX$^+$08]) deal with finding users with similar patterns in behavior. This is further extended in [ZZM$^+$11], to consider different spatial granularities and the popularity of visited locations. Semantic information about visited locations is additionally considered in [XZLX10, YLL$^+$10]. However, the latter convert GPS trajectories to semantic trajectories, thus finding similar users based on patterns of activities and behaviour, without necessarily requiring geographic proximity.

[GSWY13] study several features to identify users that are similar to a given user. Their methods are based on a logistic regression model. According to their work, a single and in many cases imprecise user location feature (such as city or country) is not effective. In our work in Chapter 4, we focus on multiple geo-tagged objects for each user, which may provide a better insight into location-based user similarity.

2.4 Shortest Path Computation

Shortest path computation is the process during which one aims in computing the distance between nodes in a graph. This thesis facilitates the use of two main concepts for the computation of the shortest path: (a) Landmarks and (b) hub labels. These two families of algorithms require the pre-processing of the data in order to be able to provide fast answers of distance queries on large-scale graphs.

2.4.1 Landmarks

In our work, we use *landmarks* as a means to optimize k-RNN processing in Chapter 3. Landmarks in relation to shortest-path problems were introduced in [GH05]. In this work, a small set of vertices called landmarks is chosen and for each vertex of the graph, the authors precompute distances to and from every landmark. The resulting method is the seminal ALT algorithm.

Potamias et al. in [PBCG09] significantly augmented the aforementioned concept of landmarks in various ways. To the best of our knowledge, they were the first to claim that the upper bounds obtained by the landmarks preprocessing phase are also important. They expand the usage of landmarks to undirected unweighted graphs (similar to our work). As a result their preprocessing phase is faster and easier, as it suffices to calculate distances from landmarks to all other vertices. Although there were many important works utilizing the concept of landmarks in various types of graphs, such as [TACGBn$^+$11] and [GBSW10], our work is heavily influenced and expands the aforementioned research results of Goldberg and Potamias. We use undirected graphs (similar to [PBCG09]) and utilize lower bounds obtained by the preprocessing phase of landmarks, in order to exclude neighbors that cannot possibly belong to the relevant neighbors set. Moreover, we use a variant of the ALT algorithm of [GH05] to accelerate graph traversal from each spatial seed towards the query point. Results of of the experimental evaluation in Chapter 3 clearly demonstrate the correctness and efficiency of our choices.

The *preprocessing stage* involved in landmarks based estimation methods is divided in two phases, the landmarks selection process and the computation of distances from landmarks to all other graph vertices (for undirected graphs). As far as the landmark selection process is concerned, many alternative strategies have been suggested in [GH05], [GW05], [PBCG09]. As Delling et al. suggest in [DW07], no technique picks landmarks that universally yield the smallest search space for random queries (although some perform better than others). Therefore in our work we utilize the simplest strategy, which is the *farthest* landmark selection strategy introduced in [GH05]. According to this, we pick a start vertex at random and a vertex v_1 that is farthest away from it. We add v_1 to the set of landmarks. Proceed in iterations, at each iteration finding a vertex that is farthest away from the current set of landmarks and adding the vertex to the landmarks set. The intuition behind this procedure is that we select landmarks to the periphery of the graph.

2.4.2 Hub Labels

Our work in Chapter 5 builds upon the 2-hop labeling or Hub Labeling (HL) algorithm of [GPPR01, CHKZ02] in which, preprocessing stores at every vertex v a forward $L_f(v)$ and a backward label $L_b(v)$. The forward label $L_f(v)$ is a sequence of pairs $(u, dist(v, u))$, with $u \in V$. Likewise, the backward label $L_b(v)$ contains pairs $(w, dist(w, v))$. Vertices u and w are denoted as the *hubs of v*. The generated labels conform to *the cover property*, i.e., for any s and t, the set $L_f(s) \cap L_b(t)$ must contain at least one hub that is on the shortest $s - t$ path. For undirected graphs $L_b(v) = L_f(v)$. To find the network distance $dist(s, t)$ between two vertices s and t, a HL query must find the hub $v \in L_f(s) \cap L_b(t)$ that minimizes the sum $dist(s, v) + dist(v, t)$. By sorting the pairs in each label by hub, this takes linear time by employing a coordinated sweep over both labels. The HL technique has been successfully adapted for road networks in [ADGW11, ADGW12, DGW13, AIKK14]. In the case of large-scale graphs, the Pruned Landmark Labeling (PLL) algorithm of [AIY13] *produces a minimal labeling for a specified vertex ordering.* In this work vertices are ordered by degree, whereas the work of [DGPW14] improves the suggested vertex ordering and the storage of the hub labels for maximum compression. The HL method has also been used for one-to-many, many-to-many and kNN queries on road networks in [DGW11] and [DW13] respectively. [EP14b] proposed *ReHub*, a novel main-memory algorithm that extends the Hub Labeling approach to efficiently handle RkNN queries. The main advantage of the *ReHub* algorithm is the separation between its costlier offline phase, which runs only once for a specific set of objects and a very fast online phase which depends on the query vertex q. Still, even the costlier offline phase hardly needs more than $1s$, whereas the online phase requires usually less than $1ms$, making *ReHub* the only RkNN algorithm fast enough for real-time applications and big, large-scale graphs.

Regarding secondary-storage solutions, Jiang et al. [JFWX14] propose their HopDB algorithm that suggest an efficient HL index construction when the given graphs and the corresponding index are too big to fit into main memory. There are also approximate solutions like [AMV15], optimized for secondary storage but since they are inexact and only focus on vertex-to-vertex queries their practical applicability is limited. The work of [ADF$^+$12] introduced the HLDB system, which answers distance and kNN queries in road networks entirely within a database by storing the hub labels in database tables and translating the corresponding HL queries to SQL commands. Throughout this work, we will compare our proposed COLD framework to HLDB, since to the best of our knowledge, it is the only framework that may answer exact distance queries entirely within a database. Moreover, within the COLD framework we also adapt the *ReHub* main-memory algorithm into a database context, so that its online phase may be translated to fast and optimized SQL queries.

2.4.3 Distance Queries on Graphs

Throughout the work in Chapter 5 we use undirected, unweighted graphs $G(V, E)$ (where V represents the vertices and E the arcs). A k-Nearest Neighbor (kNN) query in graphs seeks the k-nearest neighbors to an input vertex q. The RkNN query (also referred as the monochromatic RkNN query), given a query point q and a set of objects P, retrieves all the objects that have q as one of their k-nearest neighbors according to a given distance function $dist()$. In graph networks, $dist(s, t)$ corresponds to the minimum network distance between the two objects. Formally RkNN$(q) = \{p \in P : dist(p, q) \leq dist(p, p_k)\}$ where p_k is the k-Nearest Neighbor (kNN) of p. Throughout this work, we assume that objects are located on vertices and we always refer to *snapshot* kNN and RkNN queries on graphs, i.e, objects are not moving. Also, similarly to previous works, the term *object density* D

refers to the ratio $|P|/|V|$, where P is a set of objects in the graph and $|V|$ is the total number of vertices. Although, there is extensive literature focusing on kNN and RkNN queries in Euclidean space, since our work focuses on graphs we will only describe related work focusing on graphs.

Regarding road networks and kNN queries, the G-tree [ZLTZ13] is a balanced tree structure, which is constructed by recursively partitioning the road network into sub-networks. Then the best-first algorithm is applied on this G-tree index structure to answer kNN queries. Unfortunately, this method cannot scale for continental road networks, since it requires several hours for its preprocessing. Moreover, it requires a *target selection phase* to index which tree-nodes contain objects (requiring few seconds) and thus, it cannot be used for moving objects. Recently, the work of [DW13] expanded the graph-separators CRP algorithm of [DGPW11] to handle kNN queries on road networks. Unfortunately, (i) CRP also requires a target selection phase and thus, cannot be applied to moving objects and (ii) it may only perform well for objects near the query location. Hence, this solution is also not optimal. The latest work focusing on kNN queries on road network is the SALT framework presented in [EPV14] which may be used to answer multiple distance queries on road networks, including *vertex-to-vertex* (v2v), single source (one-to-all, range, one-to-many) and kNN queries. This work expands the graph-separators GRASP algorithms of [EP14a] and the ALT-SIMD adaptation [EP13a] of the ALT algorithm [GH05] and offers very fast preprocessing time and excellent query times. Regarding kNN queries, SALT does not require a target selection phase and hence it may be used for either static or moving objects.

For RkNN queries on road networks, the work of [SIT09] uses Network Voronoi cells (i.e., the set of vertices and arcs that are closer to the generator object) to answer RkNN queries. This work has only been tested on a relatively small network ($110K$ arcs) and all pre-computed information is stored in a database. Despite the fact that the preprocessing stage for computing the Network Voronoi cells is quite costly, the queries' executions times range from $1.5s$ for $D = 0.05$ and $k = 1$, up to $32s$ for $k = 20$, rendering this solution impractical for real-time scenarios. Up until recently, the only work dealing with other graph classes (besides road networks) is [YPMT06], although it has only been tested on sparse networks, e.g., road networks, grid networks (max degree 10), p2p graphs (avg degree 4) and a very small, sparse co-authorship graph ($4K$ nodes). In this work, the conducted experiments for values of $k > 1$ refer only to road networks, therefore the scalability of this work for denser graphs and larger values of k is questionable. Recently, Borutta et al. [BNNK14] extended this work for time-dependent road networks, but presented results were not very encouraging. The larger road network tested had $50k$ nodes (queries require more than $1s$ for $k = 1$) and for a network of $10k$ nodes and $k = 8$, RkNN queries take more than $0.3s$ (without even adding the I/O cost). In a nutshell, all existing contributions and methods have not been tested on dense, large-scale graphs, cannot scale for increasing k values and their performance highly depends on the object density D.

2.5 Location-based services based on Twitter data

Many research studies have been conducted in order to determine whether Twitter can actually give insights as to how people behave and provide location-based services based on these observations. Such studies have focused on analyzing a variety of spatio-temporal phenomena (e.g. [KP12, SGAF13]), as well as topics, sentiments and social interactions (e.g. [QECC12, QCC12]). Many research studies have been conducted in order to determine whether twitter can actually give insights as to how people behave. Recent studies, such as [KP12] study urban dynamics using user-generated data from twitter and foursquare[1]. They show how patterns in structure and function of urban areas can be extracted from the crowdsourced data (tweets) of citizen activities using a probabilistic topic model. The patterns, however, have been extracted using offline data that have been gathered in advance. They also argue that a reasonable step beyond this work could be a real-time analysis in order to infer the current state of a city.

Several other studies have been conducted mostly on the twitter graph (relationship graph between “twitter friends”) such as [QECC11] where they categorize twitter users in types based on the language they use. Then they argue whether the use of language is linked to social influence. They also use sentiment analysis in order to track “good” or “bad” influence. In [QECC12], the authors use sentiment analysis on tweets and suggest that it is possible to effectively track the emotional health of local communities from their residents’ tweets in an unobtrusive way, by using data from twitter. In [QCC12] again sentiment analysis is used in order to compare the social dynamics in Twitter to those in physical communities. [EAPD] seek to identify the users’ key locations based on their social networks profiles.

Typically, these works focus on specific problems and examine specific parts of the Twittersphere. A recent survey of approaches for Twitter analytics can be found in [GSZS14]. This survey identifies the need of a complete

[1]https://foursquare.com

framework to support the collection, analysis, visualization and querying on Twitter data. Based on this, we proposed the *TwitterViz* framework in Chapter 6 that aims in supporting all of the identified services.

Chapter 3

Efficient Processing of Relevant Nearest-Neighbor Queries

3.1 Introduction

Location-based Services have been at the forefront of mobile computing as they provide an answer to the simple question as to what is around me. A lot of effort has been dedicated to improving such services typically by improving the selectivity of each request. Rating sites augment POIs with quality criteria. Preferences add further user specific parameters to a request. Context limits the available information based on situational choices. However, what has not been captured yet are user experiences per se, i.e., assessing what people want in terms of what people in the same situation have done in the past. Our objective is to provide location-based services, specifically relevant k-RNN search based on the crowdsourced choices and experiences other users have had in the past. The basic premise of a so called k Relevant Nearest Neighbor (k-RNN) query is to combine *spatial proximity* with *cognitive proximity*, or relevance, as observed in previous user behavior. We introduce the concept of a Link-of-interest (LOI) between two POIs to express respective *relevance*, i.e., "find related nearest POIs to my location." In this first approach, relevance is inferred by observing pairs of POIs that are frequently mentioned in the same context. We discover this co-occurrence of POI pairs in travel blogs based on textual proximity, e.g., same paragraph. The more frequent POIs co-occur, the stronger and the more relevant is the LOI existing between them. Capturing POIs and LOIs in combination with k-RNN queries can be used to re-discover travel patterns, i.e., links between objects that frequently appear in other people's itineraries. Since not all objects are of equal type, e.g., restaurants, museums, bus stops, shops, hotels, etc., and this is in contrast to classical LBS, k-RNN queries will allow us to discover semantic chains of objects and, thus, provide the basis for an actual trip planning software. Eventually such semantic chains will be linked to user profiles that can then be used go generate customized travel guides.

While existing work addresses spatio-textual search, i.e., introducing a spatial aspect to (Web) search, thus, enabling it to index and retrieve documents according to their geographic context, to the best of our knowledge, the problem of combining user experience (expressed as relevance) with spatial proximity in the form of k-RNN search has only been addressed in [EP13b]. Here, relevance information is represented by means of a graph in which POIs represent nodes and LOIs are links. The spatial aspect of the data (POI locations) is captured using a spatial grid. The overall challenge in this work is on how to combine these two data structures (graph and grid) to efficiently process k-RNN queries. Initially, two basic methods are presented. *GR-Sync* (GR = Grid/Graph) expands the two indexes separately, but synchronizes their search at certain steps. *GR-Link* uses a tighter integration in that spatial search results are seeded to the graph search so as to minimize costly (and most often unnecessary) expansions. Focusing on the more efficient *GR-Link* method, our additional contribution will be on optimizing the search in the Relevance Graph by using landmark-based estimation. This approach belongs to the family of approximate shortest-path methods, since the distances between pairs of vertices can be estimated based on pre-computed distances to a fixed set of landmark nodes. We use those estimates to benefit our graph traversal by excluding nodes that based on the approximate distances are not worth exploring (filter step). The resulting method, *GR-Link-LM*, also utilizes bi-directed BFS to improve performance. Using the information we gain on lower bound distance approximations, we introduce *GR-Link-LM**, which additionally uses A*-search to further prune the search space. Experimental evaluation shows that the use of landmark-based estimation significantly improves the performance of k-RNN query processing.

The outline of the remainder of this work is as follows. The basic concepts and data structures for processing k-RNN queries are introduced in Section 3.2. Sections 3.3 and 3.4 then introduce the various k-RNN query processing methods and respective optimizations. The results of the experimental evaluation are presented in Section 5.3.

3.2 Data and k-RNN Queries

With the proliferation of the Internet as the primary medium for data publishing and information exchange, we have seen an explosion in the amount of online content available on the Web. In addition to professionally-produced material being offered free on the Internet, the public has also been encouraged to make its content available online to everyone as User-Generated Content (UGC). We, in the following, describe the data we want to utilize and how to exploit it in the context of k-RNN queries.

3.2.1 Data

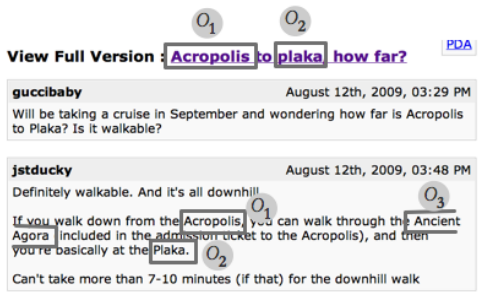
Web-based services and tools can provide means for users through attentional (e.g., geo-wikis, geocoding photos) or un-attentional efforts (e.g., routes from their daily commutes) to create vast amounts of data concerning the real world that contain significant amounts of information ("crowdsourcing"). The simplest possible means to generate content is by means of *text* when (micro) blogging. Any type of text content may contain geospatial data such as the mentioning of POIs, but also data characterizing the relationship between two POIs, e.g., eating at Eleni's tavern after visting the Acropolis in Athens, Greece.

In this work, we try to discover collections of POIs in texts and to use k-RNN queries as a means to navigate this forest of interesting locations to retrieve not only nearby, but also relevant objects. Consider the example of Figure 3.1a. In the specific text snippet of a travel blog, three distinct spatial objects are mentioned, $O_1 = \{\text{Acropolis}\}$, $O_2 = \{\text{Plaka}\}$, and $O_3 = \{\text{Ancient Agora}\}$. We introduce the concept of *Link-of-Interest* (LOI) as a means to express relevance between two POIs. Assuming that $O = \{O_1, \ldots, O_n\}$ is the set of all discovered POIs, then a text paragraph $P_x \subset O$ contains a set of POIs. It also holds that $O = \bigcup P$. We state that there exists a LOI $L_{i,j}$ between two POIs O_i and O_j, if both POIs are mentioned in the same text paragraph P_x. The set of all LOIs is defined as follows.

$$L = \{L(O_i, O_j) | \exists P_x \in P : O_i \in P_x \wedge O_j \in P_x\} \tag{3.1}$$

It is apparent that the definition of relevance is a very simple one, i.e., co-occurrence in a text paragraph. In future work, we intend to exploit the entire document structure as well as use sentiment information. However, any of these considerations will only affect the way we compute relevance and not the presented techniques for computing the specific type of query.

In the example of Figure 3.1, when considering O_1 as the query POI, we can see that it has a stronger relevance to O_2 than to O_3 and O_4, since O_1 and O_2 co-occur more often in the same paragraph of the given text snippet. Although O_4 is spatially closer to O_1 , the higher relevance of O_2 makes it the immediate relevant neighbor of O_1, i.e., when considering both, the spatial



View Full Version : Acropolis to plaka how far?

PDA

guccibaby August 12th, 2009, 03:29 PM

Will be taking a cruise in September and wondering how far is Acropolis to Plaka? Is it walkable?

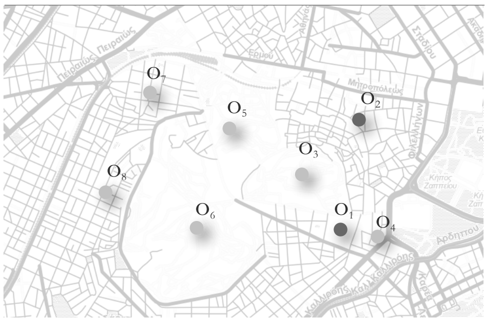
jstducky August 12th, 2009, 03:48 PM

Definitely walkable. And it's all downhill

If you walk down from the Acropolis, you can walk through the Ancient Agora (included in the admission ticket to the Acropolis), and then you're basically at the Plaka.

Can't take more than 7-10 minutes (if that) for the downhill walk

(a) POIs in text



(b) POIs on a map

Figure 3.1: "Closeby" Points Of Interest

distance and relevance. O_3 is spatially closer to O_1, but less relevant, hence overall more "distant".

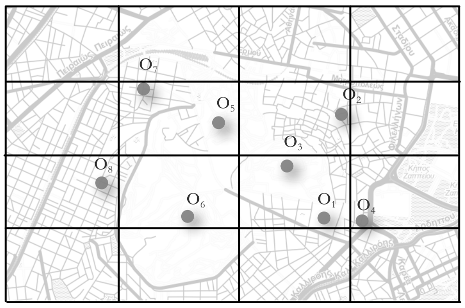
As we will see in the following section, k-RNN queries have the ambition to rediscover closeby POIs that have been visited by people in similar situations before. Here we combine the relevance information extracted from travel blog narratives with spatial proximity. Relevance can be considered an additional filter to navigate the forest of POIs in, for example, a tourist context.

3.2.2 k-RNN Queries

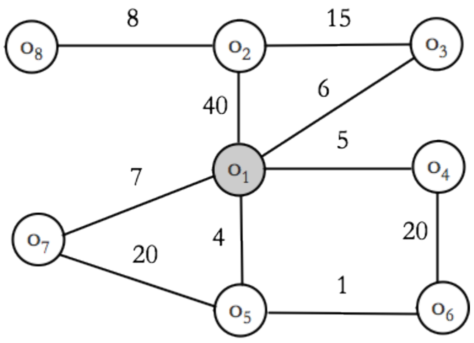
Combining relevance information with spatial distance will allow us to provide better query results, i.e., *POIs that are close-by and relevant.*

Let D be a spatial database that contains spatial objects O and Links of Interests L. Each spatial object is defined as $O_i = (O_i.id, O_i.loc)$ and each LOI as $L_k = (O_i.id, O_j.id, r)$, $O.id$ is a unique object identifier, $O.loc$ captures the object location in two-dimensional space, and r provides the relevance (score) of a LOI existing between two POIs O_i and O_j. The relevance r is computed as the total number of occurrences of a LOI, i.e., in the entire text corpus. In other words, r measures how often two POIs co-occur in paragraphs of the entire text corpus. The intuition is that the more frequently they are mentioned together, the more important is the LOI existing between them.

We introduce the k-RNN query as follows. Given a query point that is represented as a POI, find the k relevant nearest neighbors by taking into account, both, spatial proximity and relevance. An example is given in Figure 3.2 for a 1-RNN neighbor. The edge weights denote the relevance between the POIs, e.g., the weight of the edge linking O_2 and O_8 is 8. This means that the two objects co-occur eight times in the same paragraph when considering all documents of the available corpus. A combined k-RNN score assesses the Spatio-Relevance Distance between the two points, i.e., the smaller the distance (score), the better.



(a) Map



(b) Relevance graph

Figure 3.2: Relevance Graph and spatial "map" data

Equation 3.2 provides a means to compute a combined k-RNN score s_{rnn} that considers, both, spatial distance and relevance.

$$s_{rnn} = \frac{\alpha * s_r(Q, O)}{s_r.max} + \frac{(1-\alpha) * s_d(Q, O)}{s_d.max} \tag{3.2}$$

Q is the query point, O is the spatial object for which we compute the score with respect to Q, s_r is a relevance score and can be any kind of metric, $s_r.max$ is the maximum possible relevance score, s_d is the Euclidean distance between Q and O and $s_d.max$ is the maximum distance and depends on the query space. The parameter $0 \leq \alpha \leq 1$ is used to denote the importance of each distance function (Relevance or Spatial) and can be tuned according to the user's needs.

The following sections give detailed explanations as to the calculation of the Relevance Score and the computation of k-RNN queries.

3.2.3 Access Methods

Given a query and trying to define an efficient method for solving it in a data management context access methods are used to speed up processing. The following discussion mentions some methods that, either on their own, or, by combining them, efficiently solve the k-RNN query processing problem. When considering efficiency, we will also argue for the simplicity of a method, as the more complex a proposed access method is, the bigger is the challenge in implementing it in a given data management infrastructure.

Following this approach, we try to utilize spatial indexing methods with graph data structures. A regular *Spatial Grid* is used for indexing the locations of our POIs. In the *Spatial Grid* the geographical coverage of the earth is divided into a set of equally sized grid cells. Each cell corresponds to a disk block storing the POI of the respective grid cell. Each cell is identified by a unique id which is computed as a hash value of the cell's geographic coordinates. We maintain a hash table with key-value pairs for each grid cell, where the key is the grid cell's id and the mapped value is a pointer to the actual block on disk. In this way, we can locate the corresponding page on the disk in $O(1)$ time. This is also a reason for using a regular grid instead of other types of access methods.

We are using a grid instead of a more efficient spatial access method since it is easier to integrate it with a graph search. Using for example an R-tree, we would have to keep respective graph distance information with the nodes of the spatial index. As we will see later on, this presents a challenge as the *relative position* of nodes in the Relevance Graph (hops) affects the graph distance (cf. the recursive formula to compute the relevance score of Equation 3.3).In addition, with newly discovered LOI information, the weights of the graph frequently change, thus requiring frequent updates to the index structure.

As we will show, kNN queries can be processed incrementally by "radiating out" from the query point (see in the following), and it is a data structure that serves as a simple and elegant way of showing how a spatial index can be combined with others to index, e.g., space + relevance. Should a node reach is maximum capacity (fanout), an overflow node is added.

The *Relevance Graph* is defined as a graph $G(V, E)$, where V is the set of vertices that correspond to the POIs found in the set of documents, and E is the set of edges that correspond to links-of-interest (LOIs) between the POIs. The edge weights denote the relevance score r between a pair of vertices. Updating the Relevance Graph involves introducing new vertices (new POIs), edges and edge weights (both, based on new relevance). The Relevance Graph is stored by means of an adjacency list. Edges can be added in $O(1)$ time and the updates in $O(|E|)$ time.

3.3 *k*-RNN Query Processing

The major contribution of this work is how to combine the Spatial Grid and the Relevance Graph so as to efficiently support the processing of k-RNN queries. The following sections present various approaches to this integration starting with the simple *GR-Sync* method.

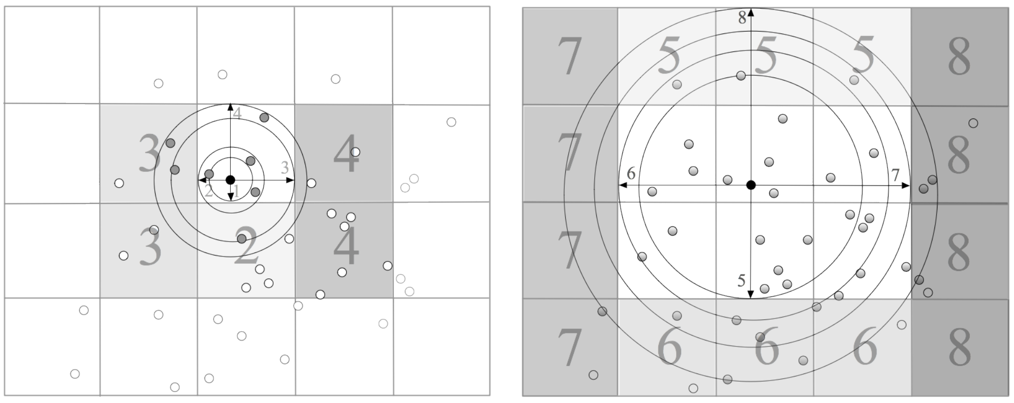
3.3.1 Index Synchronization

In a first, basic query processing method, termed *GR-Sync* (derived from Grid/Graph synchronization), we combine the results of the two separate indices to answer k-RNN queries. *GR-Sync* consists of two separate methods for identifying the k-RNN candidates in the Spatial Grid and in the Relevance Graph, respectively. The intuition behind this approach is an intermixed, stepwise execution of the search utilizing both indexes. After each step of the so-called *expansion process* in both data structures, the results (current list of respective kNN neighbor candidates) are combined. The search stops when the neighbors found are guaranteed to be the k-RNN.

3.3.1.1 Spatial Search

The *Spatial Expansion* algorithm uses a Spatial Grid as illustrated by Figure 3.2a. For each inserted POI we store four distances to the respective sides of the cell. Figure 3.3 shows an example of how the algorithm behaves for eight expansions beginning from a query POI. The algorithm first locates the grid cell of the query point.

Given that this algorithm tries to establish the relevance and proximity between POIs ("Where to go next?"), the query point is recruited from the set of POIs. If necessary, any user location can be mapped to the closest POI location.



(a) Steps 1-4 (b) Steps 5-8

Figure 3.3: Spatial expansion

The objective of the spatial expansion is to discover the nearest-neighbor POIs of the query point. We do so, by retrieving close-by POIs in neighboring cells in a step-wise fashion. As we will see, this results in a *snail-like expansion.* The order in which the cells are retrieved depends on the location of the query point within its containing cell. Consider the example of Figure 3.3a. The query point is closest to the bottom side of the cell (arrow labeled "1"). To guarantee that all POIs have been examined that are within distance 1, only the cell of the query point needs to be loaded (indicated also by the circle around the query point of radius "1"). However, for the case of distance "2" also the points of the bottom cell (labeled 2) need to be retrieved. This snail-like expansion next retrieves two cells labeled "3" and then two cells labeled "4". While we also retrieve points that are further away than d_4, we are also certain that we have not missed any candidates.

The points that are not within the maximum distance are added to a *retrievedPoints* list, whereas the points that are within the maximum distance are added to the kNN list based on Euclidean distance. With increasing distance, the search expands to neighboring cells as shown in the example of Figure 3.3b. As expected, the larger the distance from the query point, the more points are retrieved in each step, e.g., 4 cells in Step 8.

3.3.1.2 Computing the Relevance Score

To retrieve the *relevance score* s_r, we have to examine the Relevance Graph. Our method orients itself on the Breadth-First graph traversal. It starts with the query point and in a first step examines all adjacent nodes in the Relevance Graph. Subsequent steps examine all neighbors of the initially-visited nodes and so on. To compute a Relevance Score, we use the following recursive formula that computes the score of a node k based on the score of its predecessor, or, parent node p. For the initial expansion $p = q$.

$$s_r(k) = 1 - \frac{w_{p,k}}{\sum_{i=1}^{N} w_{p,i}} + s_r(p) + s_h(k) \tag{3.3}$$

$$s_h(k) = s_h(p) + h(k) \tag{3.4}$$

where s_r is the *Relevance Score* for node k, $w_{p,k}$ is the weight from the parent node p to k, N is the number of the parent's one-hop neighbors, the number of nodes that are expanded during the same step, $s_r(p)$ is the parent's relevance score, and $s_h(k)$ is a score derived from $h(k)$, which is the number of hops needed to reach k from q. $s_h(k)$ is based on the respective score of the parent node and increases with the distance of k from d. This also ensures that any node l with $h(l) > h(k)$ will have a higher relevance score than node k, i.e., $s_r(l) > s_r(k) : h(l) > h(k)$. The recursive relevance score computation emphasizes that even the "worst" one-hop neighbor is preferable to the "best" two-hop neighbor. In terms of semantics, one-hop neighbors are POIs that co-occur at least once in a paragraph. Two-hop neighbors are POIs that co-occur with another POI only in a transitive, "friend-of-a-friend" fashion. As such, the more distant two POIs are in the Relevance Graph, the less obvious is a relevance relationship existing between them. As such, both, the Relevance Score and also the Spatial Score are a penalizing score as they reflect distance, i.e., the higher s_r, the less relevant a node k with respect to q.

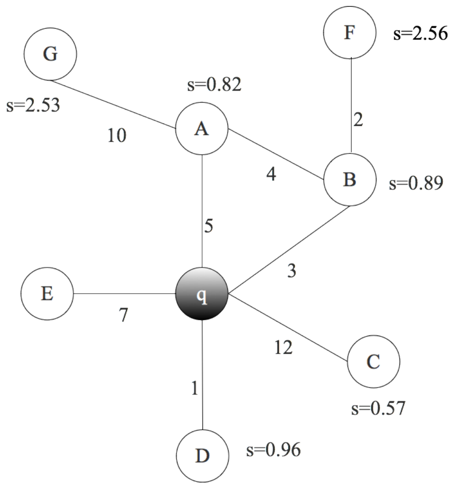


Figure 3.4: Graph expansion and relevance score

What follows are some relevance score computation examples. *Example 1* - Figure 3.4 depicts an example of the calculation of the Relevance Score for the neighbors of a query point. First, the sum of the weights of the edges that connect the query point to its neighbors is calculated. In this example $\Sigma(0) = 28$. To compute each one-hop neighbor's Relevance Score s_r, we need to get the intermediate score of each neighbor node based on the edge weight, i.e., $1 - (w_{p,k}/\sum_{i=1}^{N} w_{p,i})$. For example, node B has an intermediate score of $1 - 3/28 = 0.89$. Applying the rest of Equation 3.3, with $s_r(p) = 0$, since $p = q$ and $h = 0$ with the hop count starting at 0, $s_r(B) = 0.89$. Continuing the expansion, i.e., retrieving the two-hop neighbors, for example for node F, $s_r(F) = (1 - 2/6) + 0.89 + 1 = 2.56$ (the edge that connects the neighbor to its parent is not taken into consideration). It should be noted that for computing the score of a node, we consider the shortest path as far as the number of hops from the query point is considered. Additionally, should a node be reachable by the same number of hops through multiple nodes, we consider the lowest scoring node as a parent node.

Example 2 - (Added Table 3.1). Using Figure 3.2 as a running example, we seek the 1-RNN neighbor of query point O_1. To compute the score of O_2 with respect to O_1, first, a spatial search is initiated on the Spatial Grid and the spatial distance between O_1 and O_2 is found to be $s_d(O_1, O_2) = 350m$. Expanding O_1 on the Relevance Graph, we use Equation 3.3 to calculate the relevance score of O_2, which is 0.32. O_2 is 1 hop away from O_1, thus $s_h(O_2) = 0$, and its parent's relevance score is $s_h(O_1) = 0$ (by default). With $\alpha = 0.5$ (equal preference for relevance and spatial distance), $s_r.max = 2$, and $s_d.max = 600m$, the combined score between O_1 and O_2 is $s_{rnn} = 0.37$ (calculated using Equation 3.2). Computing the s_{rnn} scores for all other objects with respect to O_2 as shown in Table 3.1, we can see that indeed O_2 is the 1-RNN of O_1.

O	$s_h(k)$	$s_r(k)$	$s_d(Q,O)$	s_{rnn}
O_2	0	0.32	350	**0.37**
O_3	0	0.89	200	0.39
O_4	0	0.96	100	0.40
O_5	0	0.93	480	1.03
O_6	1.96	1.96	475	1.28
O_7	0	0.88	550	1.13
O_8	1.32	1.32	600	1.33

Table 3.1: Calculation of relevance score based on Figure 3.2 example

3.3.1.3 Synchronizing Expansion

To compute the k-RNN score, the *spatial and the relevance graph search need to be synchronized* and their scores combined. The following approach computes combined scores and evaluates the status of the search at *fixed intervals* determined by the number of expansions performed in each index. The Spatial Grid utilizes a snail-like expansion and retrieval of cells surrounding the query point, and the Relevance Graph uses a BFS-like expansion of increasing distance to the query point. After each expansion step, the two lists contain the closest in terms of spatial and relevance score neighbors, respectively. To synchronize the two expansions, we define a respective rate of expansion steps. The expansion ratio χ determines the ratio of spatial to relevance graph expansions. χ will typically be in the range of 4 to 16, as spatial expansions are considerably cheaper (read accessess). Each search maintains a list of expanded POIs and their respective score. After each expansion cycle (considering the expansion ratio), the two lists are checked and the common POIs, i.e., appearing in both lists, are identified. Using Equation 3.2, their combined score s_{rnn} is computed. All POIs with such a score are added to a queue sorted by s_{rnn} and essentially containing all top k-RNN neighbors that have been identified at this point, i.e., POIs that have been examined in both data structures.

To define a termination criterion, we have to guarantee that all k-RNN POIs have been found, i.e., further search will not reveal any POIs with a better score than the ones already identified. Each search keeps an *open list* for each set of respective POIs found so far to record the corresponding k-RNN score. If a POI is found in only one index, to compute its score, we use a *best-case estimate for the missing score*, i.e., that it will be found during the next expansion. For example, assuming a POI was retrieved in the spatial search, but not yet in the relevance search, the relevance score will be the lowest possible score after the next expansion (relevance score is dominated by number of hops = expansion steps). Similarly, should the spatial score be missing, we assume the best case, i.e., that the POI will be discovered during the next round of expansion with a distance from the query point just beyond the current search distance. As the searches progress, the scores of the POIs, and thus the open lists, will be updated based on the current number of hops and search distance. The *GR-Sync* algorithm is shown in Algorithm 1. $SpatialExpansion(q, k, (n \times \chi), SG, sg)$ takes as input the query point q, the number of sought neighbors k, the current expansion radius $(n \times \chi)$, the spatial grid SG and the sg list that contains the discovered points so far by the spatial expansion algorithm (cf. Section 3.3.1.1). $RelevanceExpansion(q, k, n, RG, rg)$ takes as input the query point q, the number of sought neighbors k, the relevance graph step count n, the Relevance Graph RG and the rg list that contains the discovered points so far by the relevance expansion algorithm (cf. Section 3.3.1.2). *GR-Sync* uses s^d_{rnn} and s^r_{rnn} as the minimum predicted combined k-RNN scores for nodes with no valid spatial, or relevance score, respectively. If $s^*_{rnn} = min(s^d_{rnn}, s^r_{rnn})$, the minimum score that still could be found, is less than $max(s_{rnn})$, i.e., the maximum score of identified rnn candidates, it means that the POIs in the (top-k) result list are guaranteed to be the top k-RNN neighbors (Line 11). This condition can be used as the termination criterion.

3.3.2 Index Linking

The k-RNN query processing algorithm presented so far does not actually combine the two indices (spatial and relevance), but only evaluates the results at times with the expansion steps used as a means of synchronization.

ALGORITHM 1: GR-Sync k-RNN algorithm

```
Input: Query POI $q$, Number of Neighbors $k$, Relevance Graph $RG$, Spatial
       Grid $SG$, Discovered POIs in $RG$: $rg$, Discovered POIs in $SG$: $sg$,
       $k$-RNN result list $rnn$, Relevance Graph step count $n$, Expansion ratio
       for Spatial Grid $\chi$.
Output: Complete $k$-RNN result list $rnn$.
1  $complete$ = false;
2  while $\neg$ $complete$ do
3      $SpatialExpansion(q, k, (n \times \chi), SG, sg)$;
4      $RelevanceExpansion(q, k, n, RG, rg)$;
5      $rnn = sg \cap rg$ NN results with complete score;
6      $s^r_{rnn} = MinScore(rg)$ Min. predicted score, Relevance Graph;
7      $s^d_{rnn} = MinScore(sg)$ Min. predicted score, Spatial Grid;
8      $s'_{rnn} = Min(s^d_{rnn}, s^r_{rnn})$ Min. expected score;
9      $s^*_r = MaxScore(rnn)$ Max. score in current result list;
10     If the max $k^{th}$ RNN distance found so far is less than the min predicted
         distance, the algorithm completes;
11     $complete = (|rnn| \geq k \wedge s^*_r < s'_{rnn})$;
```

An inherent problem with this method is the search in the Relevance Graph, which after the first hops becomes very costly. Big expansions in the Relevance Graph (empirically observed for hops > 3) retrieve a lot of data and, hence, incur disk activity. We have devised a variation of our query processing technique that manages to *bound the Relevance Graph expansion.* This so-called *GR-Link* method, termed as such since it combines the two indexes (Grid/Graph linking), we use the results of the spatial search as seed elements for the search in the Relevance Graph. The intuition is that many 1-hop expansions (spatial search seeds) are "cheaper" than a single n-hop expansion of the query point. POIs retrieved by the spatial search (and not discovered yet in the relevance search) are expanded in the Relevance Graph (cf. Figure 3.5). The intuition is that the graph searches of (i) the seeded POIs and (ii) the query point will meet eventually and, thus, we can compute a POI's relevance score. Without *seeding the search*, we would have to wait until the query point expansion reaches a POI. The simplified example of Figure 3.5 should illustrate that the POIs expanded are much fewer than the POIs that would have been retrieved if a second expansion step from the query point would have been performed.

The *GR-Link* approach also allows us to better bound the estimates of

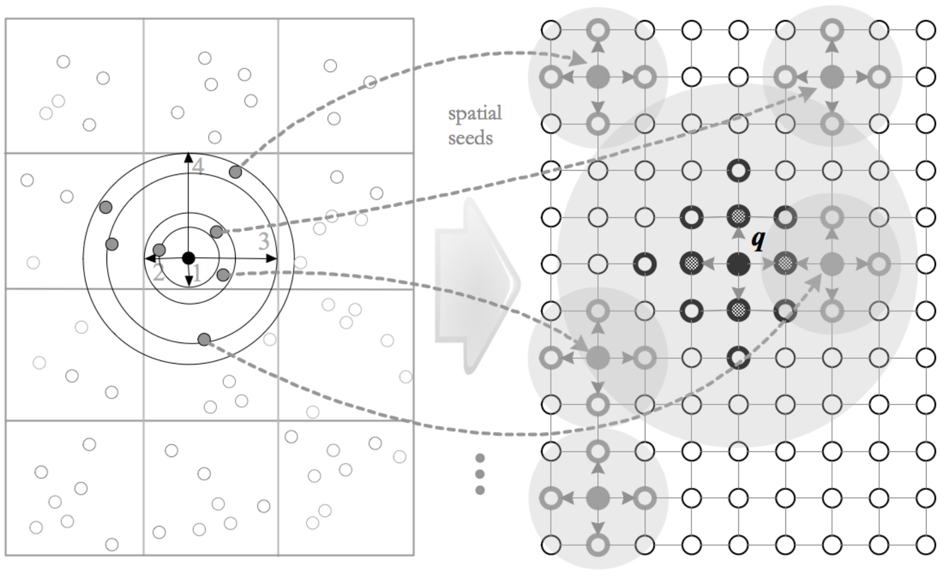


Figure 3.5: GR-Link method using Grid POIs in Graph search

missing scores. Placing a POI on the Relevance Graph, we know for sure that if their expansions (POI and query point) do not meet, the base for their score computation is at least the sum of the two expansions plus one hop (since they did not meet). Therefore, the predicted s_r score for such a point is larger than what it would have been in the non-hybrid case.

The pseudo code for this method is shown in Algorithm 2. The difference to the *GR-Sync* algorithm are the statements of Lines 4-8, which are seed POIs for the Relevance Search.

As the experimental section will show, the intuition of choosing many small Relevance Graph expansions over one large expansion pays off and the *GR-Link* method shows superior performance in terms of IO when compared to the *GR-Sync* solution.

3.4 *k*-RNN Query Optimization

Having introduced the *k*-RNN problem and several processing algorithms, we, in the following, introduce two novel methods that provide improved query processing by optimizing the search in the Relevance Graph. The methods involve *landmark-based distance estimation* and the *ALT* algorithm for additional pruning of the search space.

ALGORITHM 2: GR-Link k-RNN algorithm

```
Input: Query POI $q$, Number of Neighbors $k$, Relevance Graph $RG$, Spatial
       Grid $SG$ Discovered POIs in $RG$: $rg$, Discovered POIs in $RG$ - spatial
       seeds: $rg'$, Discovered POIs in $SG$: $sg$, $k$-RNN result list $rnn$, Relevance
       Graph step count $n$, Expansion ratio Spatial Grid $\chi$.
Output: Complete $k$-RNN result list $rnn$.
1  $complete$ = false;
2  while $\neg$ $complete$ do
3      $SpatialExpansion(q, k, (n \times \chi), SG, sg)$;
4      for $each\ poi \in sg$ do
5          $RelevanceExpansion(poi, 1, RG, rg')$;
6      $RelevanceExpansion(q, n, RG, rg)$;
7      $rg = Connect(rg, rg')$ Combine graph expansions;
8      $rnn = sg \cap rg$ NN results with complete score;
9      $s^r_{rnn} = MinScore(rg)$ Min. predicted score, Relevance Graph;
10     $s^d_{rnn} = MinScore(sg)$ Min. predicted score, Spatial Grid;
11     $s'_{rnn} = Min(s^d_{rnn}, s^r_{rnn})$ Min. expected score;
12     $s^*_r = MaxScore(rnn)$ Max. score in current result list;
13     If max $k^{th}$ RNN distance found so far is less than min predicted distance, the
         algorithm completes;
14     complete = $(|rnn| \geq k \wedge s^*_r < s'_{rnn})$;
```

3.4.1 Landmarks

Landmark-based distance estimation in graphs is a method that involves the pre-selection of a set of landmark nodes and computing the distances of all graph vertices from those landmarks. Given two arbitrary graph vertices, the distance between them can be estimated based on the respective distances from landmarks as follows. Given a set $S \subseteq V$ of landmarks and distances $d(L_i, v)$, $d(v, L_i)$ for all nodes $v \in V$ and landmarks $L_i \in S$, the following triangle inequalities hold: $d(u,v)+d(v,L_i) \geq d(u,L_i)$ and $d(L_i,u)+d(u,v) \geq d(L_i,v)$. Therefore, the function $max_{L_i} max\{d(u,L_i) - d(v,L_i), d(L_i,v) - d(L_i,u)\}$ provides a lower bound for the distance $d(u,v)$.

For the specific case of k-RNN queries, we will utilize landmarks for the Relevance Graph search to estimate the distance of seeded POIs to the query point and thus minimize the necessary expansions in the Relevance Graph. The necessary preprocessing stage is divided into two phases, the landmarks selection process and the computation of distances from landmarks to all other graph vertices (for undirected graphs).

As far as the landmark selection process is concerned, many alternative strategies have been suggested in [GH05], [GW05], [PBCG09]. As Delling et al. suggest in [DW07], no technique picks landmarks that universally yield the smallest search space for random queries (although some perform better than others). Therefore in our work we utilize the simplest strategy, which is the *farthest* landmark selection strategy introduced in [GH05]. In a nutshell the algorithm randomly selects a starting vertex and performs a Dijkstra search. Upon termination, the whole graph has been expanded, producing shortest-paths between the start and all other vertices of the graph. Choosing now the most distant vertex, it becomes the first landmark. Continue the search by finding at each step a new vertex that is furthest away from the current set of landmarks. Stop when n landmarks have been added to the set of landmarks L. The intuition behind this procedure is to choose landmarks that are at the periphery of the graph.

We also implemented a custom simplified alternative strategy, to have a clear view of the impact of the landmark selection process to our results. Our new method, referred hereafter as the *partitioning - highest degree* method initially partitions the Relevance Graph. For each partition, the node of the highest degree is added to the landmarks set. To partition the Relevance Graph, we used METIS [KK98] a free, well-known graph partitioning tool, used often in the context of shortest path computation [KMS06], [MSM10], [EPV11], [ETP12]. Thanks to METIS partitioning, our partitioning - highest degree method guarantees a uniform distribution of landmarks throughout the Relevance Graph. Picking the highest degree node for each partition is also a natural choice, since as Potamias et al. suggest in [PBCG09], "the more connected a node is, the higher the chance that it participates in many shortest paths". Following the selection of landmarks L, we compute the distance from the vertices of the graph to the landmarks. As such these landmark distances will provide us with a *relevance score* s_r *estimate* for each spatial seed.

To simplify the approach, we consider the *Relevance Graph* G to be unweighted and undirected. This is a simplification over the approach presented so far, since the focus is on the efficiency of the graph traversal and not on the relevance calculation *per se*. Another modification in relation to the graph dataset is that now the graph needs to be fully connected, since landmark-based approaches do not apply to unconnected graphs.

Following landmark selection, it suffices to pre-compute the graph (hop) distances from each landmark to all graph nodes by running a Breadth-First search from each landmark. This information is captured by a two-dimensional distance matrix D of size $N * M$, where N is the total number of vertices and M the number of landmarks used. Therefore, D_{ij} refers to the hop distance from landmark j to vertex i.

The upper and the lower bound distances to landmarks can now be used as estimates for the true distance between any two graph vertices. Given a set of landmarks L and two vertices u and v, whose distance we need to determine, for undirected, unweighted graphs the following equation holds [PBCG09]:

$$max_{L_i}|d(L_i, v) - d(L_i, u)| \leq d(u, v) \leq min_{L_i}|d(L_i, v) + d(L_i, u)| \tag{3.5}$$

Therefore, a good approximation for the lower bound of the graph distance $d(u, q)$ between any vertex u and the query point q in the Relevance Graph is as follows.

$$d(u, q) \geq max_{L_i}|d(L_i, u) - d(L_i, q)| \tag{3.6}$$

Having a way to approximate the distance between a vertex and a query point in the Relevance Graph, we can prune the graph search space based on this approximation. In Algorithm 3 the *GR-Link-LM* algorithm is shown, which is an expansion of the *GR-Link* algorithm for the computation of the k-RNN query.

The algorithm works just like the *GR-Link* approach, with the difference that for the expansion of the spatial seeds the routine *LandmarkExpansion* (cf. Line 5) is used. More specifically, after a number of χ expansions are made in the spatial index, a set rg' of POIs has been discovered. These POIs serve as spatial seeds in the Relevance Graph search (cf. *GR-Link* approach). At this point, instead of initiating a "blind" BFS traversal of the spatial seeds, we use the *LandmarkExpansion* algorithm in order to facilitate the use of landmark lower bounds and therefore improve the performance of the graph traversal. Algorithm 4 shows the algorithm for the expansion of the spatial seeds.

Let s be the spatial seed node that is to be expanded in the Relevance Graph G, q is the query POI, and L_i is a landmark node. For the spatial seed s we calculate the lower bound $max_{L_i}|d(L_i, s) - d(L_i, q)|$ using Equation 3.6. If the calculated lower bound for s is greater than the actual graph distances of the current k-RNN results, then the spatial seed s could be safely ignored, i.e., it is too *far-away* from the query point in the Relevance Graph to ever become a viable solution. With this simple but crucial optimization, we efficiently prune the spatial seeds that need to be expanded and, thus, significantly lower the number of I/O operations. This of course results in improved performance (see Section 5.3), while still guaranteeing the optimality of results.

ALGORITHM 3: *GR-Link-LM* k-RNN algorithm

```
Input: Query POI $q$, Number of Neighbors $k$, Relevance Graph $RG$, Spatial
       Grid $SG$ Discovered POIs in $RG$: $rg$, Discovered POIs in $RG$ - spatial
       seeds: $rg'$, Discovered POIs in $SG$: $sg$, $k$-RNN result list $rnn$, Relevance
       Graph step count $n$, Expansion ratio Spatial Grid $\chi$.
Output: Complete $k$-RNN result list $rnn$.
1  $complete$ = false;
2  while $\neg$ $complete$ do
3      $SpatialExpansion(q, k, (n \times \chi), SG, sg)$;
4      for each $poi \in sg$ do
5          $LandmarkExpansion(poi, q, RG, L, D, rg')$;
6      $RelevanceExpansion(q, n, RG, rg)$ $rg = Connect(rg, rg')$ Combine graph
        expansions;
7      $rnn = sg \cap rg$ NN results with complete score;
8      $s^r_{rnn} = MinScore(rg)$ Min. predicted score, Relevance Graph;
9      $s^d_{rnn} = MinScore(sg)$ Min. predicted score, Spatial Grid;
10     $s'_{rnn} = Min(s^d_{rnn}, s^r_{rnn})$ Min. expected score;
11     $s^*_r = MaxScore(rnn)$ Max. score in current result list;
12     If the max $k^{th}$ RNN distance found so far is less than min predicted distance,
        the algorithm completes;
13     complete = $(|rnn| \geq k \wedge s^*_r < s'_{rnn})$;
```

An example of an k-RNN query with landmarks is shown in Figure 3.6. In comparison to Algorithm 2 the spatial search portion is omitted. The example shows five spatial seeds. Before each of the seeds are expanded, its lower bound distance to the query point is calculated based on the landmark information. In this example, two spatial seeds are not expanded further since their landmark distance is above the threshold for them to become viable k-RNN candidates. The remaining three seeds are expanded. This example illustrates that by eliminating nodes the expanded graph portion becomes smaller. It is expected that the *GR-Link-LM* method thus shows improved performance.

3.4.2 ALT algorithm optimization

The presented k-RNN method combines *central* BFS traversal from the query point q with the individual BFS traversals originating from the seed points. In this section, we further optimize this process by using the lower bounds

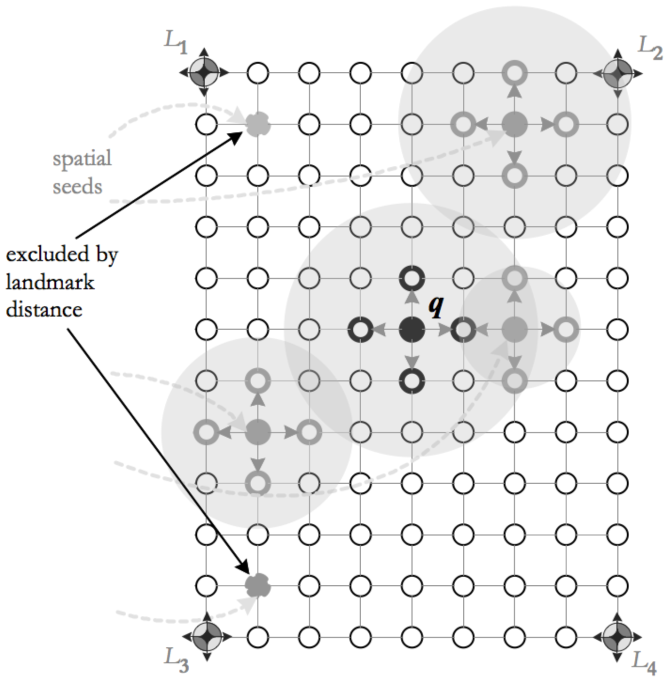


Figure 3.6: *GR-Link-LM*method using Landmark nodes on the Graph periphery

ALGORITHM 4: Landmark Based Expansion algorithm

```
Input: The spatial seed to be expanded $s$, The query POI $q$, Relevance Graph
       $RG$, Set of landmarks $L$, Distance matrix from landmarks to all nodes $D$,
Output: Discovered POIs in $RG$ - spatial seeds: $rg'$
1 Chosen landmark $L_i$ = -1;
2 Lower bound $d_{approx}$ = -1;
3 $L_i = selectLandmark(q, s, L, D)$;
4 $d_{approx} = computeLowerBound(L_i, q, s, D)$;
5 if $\neg worthExpanding(s, L, d_{approx})$ then
6     return;
7 while $\neg$ completedHopExpansion() do
8     $BFSexpansion(s.BFSqueue, s, q)$;
```

of the landmarks preprocessing phase to exclude spatial seeds that cannot belong to the k-RNN set. *GR-Link-LM* traverses the graph from the seed points by using plain BFS in an un-directed fashion. This is also illustrated by the gray circles in Figure 3.6. In what follows, we augment the seed expansions by using a unidirectional version of the ALT algorithm of [GH05]. The ALT algorithm is a well-known goal direction shortest-path technique that combines A* search [HNR68] with the lower bounds provided by the landmarks preprocessing phase. Combining the *central* BFS traversal from the query point q and the individual unidirectional ALT algorithms from the seed points requires several additional enhancements. For that purpose, we adapt several core-concepts from [GH05]:

- For each unidirectional ALT algorithm originating from a seed point, we must maintain the shortest path length μ seen so far between the seed and the query point. Initially, for each such search μ is set to infinity. When an edge (u, w) is scanned by the ALT algorithm and w has already been scanned in the BFS traversal from the query point, we know the shortest paths for $s - u$ and $w - q$ are of lengths $d_s(u)$ and $d_q(w)$, respectively. If $\mu \geq d_s(u) + l(u, w) + d_q(w)$, we have found a shorter path than those seen before, so we update μ accordingly.

- The unidirectional ALT algorithm cannot terminate as soon as it scans a node already scanned by the *central* BFS traversal from the query point. Instead, we can safely abort the search only provided the ALT algorithm is about to scan a vertex v with $k(v) = d_s(u) + max_{L_i}|d(L_i, u) - d(L_i, q)| \geq \mu$.

ALGORITHM 5: The unidirectional ALT algorithm optimization

```
Input: The spatial seed to be expanded $s$, The query POI $q$, Relevance Graph
       $RG$, Set of landmarks $L$ , Distance matrix from landmarks to all nodes
       $D$, $ALT$ expanded nodes $\sigma$.
Output: Discovered POIs in $RG$ - spatial seeds: $rg'$.
1 Chosen landmark $L_i$ = -1;
2 Lower bound $d_{approx}$ = -1;
3 $L_i = selectLandmark(q, s, L, D)$;
4 $d_{approx} = computeLowerBound(L_i, q, s, D)$;
5 if $\neg worthExpanding(s, L, d_{approx})$ then
6     return;
7 while $\sigma \neq 0$ do
8     $A * expansion(s.ALTqueue, s, q)$;
9     $\sigma = \sigma - 1$;
```

Another essential difference between the *GR-Link* method and the current proposal is the way we alternate between the two opposing searches (the central BFS traversal from the query point and searches originating from the seeds). For the *GR-Link* method, the alternation strategy was measured in hops, i.e., we performed one-hop BFS expansion from the query point and one-hop BFS expansion from each seed. Although we still use a one-hop BFS expansion from the query point, the ALT search from the seeds can no longer be measured in hops. Instead we keep track of the number of nodes extracted from the priority queue per seed search. Once the set number of nodes has been taken off the priority queue of the ALT search for a particular seed, we then proceed to the next seed. Once all seeds perform the necessary number of node extractions from their priority queues, we perform another one-hop BFS expansion from the query point and proceed in this iterative fashion. Algorithm 5 shows the algorithm in detail.

An example of a k-RNN query using the ALT optimization is shown in Figure 3.7. The decision whether or not the spatial seed, or the subsequently discovered nodes will be further expanded is taken based on the landmark lower bounds. Additionally, since the *ALT* algorithm uses the landmarks-based lower bounds, we observe that nodes closer to the query point q are expanded first, allowing the algorithm to find a common node with the query point BFS expansion sooner, and therefore calculate the spatial seed's relevance score way before all nodes will be expanded.

The intuition behind the *GR-Link-LM** approach is shown in Figure 3.8. The spatial seed s, using the *ALT* algorithm, expands towards the query POI

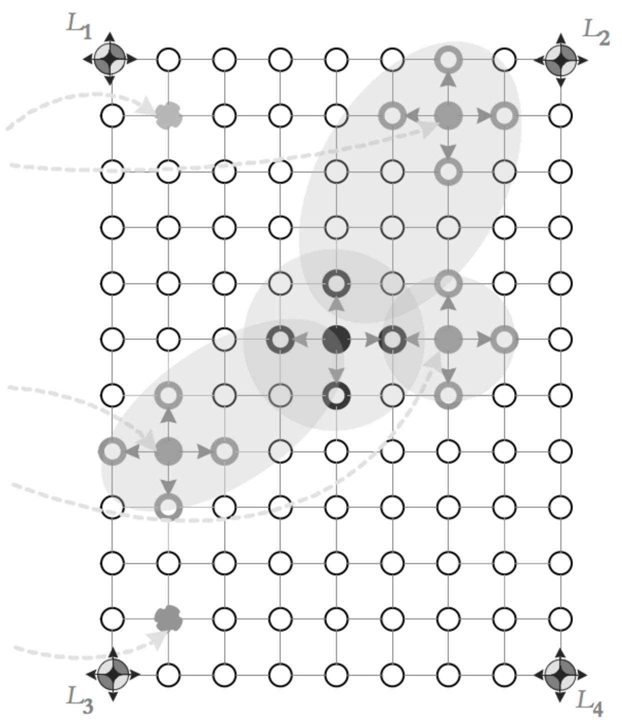


Figure 3.7: *GR-Link-LM** method: ALT algorithm significantly optimizes the relevance graph search

q. In this way, it will retrieve a smaller number of relevance graph nodes until it reaches q's central BFS expansion. The overall impact of this approach is visualized by the ellipses, rather than the circles of the *GR-Link-LM* case, in Figure 3.7. Using this directed search, a smaller portion of the Relevance Graph is expanded, overall resulting in improved performance.

The following section will examine the performance of the proposed algorithms with respect to existing approaches.

3.5 Experimental Evaluation

We assess the efficiency of the proposed k-RNN query processing methods a in performance study measuring accessed data (disk I/O operations) and CPU time. The experiments use real and synthetic datasets. Overall, we compare five methods, (i) the *GR-Sync* method (naive method), (ii) the *GR-Link* method, (iii) the *GR-Link-LM* method, (iv) the *GR-Link-LM** method and (v) an hypothetic ideal method (see below for an explanation). What we expect from the experiments is to see a well-performing *GR-Link* method

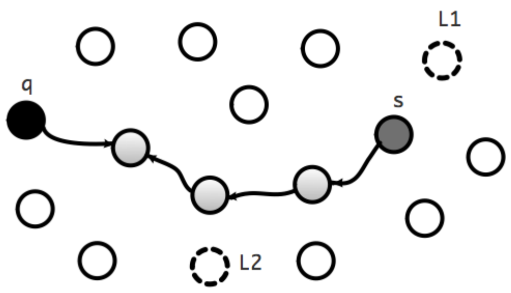


Figure 3.8: Intuition for A*/ALT search: Prune the graph space and go faster from s towards the query POI's q expansion

as well as a boost in performance from the *GR-Link-LM* and *GR-Link-LM** methods, that come close to the performance of the ideal method.

3.5.1 Data

Actual POI + LOI data was collected from a corpus of 120k documents from user-generated content found in travel blog sites[1]. The texts were preprocessed to collect the necessary information for our index, i.e., (i) identification of POIs, (ii) geocoding of POIs (spatial position) and (iii) location within the document (paragraph id and offset).

The procedure of generating the realistic dataset used in this work follows simple heuristic rules derived from the characteristics of the real-world data. Essentially, the data generation approach is based on hotspots that resemble cities, related POIs, and then LOIs between the POIs. We generate center points (cities, attractors) in an area of 30×30 degrees (approximately 3300 × 3300km) using a uniform distribution. For each center point, using a normal distribution, we generate neighboring points (POIs). We then generate relationships (LOIs) between POIs using again a normal distribution, i.e., the neighbors that are spatially closer to a POI in question are more likely to be linked to it than the ones further away (Tobler's spatial bias). On average, we generate 15 LOIs per POI (maximum = 50). Our synthetic dataset consists of approximately 810k POIs that generate a Relevance Graph with a total of 6 million edges (LOIs). The synthetic data is considerably larger than the Relevance Graph derived from travel blogs and thus should produce more conclusive results. Figure 3.9 visualizes the major nodes and a closeup view of the links between nodes.

[1] www.travelblog.com, www.traveljournal.com, www.travelpod.com

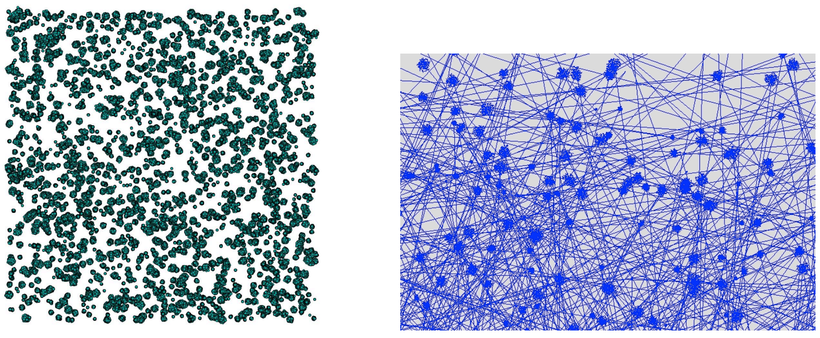

(a) Global view (b) Detailed View

Figure 3.9: Synthetic dataset

3.5.2 Experimental Setup

We evaluate the various query processing methods not only by means of different datasets, but also under a varying set of parameters including (i) the number of k and (ii) the preference parameter α emphasizing either spatial distance or relevance. The performance is assessed in terms of number of page accesses (I/O) performed to retrieve the data from disk for each query as well as CPU time. In each case, we performed 100 queries and computed the average I/O and CPU time shown in the respective charts. The 100 query points were randomly picked from the POIs in the dataset. An important aspect in the experiments is the spatial grid used to index the data. We use a regular grid with a spacing of 0.02 degrees (approximately $2km$) in longitude and latitude. The synthetic dataset consists of approximately 810k POIs covering an area of 30 degrees longitude and latitude, respectively, an extent somewhat comparable to Europe. The 810k POIs are grouped into 160k cells amounting to an average space utilization of 5, but not exceeding 30. Keep in mind that we only consider occupied cells in our index.

The real dataset contains 120k points that are scattered all over the globe. Here, 80k cells contain at least one POI. This amounts to an average space utilization of < 2, with few cells containing more than 5 POIs. Essentially, we used this dataset as a template for the synthetic data generation. Still, we wanted to present the results of the performance study to see whether the performance trends with respect to the indexing methods persist.

The expansion ratio χ was set to 12, i.e., for each expansion in the Relevance Graph, 12 expansions in the Spatial Grid are performed. Unless mentioned otherwise, the parameter α used in Equation 3.2 to compute the k-RNN score s_{rnn} is fixed to 0.5, i.e., considering the spatial and relevance score equally important.

All experiments were performed using a computer with a Intel Core i5 2400 CPU and 8GB DDR3 RAM, running Ubuntu Linux 11.10. The index and algorithms were implemented in Java.

3.5.3 Basic *k*-RNN Query Performance

In our experimental setup, we first vary the number of sought nearest neighbors k to see how scalable our algorithms are. α is set to 0.5, i.e., considering the spatial and relevance score equally important.

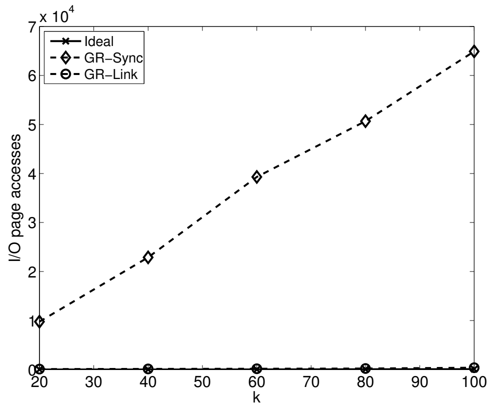
The first experiment should verify the performance advantage of the *GR-Link* approach over the naive *GR-Sync* algorithm. Figure 3.10a shows that *GR-Link* outperforms *GR-Sync* by an order of magnitude. This is due to the fact that *GR-Sync* needs to search large parts of, both, the Spatial Grid and the Relevance Graph in order to guarantee the result. Therefore, when POIs are found in one of the two indices, it needs to keep searching the other to find the combined k-RNN score. This causes the algorithm to retrieve too many (irrelevant) pages. On the other hand, the *GR-Link* method uses the results of the Spatial Search to limit the expansions in the Relevance Graph.

While a comparison to the naive *GR-Sync* approach does not pose a challenge, so does the next experiment relate the performance of *GR-Link* to an "ideal" approach (cf. Figure 3.10b). The ideal approach simulates a Relevance Graph search that terminates as soon as the k-RNN POIs are found, i.e., we have a-priori knowledge of the results and are only expanding the graph until the result is "discovered". As such, this method is unrealistic, but serves as a lower-bound for the performance of the hybrid index. Figure 3.10b shows that the *GR-Link* method examines more data than the ideal approach (albeit little data overall). Still, in terms of comparison to this baseline approach, the hybrid index's performance is close to that of the ideal method.

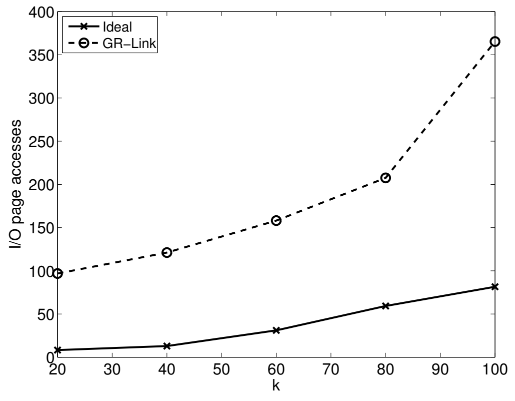
Using a real-world dataset (cf. Figure 3.11), the number of page accesses as well as CPU time when compared to the experiments with the synthetic dataset appear to be orders of magnitude greater due to a sparse dataset. Overall however, this experiment shows the same trends observed for synthetic data.

3.5.4 Tuning Landmark-based Optimizations

The naive *GR-Sync* approach performs orders of magnitude worse than *GR-Link*. We thus excluded it from the following experiments that establish an optimal setting for (i) the number of landmark nodes N, (ii) the choice of a landmark selection algorithm, (iii) the number of expanded nodes during the



(a) *GR-Sync* vs. *GR-Link* vs. Ideal



(b) *GR-Link* vs. Ideal

Figure 3.10: Naive, Hybrid and Ideal Index I/O performance

ALT traversal for each expanded spatial seed, and (iv) the number of RNNs k.

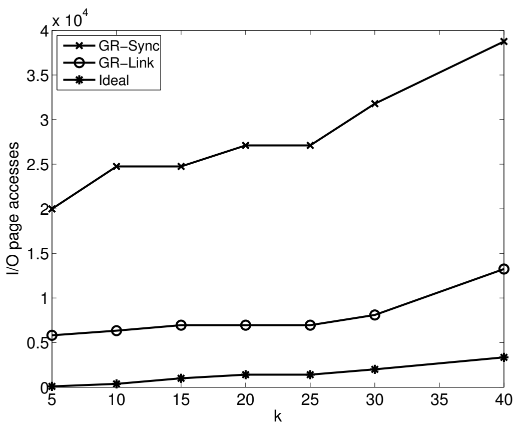
An important parameter is the number of *ALT* expansions σ per spatial seed for all numbers of sought neighbors k. As explained in Section 3.4.2, for every step of the main query point BFS expansion, for each of the spatial seeds the *ALT* algorithm expands a fixed number of σ nodes in the graph search. This is not the case for BFS since there we can ensure that each expansion step is in fact a one-hop expansion. We also perform experiments varying the number N of landmarks, as this affects the performance of the *GR-Link-LM* and *GR-Link-LM**algorithms. Both proposed landmark selection algorithms, i.e., using the *farthest* as well as the *partitioning - highest degree* landmarks, are assessed.

The disk space overhead for the *GR-Link-LM* method includes the distance table D from the landmark nodes to all other nodes in the graph. The overhead for each number of landmarks is shown in Table 3.2. The size of the database containing the data needed to build the indices (POI and LOI information) is 74.96 MB.

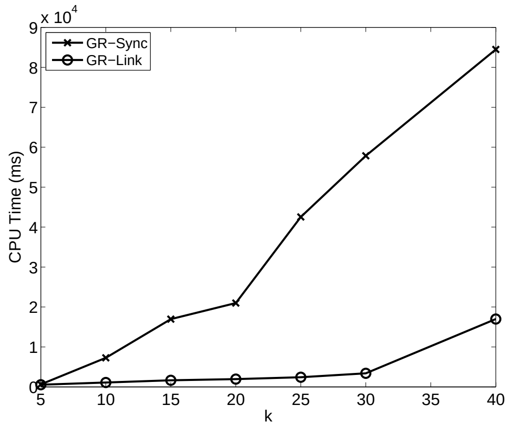
What follows are experiments relating to the various parameters affecting the k-RNN query performance.

3.5.4.1 Varying Landmarks

Seeking $k = 60$ results, Figure 3.12 shows that the *GR-Link-LM* and *GR-Link-LM** algorithms perform best with N=8. Choosing more landmarks provides a better lower bound estimate (to points to “triangulate” to). How-



(a) I/O performance



(b) CPU Performance

Figure 3.11: Real Dataset: varying k

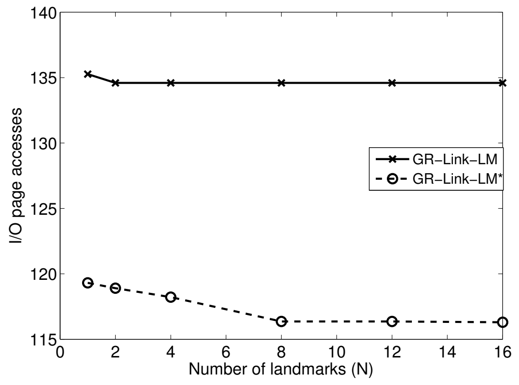
Num of Landmarks	Overhead (MB)
1	3.08
2	6.16
4	12.33
8	24.66
12	36.96
16	49.28

Table 3.2: Overhead for the number of landmarks

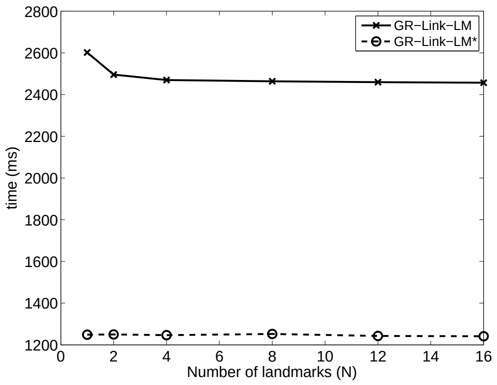
ever, increasing the number of landmarks to $N > 8$ does not provide tighter distance bounds and thus no further performance gain.

Using a different landmark selection algorithm, the *partitioning - highest degree* method, Figure 3.13 shows a performance similar to that of the *farthest* landmark selection method, i.e., reaching N=8 we have the best performance.

The *farthest* method precomputes the distances between the landmarks and all other nodes in the graph in one step. The *partitioning - highest degree* method requires extra pre-processing steps (i) to partition the graph (METIS) and then (ii) to select the high degree nodes from each of the partitions. Therefore, the pre-processing cost using the *farthest* landmark selection method is lowest. Since with both methods have comparable performance, we chose the *farthest* method for subsequent experimentation. Also, the number of landmark nodes will be fixed to $N = 8$.



(a) I/O Performance



(b) CPU Performance

Figure 3.12: Varying the number of Landmark nodes N, I/O and CPU Performance. Landmarks are selected using the *farthest* method (k=60)

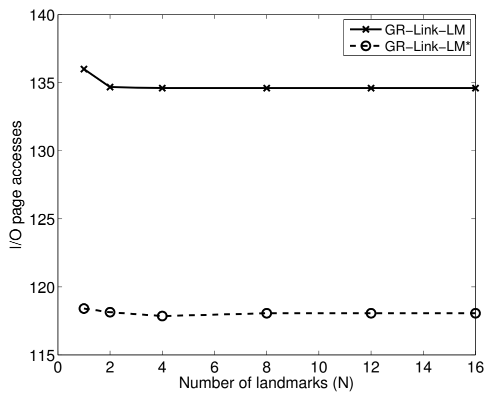
3.5.4.2 *ALT* Optimization

The *GR-Link-LM**algorithm uses an *ALT* approach for the expansion of the spatial seeds (A*combined with landmarks). Using stepwise expansion, we need to determine the number of the *ALT* expanded nodes σ in each *GR-Link-LM**step for each spatial seed. Figure 3.14 shows the performance for varying σ and k in terms of I/O disk page accesses. For smaller k, the optimal σ is lower, whereas for larger k, the optimal σ increases. The best choice seems to be $\sigma = 80$. For smaller ks, the main query point expansion reaches in a few steps the spatial seeds (not a lot of data to look for). In this case, performing extra node expansions for the spatial seeds incurs additional disk page accesses without any benefit (the solution was found already). However, for larger ks, in which cases the *GR-Link-LM** algorithm needs to search larger portions of the graph to guarantee the top k-RNN expanding more nodes at each *ALT* step results in fewer node accesses. The graph search is directed towards the query point expansion with a larger "momentum".

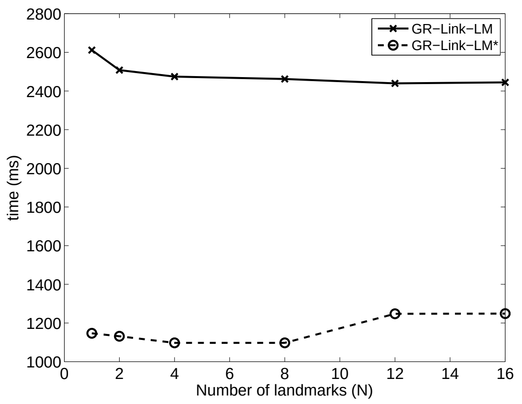
3.5.5 Query Performance for Varying k

Having determined the optimal σ values, the core experiments will be to assess how the k-RNN processing methods perform under varying k values.

Figure 3.15a presents a comparison of all of the methods discussed in this paper. Figure 3.15b focuses specifically on the respective performance of the *GR-Link-LM* and *GR-Link-LM**methods. As mentioned, the benchmark "ideal approach" simulates a Relevance Graph search that terminates as soon as the k-RNN POIs are found, i.e., we have a-priori knowledge of the results



(a) I/O Performance



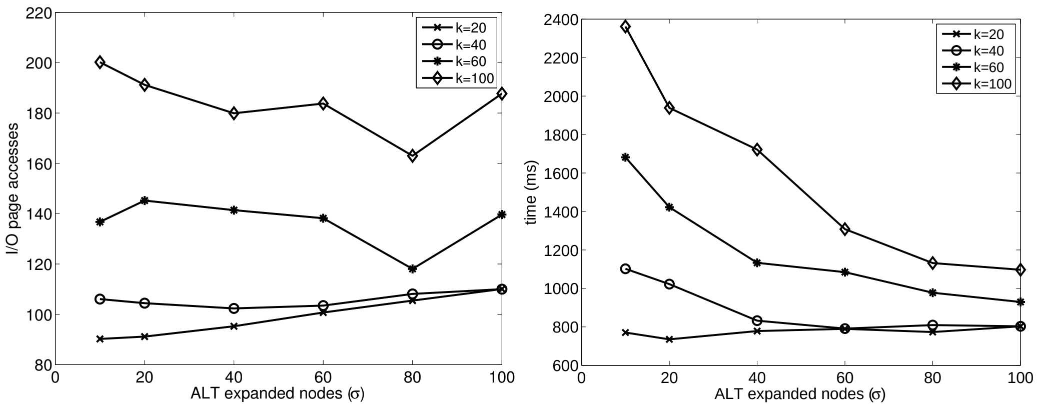
(b) CPU Performance

Figure 3.13: Varying the number of Landmark nodes N, I/O and CPU Performance. Landmarks are selected using the *partitioning - highest degree* method (k=60)

and are only expanding the graph until "discovered". As we can see, the *GR-Link-LM* and *GR-Link-LM** methods outperform the *GR-Link* method, both, in terms of I/O page accesses and CPU time. Also, while not quite reaching the performance of the ideal method, especially the *GR-Link-LM** method not only reduces the gap considerably, but also shows the same I/O and CPU cost k growth behavior as the ideal method.

Focusing on the respective performance of the two new methods, for large k, *GR-Link-LM** performs considerably better than the simple, Breadth-first approach of *GR-Link-LM*. The *GR-Link* algorithm - as expected - performs worse than our proposed *GR-Link-LM* and *GR-Link-LM** algorithms. This shows that the exclusion of expanded nodes based on our landmarks-based method benefits the proposed algorithms and shows a considerable performance gain over the simple, "blind" bidirectional BFS approach. In addition, the performance of *GR-Link-LM** is improved over the *GR-Link-LM* approach. The *ALT* algorithm used by *GR-Link-LM** for the spatial seed expansions proves to work as expected, directing the search towards the query point and therefore saving the algorithm from expanding unused graph nodes.

Figure 3.16 shows the running times of the algorithms. The results are in line with the I/O-based experiments and the respective performance advantage is even more evident.



(a) I/O Performance (b) CPU Performance

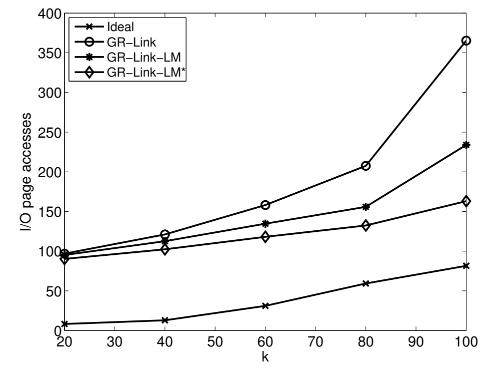
Figure 3.14: Varying the number of expanded ALT nodes σ, I/O and CPU Performance. Different k values show different optimal σ values.

3.5.6 Assessing the Effect of α

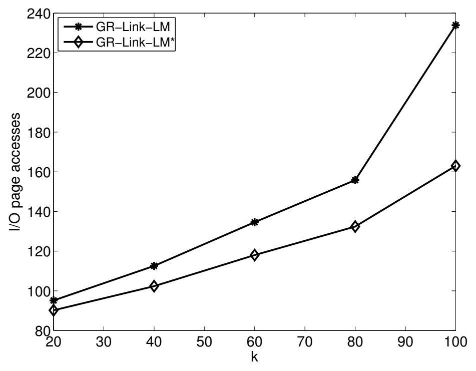
The preference parameter α balancing the effect of spatial distance vs. relevance on the query result is not only a critical factor for the quality of the result, but also affects the query processing cost. With $\alpha < 0.5$ it favors the Spatial Score and with $\alpha > 0.5$ it favors the Relevance Score. Figure 3.17 shows that with an emphasis on Relevance ($\alpha > 0.5$), the cost increases due to a more expensive Relevance Graph expansion. Also, algorithms with an optimized graph search increase their performance advantage with increasing α. The results are analogous when measuring the CPU time.

3.5.7 Spatial Grid vs. Graph Partitioning

All POI data is kept on disk. The assumption so far is that each populated Spatial Grid cell corresponds to a disk block. This approach supposedly has the disadvantage of only considering the spatial characteristics without taking into account that the nearest-neighbor search explores the Relevance Graph as well. It is expected that the further the points are spatially distributed, the more blocks need to be retrieved, thus, increasing the I/O cost. Our approach in extending the current disk layout is now to group POIs together based on Relevance Graph proximity. This is achieved by partitioning the Relevance Graph using the METIS graph partitioning tool [KK98]. However, the grid-based partitioning is essential for spatial search and the underlying expansion mechanism. Using graph-based partitioning, we mapped the spatial grid cells to graph partitions. Now, when the spatial search requests a specific grid cell, the respective one or more graph partitions are



(a) *GR-Link* vs. LM vs. LM* vs. Ideal



(b) *GR-Link-LM* vs. *GR-Link-LM**

Figure 3.15: *GR-Link*, *GR-Link-LM*, *GR-Link-LM**and Ideal Index I/O performance

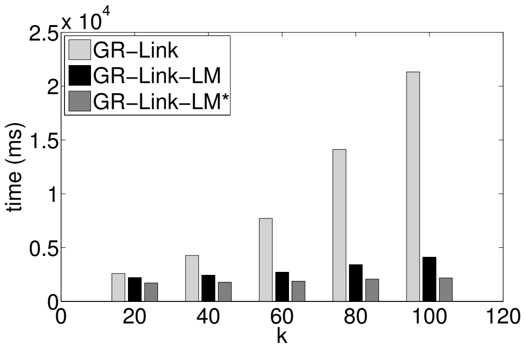


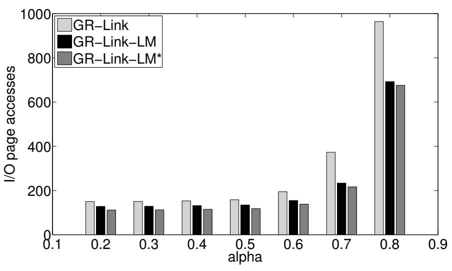
Figure 3.16: CPU performance, varying k

fetched.

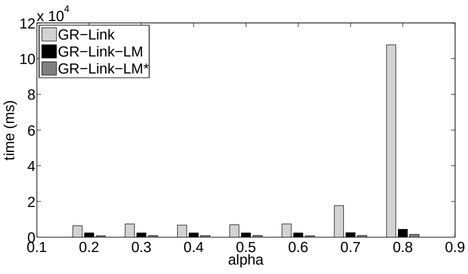
Ultimately, Figure 3.18a shows that graph-partitioning does not provide a performance advantage. What is interesting is that the performance advantage of the ideal method when compared to *GR-Link* is diminished in this case. An explanation here is that the expansion solely relies on the graph and, hence, a respective partitioning would provide some (respective) advantage for this method. Overall however, the ideal method performs best in the case of the Spatial Grid, which is evident when comparing Figures 3.10b and 3.18b.

3.5.8 Summary

The experiments show that by optimizing graph search, the resulting algorithms, *GR-Link-LM* and especially *GR-Link-LM** manage to improve the

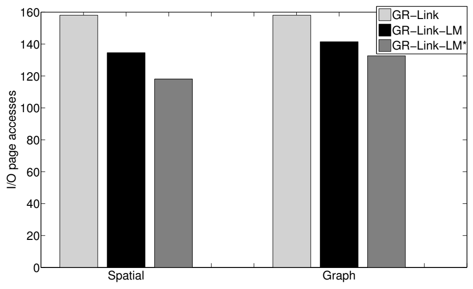


(a) I/O Performance

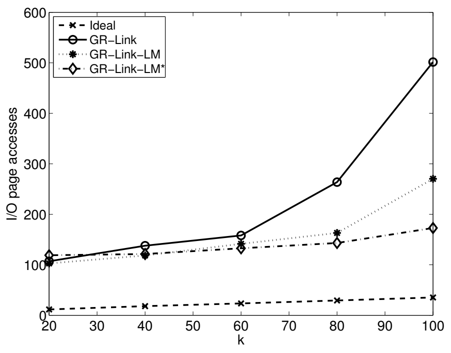


(b) CPU Performance

Figure 3.17: Varying preference parameter α



(a) k=60



(b) Graph partitioning, varying k

Figure 3.18: Partitioning: Spatial Grid vs. Relevance Graph

performance of the *GR-Link* method considerably. The two proposed algorithms perform also very well in comparison to an "ideal" method.

Chapter 4

Similarity Search on Spatio-Textual Point Sets

4.1 Introduction

Online social platforms, such as Twitter, Flickr, Facebook and Foursquare have attracted billions of active users. For example, according to Twitter, 500 million tweets are exchanged every day from 100 million active users. User activities in these platforms generate content that in most cases is either textual (e.g., status updates, short messages), or has textual information associated with it (e.g., tags assigned to uploaded photos). At the same time, following the widespread use of GPS and mobile devices, an increasing amount of this content is additionally associated with geospatial information (e.g., geotagged tweets, geotagged photos, user check-ins at specific places). Thus, user actions generate "traces", where each trace is an entity that can be associated with textual content and a location.

Efficient indexing and querying of spatio-textual data has received a lot of attention over the past years, due to the high importance of such content in location-based services, such as nearby search and recommendations. In particular, multiple types of spatio-textual queries have been extensively studied, including boolean range queries, top-k queries, k-nearest neighbor queries, and more recently, spatio-textual similarity joins [CCJW13, BGM12].

Nevertheless, in existing works spatio-textual entities are typically treated as *individual, independent items.* A typical example is a query to find nearby restaurants, or hotels offering certain amenities. The work presented in [BGM12] deals with finding pairs of entities that are both spatially close and textually similar. This work is highly applicable, such as for de-duplicating Points of Interest across datasets, or finding matching photos taken at ap-

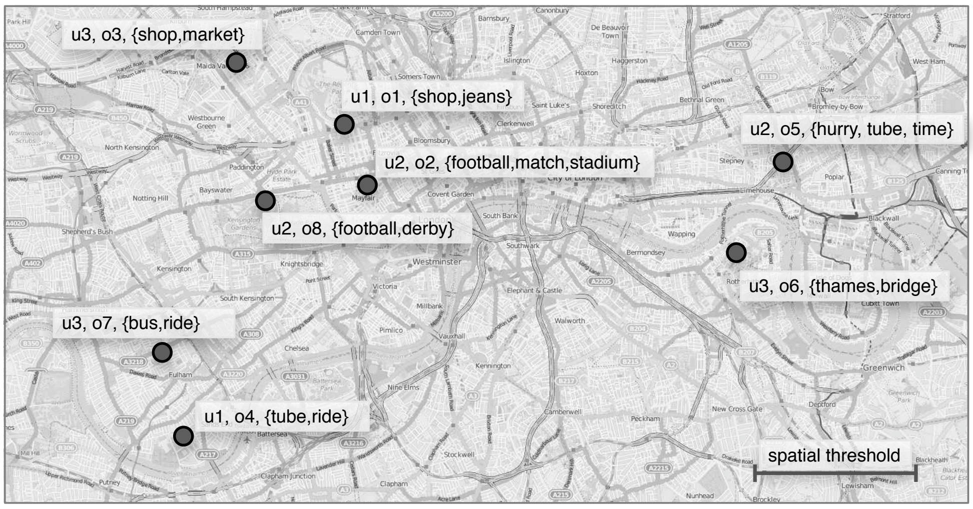


Figure 4.1: STPSJoin query scenario. Multiple objects are spatially or textually similar, but only users u_1 and u_3 have objects which are mutually similar.

proximately the same location and with very similar tags.

Similarity search on isolated entities can be restrictive. Consider, for example, searching for similar users in social networks. Users (entities) are associated with sets of spatio-textual objects (i.e. geotagged photos, or tweets the user has generated). Clearly, characterising two users as similar requires that their *sets of spatio-textual objects* are "similar".

Motivated by such applications, in this paper we address the problem of similarity search for spatio-textual entities, where each entity is represented by a set of spatio-textual objects. In particular, we introduce the *Spatio-Textual Point-Set Similarity Join* (STPSJoin) query. Given sets of spatio-textual objects, each one belonging to a specific entity, this query seeks pairs of entities that contain similar spatio-textual objects. The spatio-textual point set similarity join can naturally model, for example, the search for users, which exhibit similar behavior according to the spatio-textual objects they generate. For instance, given the locations and nature of store chains as spatio-textual objects, we can formalize the problem of identifying similar franchises and support site analysis for a franchise expansion.

To efficiently process spatio-textual point-set similarity joins, we adapt and extend the state-of-the-art algorithms for processing similarity joins for single points [BGM12]. The proposed algorithms make use of spatio-textual indexes in conjunction with an early termination and a filter-and-refinement strategy to effectively prune the search space, reducing the execution time by orders of magnitude. More specifically, the contributions of our work are as follows.

- We formally define the spatio-textual point set similarity join (`STPSJoin`) query, which extends and generalizes the spatio-textual similarity join for the case of point sets.
- We derive a baseline algorithm for the `STPSJoin` query by adapting the state-of-the-art `PPJ-C` algorithm to work for point sets.
- We propose two optimized algorithms, `S-PPJ-B` and `S-PPJ-F`, which apply an early termination and a filter-and-refinement strategy, respectively, in order to drastically prune the search space. This significantly reduces the number of comparisons required, both, in terms of pairs of entities and in terms of individual points for each candidate pair.
- In addition, we present an alternative version of `S-PPJ-F`, denoted as `S-PPJ-D`, which relies on an R-tree instead of a grid for the spatial indexing.
- Finally, we perform an extensive experimental evaluation using three large, real-world datasets. The results of the experimental evaluation demonstrate that the proposed algorithms achieve an order of magnitude and above improvement in terms of execution time when compared to the baseline method.

The rest of the paper is structured as follows. The `STPSJoin` query is formally introduced in Section 4.2. Then, our algorithms for the efficient evaluation of `STPSJoin` queries are presented in Section 4.3. Section 4.4 presents an experimental evaluation of our proposed approaches.

4.2 Problem Definition

We assume a database $\mathcal{D}$ of spatio-textual objects created by different users U. A spatio-textual object $o \in \mathcal{D}$ is a triple $o = \langle u, loc, doc \rangle$, where $u \in U$ is the user associated with this object, $loc = \langle x, y \rangle$ is a spatial point and doc is a set of keywords. We refer to the user, location and keywords associated with an object o using the notation $o.u$, $o.loc$ and $o.doc$ respectively. In addition, we use D_u to denote the set of objects belonging to user u.

The spatial distance $\delta(o, o')$ between two objects is calculated as the Euclidean distance between their spatial locations. Moreover, the textual similarity $\tau(o, o')$ is measured according to the Jaccard similarity of their keywords:

$$\tau(o, o') = \frac{|o.doc \cap o'.doc|}{|o.doc \cup o'.doc|}\,.$$

Given a spatial threshold ϵ_{loc} and a textual threshold ϵ_{doc}, we say that two objects $o, o' \in \mathcal{D}$ match if their spatial distance is below ϵ_{loc} and their textual similarity is above ϵ_{doc}. Matching between objects is defined using the predicate μ:

$$\mu(o, o') = \begin{cases} \textit{True} & \textit{if } \delta(o, o') \leq \epsilon_{loc} \textit{ and } \tau(o, o') \geq \epsilon_{doc} \\ \textit{False} & \textit{otherwise.} \end{cases}$$

For brevity, we overload μ to account for matching an object with a set of objects $D \subseteq \mathcal{D}$:

$$\mu(o, D) = \begin{cases} \textit{True} & \textit{if there exists } o' \in D \textit{ such that } \mu(o, o') \\ \textit{False} & \textit{otherwise.} \end{cases}$$

Furthermore, let two spatio-textual point-sets D and D'. Function $M(D, D')$ returns the set of objects in D that match with at least one object in D':

$$M(D, D') = \{o \in D \textit{ such that } \mu(o, D')\}\,.$$

We then use M to define the similarity of point-sets D and D'. In particular, this is measured as the fraction of the matched points from one set to the other divided by the total number of points in the two sets. Formally:

$$\sigma(D, D') = \frac{|M(D, D')| + |M(D', D)|}{|D| + |D'|}\,.$$

The employed measure is inspired by the Jaccard similarity, which is not directly applicable since it does not support partial similarity between elements. More elaborate similarity metrics over point sets can be found in [EM97, RB01].

We can now define the *Spatio-Textual Point Set Join* query (`STPSJoin`). `STPSJoin` identifies all pairs of users U which are associated with sets of spatio-textual objects that have a match higher than a specified threshold ϵ_u. We assume a total ordering over U (i.e. $\prec_U$) to avoid returning duplicate pairs. Formally, the `STPSJoin` query is defined as follows.

Definition 1. *Given a database $\mathcal{D}$ of spatio-textual objects belonging to a set of users U, the `STPSJoin` query is a tuple $Q = \langle \epsilon_{loc}, \epsilon_{doc}, \epsilon_u \rangle$ which returns a set R containing all pairs of users (u, u') such that $u, u' \in U$, $u \prec u'$, and $\sigma(D_u, D_{u'}) \geq \epsilon_u$ with respect to the spatial and textual thresholds ϵ_{loc} and ϵ_{doc}.*

4.3 Algorithms for STPSJoin

In this section, we present our algorithms for the evaluation of the `STPSJoin` query. We first present a baseline algorithm, and then we introduce methods that exploit filter and refine strategy in combination with spatio-textual indexes in order to direct the search for similar point sets.

4.3.1 Baseline Approach

Preliminaries. The straightforward method for evaluating an `STPSJoin` query is to find, for every pair of users, the set of matching objects, and then to check whether the resulting similarity score σ exceeds the specified threshold ϵ_u. Thus, for a pair of users (u, u'), the problem can be cast as a spatio-textual similarity join query, `ST-SJOIN`$(D, \epsilon_{loc}, \epsilon_{doc})$, which has been studied in [BGM12]. This query returns all pairs of objects (o, o') in D such that $o, o' \in D$, $\delta(o, o') \leq \epsilon_{loc}$ and $\tau(o, o') \geq \epsilon_{doc}$. Based on this, we can find the objects of u that match with those of u', and vice versa, and then proceed with computing the score σ for this pair of users.

For this purpose, we adopt the PPJ-C algorithm from [BGM12] for the purposes of ST-SJOINs. PPJ-C uses a grid to partition the space, in order to limit the search to those candidates that can satisfy the spatial predicate of the join. The grid is constructed dynamically at query time, using cells that have an extent in each dimension that equals the spatial distance threshold ϵ_{loc}. The cells are assigned ids in a row-wise order from bottom to top (see Figure 4.2a).

PPJ-C visits the cells in ascending order of their ids, taking advantage of the spatial filtering, since the objects in each visited cell c need to be joined only with those in c and in the cells adjacent to c. In fact, to avoid duplicates, only the adjacent cells with ids lower than c need to be examined. Thus, for each cell, one self-join operation and at most four non-self join operations need to be performed. These are performed using the PPJ algorithm, that in turn extends the set similarity join algorithm PPJOIN [XWL+11] by including an additional check on the spatial distance of two objects.

The `S-PPJ-C` algorithm. Using `PPJ-C` as basis, we can derive a baseline algorithm, denoted as `S-PPJ-C` (<u>S</u>et-PPJ-C), for the `STPSJoin` query. `S-PPJ-C` is presented in Algorithm 6. During the construction of the grid, we maintain the following additional information: (a) for each cell c, we maintain the contained objects in separate lists according to the user they belong to; we denote by D_c^u the set of objects of user u that are contained in c; (b) for every user u, we maintain a list of cells C_u that contain objects belonging to u; C_u is sorted according to cell ids in ascending order.

ALGORITHM 6: S-PPJ-C Algorithm

```
Input: $D$, $U$, $\epsilon_{doc}$, $\epsilon_{loc}$, $\epsilon_u$
Output: Pair of matched users $R$
1  $R \leftarrow \emptyset$
2  $selectedUsers \leftarrow \emptyset$
3  $G \leftarrow createGridIndex(D, U, \epsilon_{loc})$
4  foreach $u_1 \in U$ do
5      foreach $u_2 \in selectedUsers$ do
6          $r \leftarrow PPJ\text{-}C(u_1, u_2, \epsilon_{doc}, \epsilon_{loc})$
7          $\sigma \leftarrow \frac{|r|}{|D_{u_1}|+|D_{u_2}|}$
8          if $\sigma \geq \epsilon_u$ then
9              $R.add(\langle u_2, u_1 \rangle)$
10     $selectedUsers.add(u_1)$
11 return $R$
```

The `S-PPJ-C` algorithm loops through all pairs of users, taking into consideration the total ordering $\prec_U$ of the user set U. For each pair of users (u, u'), `S-PPJ-C` executes a non-self join version of the `PPJ-C` algorithm from [BGM12] presented above. The difference with the standard `PPJ-C`, lies in the fact that in this case pairs with objects from both users are returned. To do so, first the lists C_u and $C_{u'}$ containing the cells for u and u' respectively are gathered. Next, the algorithm iteratively selects from either list the cell c with the lowest id that has not been selected yet. Assume that the next selected cell c is from the list of user u. For every cell c' in $C_{u'}$ with $c'.id \geq c.id$, a non-self join version of `PPJ` is executed with input the spatio-textual point sets D_u^c and $D_{u'}^{c'}$. Since C_u and $C_{u'}$ may both contain the cell c, we avoid the duplicate execution of `PPJ` for c.

The results of `PPJ-C` are used to compute the user similarity score σ (line 6-7). Pairs of users that achieve a similarity score above the threshold ϵ_u are collected in the result set.

4.3.2 The S-PPJ-B Algorithm

The drawback of the `S-PPJ-C` algorithm is that for each pair of users it finds all their matching points and computes the exact value of their similarity score σ before checking whether this exceeds the given threshold. Instead, since we are only interested in finding those pairs with a similarity that exceeds ϵ_u, we can reduce the execution time of the algorithm by terminating the computation for a pair of users as soon as it can be decided that their similarity is below ϵ_u. Following this observation, we derive a more efficient algorithm, denoted as `S-PPJ-B` (where B stands for bound).

`S-PPJ-B` operates in the same manner as `S-PPJ-C`, with the only difference that it replaces the execution of `PPJ-C` with a modified process, denoted as `PPJ-B`. `PPJ-B` leverages the use of an upper bound on the number of unmatched objects for a pair of users to effectively prune the search on the spatial grid. More specifically, the intuition behind `PPJ-B` is the following. While examining two users, `PPJ-B` leverages the user similarity threshold ϵ_u and the number of objects belonging to each user in order to compute an upper bound on the number of unmatched objects between the two users, above which the user similarity cannot exceed ϵ_u. In the following, we first derive this upper bound, and then we explain the process followed by `PPJ-B` in order to allow for early termination during the examination of two users.

For a pair of users (u, u'), let $\beta_{u,u'}$ denote the number of objects from user u and user u' that do not match with the other user, i.e.:

$$\beta_{u,u'} = |D_u| + |D_{u'}| - |M(D_u, D_{u'})| - |M(D_{u'}, D_u)|$$

An upper bound for $\beta_{u,u'}$ is derived as follows.

Lemma 1. *For a pair of users (u, u'), if $\beta_{u,u'} > (1 - \epsilon_u) \cdot (|D_u| + |D_{u'}|)$ then $\sigma(D_u, D_{u'}) < \epsilon_u$.*

Proof. The proof is derived from the definition of the similarity score between two users, as follows:

$$\begin{aligned}
&\sigma(D_u, D_{u'}) \geq \epsilon_u \Rightarrow \frac{|M(D_u, D_{u'})| + |M(D_{u'}, D_u)|}{|D_u| + |D_{u'}|} \geq \epsilon_u \Rightarrow \\
&\frac{|D_u| + |D_{u'}| - \beta_{u,u'}}{|D_u| + |D_{u'}|} \geq \epsilon_u \Rightarrow 1 - \frac{\beta_{u,u'}}{|D_u| + |D_{u'}|} \geq \epsilon_u \Rightarrow \\
&\beta_{u,u'} \leq (1 - \epsilon_u) \cdot (|D_u| + |D_{u'}|)
\end{aligned}$$

□

Upon traversing a cell c, `PPJ-C` checks for potential matches in cells with ids lower than $c.id$. Therefore, we cannot be certain that objects that have not been matched so far will also not match with objects in cells with higher ids (i.e. in the next cell or row). Therefore, the bound may be used within `PPJ-C`, but only with respect to the objects discovered from the beginning of the grid until the previous row. The objects that were traversed in the current row have to be excluded from calculation.

To that end, PPJ-B devises a different grid traversal strategy that allows the pruning mechanism to utilize every object appearing in cells traversed when the bound evaluation is executed. Specifically, this strategy traverses the rows from bottom to top (considering the id of the bottom row as 1), and depending on whether the id of a row is odd or even, different treatment is followed. If a cell $c_{i,j}$ belongs to a row with odd id, i.e. j is odd, then the objects contained in it are matched with objects from all surrounding cells, except the cell directly on the right, i.e. $c_{i+1,j}$. Matching is done by executing PPJoin. Otherwise, if the cell belongs to an even row, then we match its objects only with objects from the other user from the cell that is directly on the left, i.e. $c_{i-1,j}$. This process is illustrated in Figure 4.2b.

PPJ-B is described in Algorithm 7. Following this traversal strategy, PPJ-B allows for early termination using the bound β, while still maintaining the property of PPJ-C to avoid duplicate examination of the same pair of cells. Indeed, when PPJ-B traverses the last cell of an odd row (lines 13-16), it has considered every potential match for any object it has encountered up to that point. Thus, it checks whether the number of objects that have not been matched exceeds the calculated bound β. If so, the search stops, since it is impossible to result in a user similarity score that exceeds ϵ_u. Note that, in practice, since the grid may be rather sparse, some rows may be empty. In that case, when the next visited cell belongs to a row that is not directly above the previous one, the same check can be performed, even if the last examined row was even, since previously encountered objects cannot have any new matches in the future.

4.3.3 The S-PPJ-F Algorithm

The S-PPJ-B algorithm presented above exploits an upper bound on the number of unmatched objects between two users in order to allow for early termination when comparing each pair of users. In the following, we present the S-PPJ-F algorithm that further increases efficiency by following a filter and refine strategy that concentrates the search on those pairs of users that are promising candidates, while pruning others that can not exceed the similarity threshold ϵ_u.

S-PPJ-F is outlined in Algorithm 8. It operates on top of a spatio-textual index structure that is constructed at runtime. In every iteration, the algorithm selects a new user u, searches for potential matches with the users that have been selected in previous steps, and updates the spatio-textual index with the objects in D_u.

The spatio-textual index is a dynamic grid enhanced with an inverted index for every cell. This list maintains for every token that appears in objects in a cell, the users that are associated with these objects. An example is depicted in Figure 4.3. The grid structure additionally maintains the objects associated with every user within a cell.

The search for matches follows the filter and refine principle. After a user u is selected, the algorithm traverses through every cell $c \in C_u$ which contains objects associated with u, and calculates the set of tokens T that appear in any one of these objects. This set is then utilized to identify candidate users in c and its surrounding cells (lines 6-9). Every user u' that is associated with an object that appears in one of these cells and contains at least one keyword from T is considered to be a candidate. `S-PPJ-F` maintains every cell for u and u' that contains objects that potentially (both spatially and textually) match in $M^u_{u'}$ and $M^{u'}_{u'}$ respectively.

For every user u and candidate user u', the algorithm calculates an upper bound $\bar{\sigma}$ of their user similarity score (lines 12-13). This is performed by assuming that all of their objects which are contained in the same or adjacent cells match. Formally, $\bar{\sigma}$ is computed as follows:

$$\bar{\sigma} = \frac{\sum_{l \in M^u_{u'}} |D^l_u| + \sum_{l' \in M^{u'}_{u'}} |D^{l'}_{u'}|}{|D_u| + |D_{u'}|}.$$

If $\bar{\sigma} < \epsilon_u$, then this pair can be safely pruned. Otherwise, a refinement step follows, during which the `PPJ-B` algorithm is executed to identify whether the exact similarity score for the pair exceeds the user similarity threshold.

4.3.4 The S-PPJ-D Algorithm

In the following, we consider databases that are already partitioned by a data partitioning scheme. In particular, we consider data partitioning schemes induced by an R-tree structure combined with a textual index similar in fashion to the index outlined with respect to `S-PPJ-F`. The main difference is that instead of indexing grid cells, in this case, we index the leaf nodes of the R-tree.

`S-PPJ-D` implements a filter and refinement strategy similar to `S-PPJ-F`, based on a given data partitioning and a spatio-textual index I that is constructed at runtime. I maintains an entry for every leaf node l in the tree. This entry holds an inverted list that maps a token t U^l_t (i.e. users with objects in l that contain t). In addition, every leaf node l maintains a mapping between users and their objects within l, denoted by D^l_u. Finally, the intersections among the extended MBRs of the leaf nodes in the tree are precomputed by performing a spatial join using the process described in [BKS93]. `S-PPJ-D` is described by Algorithm 9.

The filter step iterates over the leaf nodes L_u of a user u. For every leaf node l, it calculates the set of tokens T that appear in objects within l that are associated with u (i.e. D_u^l). These tokens are then used to probe the spatio-textual index (line 8) and identify the candidate users that are associated with objects containing tokens from T. This is performed for each leaf node that intersects with the ϵ_{loc}-extended bounding box of l (line 6). In order to avoid duplicates, we only search for candidate users which are higher in the user ordering. M maintains for every candidate u' the leaf nodes of u' contain objects that can potentially match objects associated with user u $M_{u'}^{u'}$, as well as the leaf nodes of the relevant objects from u $M_{u'}^{u}$. `S-PPJ-D` then calculates for every candidate u' a bound on the similarity score between u and u'. This bound is calculated by considering the extreme case in which every object from $M_{u'}^{u'}$ and $M_{u'}^{u}$ match (lines 11-12). The refinement step uses `PPJ-D` in order to calculate the exact similarity between candidate users.

Algorithm 10 outlines `PPJ-D`. `PPJ-D` leverages the spatio-textual index in combination with an appropriate leaf node traversal strategy in order to return the similarity score between two users. `PPJ-D` functions similar to `PPJ-B` for the context of a data-driven partitioning scheme. Given two users u_1 and u_2, two lists L_1 and L_2 are maintained for their leaf nodes ordered with respect to a predefined ordering (e.g. in ascending order of their ids). The algorithm proceeds iteratively, and selects the lowest (with respect to the ordering) unvisited leaf node l from L_1 and L_2.

Let user u be the user from which the element was selected, and u' the other user. The index is used to identify every leaf node l' that is spatially relevant to l, and contains objects from u'. Spatially relevant leaf nodes are nodes with intersecting ϵ_{loc}-extended MBRs. For every l' we execute `PPJoin` to identify the exact similarity between the objects D_u^l and $D_{u'}^{l'}$. This is performed by focusing only in objects that belong within the intersection A of the ϵ_{loc}-extended MBRs of l and l' (lines 11-12, 18-19). This optimisation is based on the observation that objects which are not contained in A do not satisfy the spatial threshold ϵ_{loc}.

`PPJ-D` follows a similar pruning strategy with `PPJ-B`. The objects of every leaf node for user u are evaluated against every potential candidate from $D_{u'}$ that falls within a leaf node that is higher in the given ordering. Therefore, after an iteration that visits an object, candidate matches from leaf nodes, both higher and lower in the ordering, are considered. This observation is the basis of a pruning step (lines 21-22) that calculates the number of objects $t - |J|$ that are already found to fail to satisfy the thresholds. If this number is lower that a computed bound, the search is pruned since the users fail to satisfy the user similarity threshold.

4.4 Experimental Evaluation

Next, we present our experimental evaluation of the proposed algorithms. We first describe the datasets used and the parameters involved, and then we present the results.

4.4.1 Experimental Setup

Datasets. We have used three real-world datasets of spatio-textual web objecs for our experiments. The *Flickr* dataset is derived from the Flickr Creative Commons dataset provided by Yahoo [TSF+15]. The whole dataset contains about 99.3 million images, about 49 million of which are geotagged. For our experiments, we concentrate on objects from the geographical boundaries of London, UK and we filtered out images that do not contain coordinates or tags. The resulting dataset contains 17,008 users and 1,189,335 objects. The *GeoText* dataset [EOSX10] is a geotagged microblog corpus available online.[1] It comprises 377,616 posts by 9,475 different users within the US. Finally, the *Twitter* dataset is a collection of geotagged tweets from the geographical area of London, UK, that we have collected. It contains 9,724,579 tweets generated by 40,000 different users in 2014.

The NLTK toolkit[2] was employed to identify named entities from the text associated with the objects. The extracted named entities were used in combination with other related information, such as tokens, hashtags and mentions, as keywords associated with the respective objects. The characteristics of the three datasets are summarized in Table 4.1.

Evaluation measures and parameters The purpose of the experimental evaluation is to compare the performance of the proposed algorithms in terms of the execution time in different settings. We investigate the effect of the following parameters: (a) the dataset size N in terms of number of users, (b) the query thresholds for spatial distance (ϵ_{loc}), textual similarity (ϵ_{doc}) and user similarity (ϵ_u) and (c) the *fanout* parameter of the R-tree structure.

All algorithms were implemented in Java, and the experiments were executed on a machine with an Intel Core i5 2400 CPU and 16GB RAM, running on Ubuntu Linux. During the experiments, 15GB of memory were allocated to the JVM. All plots report running time in a logarithmic scale.

[1]`http://www.ark.cs.cmu.edu/GeoText/`

[2]`http://www.nltk.org/`

4.4.2 Scalability

The scalability experiments evaluate the performance of our methods in datasets of different sizes. We divided the Twitter, Flickr and GeoText datasets for a variable number of users. The resulting datasets range from 4,000 users with 72,094 objects to 40,000 users with 9,724,579 objects. Different parameter values are used for different datasets, in order to account for the different sizes and token selectivity across the datasets. Lower thresholds are set for the GeoText dataset in order to avoid empty result sets, whereas higher thresholds are set for the Flickr dataset to account for the increased amount of textual information.

Figure 4.4 shows the scalability evaluation results. The results clearly show that `S-PPJ-F` outperforms the other methods by several orders of magnitude. This is consistent for all datasets, irrespective of size. The efficiency of `S-PPJ-F` compared to the other approaches is attributed to the effect of the filter and refinement scheme, in combination with the suitability of the dynamic grid partitioning over the objects. The grid partitioning is tailor made to the spatial threshold parameter ϵ_{loc}, which allows the search for matching objects to be limited exclusively in adjacent cells. Additionally, the inverted lists maintained within each cell of the grid, allow the effective filtering of candidate user pairs associated with spatially similar, but textually diverse, objects.

The performance of `S-PPJ-B` does not compare favourably against `S-PPJ-F`. This result is expected since `S-PPJ-F` builds on `S-PPJ-B` by leveraging the filter and refinement scheme. The comparison between `S-PPJ-B` and `S-PPJ-C`, allows the evaluation of the early termination strategy, as well as the traversal mechanism, differentiating `S-PPJ-B` from `S-PPJ-C`. The results indicate that `S-PPJ-B` offers a consistent improvement in execution time compared to `S-PPJ-C`, confirming that the proposed techniques manage to prune the search space for similarity search among two point sets.

Finally, the results show that `S-PPJ-D` outperforms the baseline methods, but it is not comparable to the grid-based `S-PPJ-F`, which follows the same principles. The discrepancy in execution time can be attributed to the use of different spatial indexes. The data driven-partitioning imposed by the R-tree is independent of the spatial threshold given as a parameter to the `STPSJoin` query. As a result, the imposed partitioning leads to an ineffective division of the database. Inspection of the performance of `S-PPJ-D` shows that both partition size and overlap may lead to subpar performance, since objects within large partitions tend to be spatially irrelevant, and overlaps require the evaluation of multiple join operations. We revisit this issue in Section 4.4.4.

4.4.3 Effect of similarity thresholds

In the following experiments, we vary the parameters and evaluate the proposed algorithms for different combinations of textual, spatial and user similarity thresholds. Similar to the scalability experiments, different ranges in threshold values are used across datasets.

Figure 4.5 presents the results. We observe that the dominant parameter is the spatial threshold ϵ_{loc}. High values on ϵ_{loc} result in significantly higher execution times. This is particularly obvious for the Flickr and Twitter datasets, which contain significantly larger amounts of objects. When the spatial distance threshold reaches metropolitan level distances, the majority of the objects fall into adjacent partitions. As a result, the filtering step of `S-PPJ-F` and `S-PPJ-D` returns a high number of candidates. In these cases, the overhead imposed by the additional indexing maintained by `S-PPJ-F` and `S-PPJ-D` is apparent. We observe a peak in the case of `S-PPJ-D`, especially with respect to the Flickr dataset. In this case, inspection shows that the R-tree partitioning does not manage to result in an efficient partition of the object database.

This does not apply for GeoText, mainly due to the fact that the objects in GeoText are scattered in the significantly larger area of the whole of USA. The results show that the proposed pruning strategies are highly functional in combination with a grid-based partitioning scheme. `S-PPJ-F` outperforms the other methods in every scenario, and apart from the case of the Flickr dataset with $\epsilon_{loc} = 0.01$, its performance is independent of the parameter values.

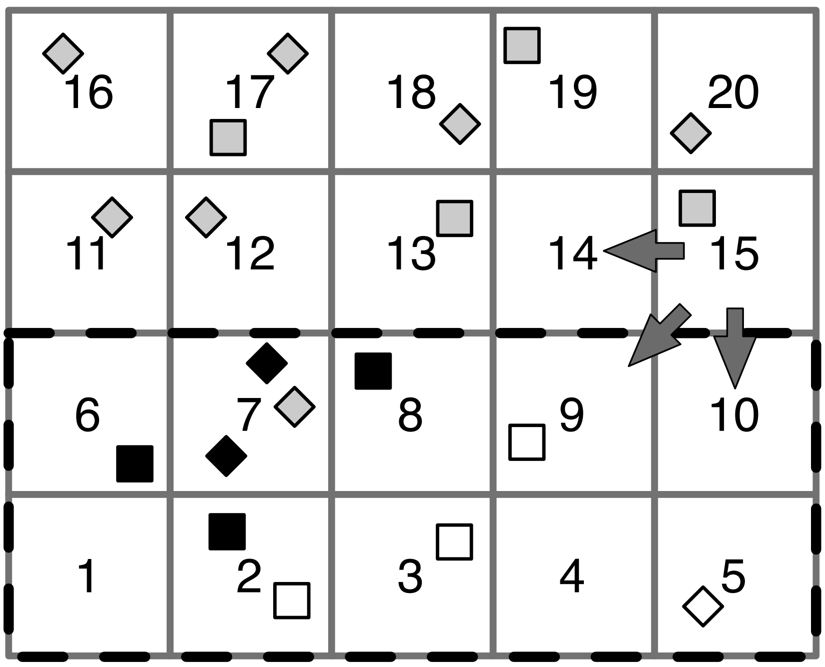
4.4.4 Effect of Fanout on S-PPJ-D

An important parameter for data partitioning schemes based on R-trees is the *fanout* parameter. This parameter is associated with the number of objects that reside in a node of the R-tree. The effect the fanout parameter has on the performance of `S-PPJ-D` is twofold. First of all, `S-PPJ-D` executes a spatial distance join in order to identify spatial relations among the leaf nodes of the tree, which are treated by the algorithm as spatial data partitions. Since the fanout parameter affects both the depth and the breadth of the R-tree, it also affects the performance of the spatial join. Second, `S-PPJ-D` is build on top of the partitioning imposed by the leaf nodes. Therefore, the fanout affects both the number and the size of leaf nodes, which are relevant to `S-PPJ-D`.

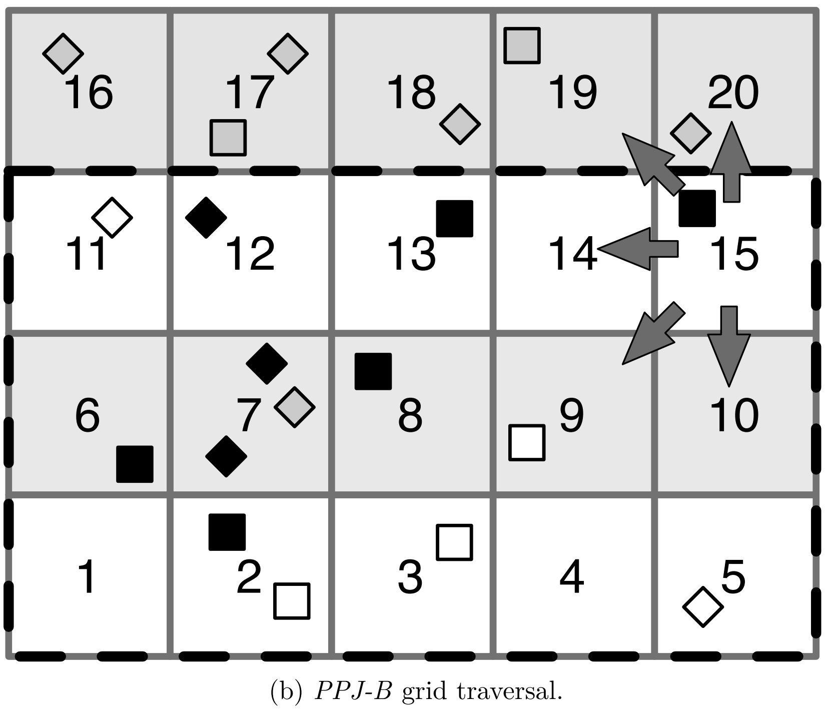
In order to experimentally evaluate the effects of the fanout parameter, experiments with values ranging from 50 to 250 were conducted. The results are shown in Figure 4.6. The results verify that S-PPJ-D is sensitive to the fanout value. Even though no single fanout value achieves the best results in all datasets, we observe that an appropriate fanout value for STPSJoin queries falls within the range of 100 to 200.

4.4.5 Summary

Our experimentation verifies the superiority of the proposed algorithms in terms of execution time for all datasets used. The pruning strategy employed by S-PPJ-F manages to significantly boost the algorithm's performance for the computation of the STPSJoin query. Furthermore, the S-PPJ-D algorithm which induces data partitioning is efficient enough to be considered as a viable choice in cases when the data are already partitioned with an R-tree (or any other data partitioning method). The experimental analysis is conducted on three real datasets of varied size (the two of them are publicly available) and different parameters setting have been examined in order to reach to the optimum configurations. The results show that the algorithms scale well in very large databases and can therefore be used effectively in real-world scenarios.



(a) *PPJ-C* grid traversal.



(b) *PPJ-B* grid traversal.

Figure 4.2: *PPJ-C* and *PPJ-B* grid traversal strategies examples. Objects associated with user u (u') are depicted by squares (diamonds) respectively. Object colors represent whether the search has determined if an object matches with objects from the other user. Matched objects are painted black, objects that do not match are painted white, while objects whose state has not been determined are painted grey. *PPJ-B* has determined the state of every object in cells 1 to 15, while *PPJ-C* the objects until cell 10.

ALGORITHM 7: PPJ-B Algorithm

```
Input: $D_{u_1}$, $D_{u_2}$, $G$, $\epsilon_{doc}$, $\epsilon_{loc}$, $\epsilon_u$
Output: Similarity score for users $u_1$, $u_2$
1  $\beta \leftarrow (1 - \epsilon_u) \cdot (|D_{u_1}| + |D_{u_2}|)$
2  $J \leftarrow \emptyset$
3  $C_{u_1} \leftarrow G.getUserCells(u_1)$
4  $C_{u_2} \leftarrow G.getUserCells(u_2)$
5  $t \leftarrow 0$
6  $c_1 \leftarrow -1$, $c_2 \leftarrow -1$
7  while $c_1 < |C_{u_1}| - 1$ or $c_2 < |C_{u_2}| - 1$ do
8      if $C_{u_1}[c_1 + 1] \leq C_{u_2}[c_2 + 1]$ then
9          $c_1 \leftarrow c_1 + 1$, $c \leftarrow c_1$, $u_c \leftarrow u_1$
10     else
11         $c_2 \leftarrow c_2 + 1$, $c \leftarrow c_2$, $u_c \leftarrow u_2$
12     $r \leftarrow G.getRow(c)$
13     if ($r_{-1}$ is odd and $r > r_{-1}$) or
14             ($r_{-1}$ is even and $r - r_{-1} > 1$) then
15         if $t - |J| > \beta$ then
16             return $0$
17     if $r$ is odd then
18         $C_{rel} \leftarrow G.getRelevantCellsExceptRight(c)$
19         foreach $c' \in C_{rel}$ do
20             if $c = c'$ and $c_{-1} = c$ then continue
21             else if $u_c = u_1$ then
22                 $PPJoin(D^c_{u_1}, D^c_{u_2}, J)$
23             else
24                 $PPJoin(D^{c'}_{u_2}, D^c_{u_1}, J)$
25     else
26         if $c \neq c_{-1}$ then
27             $PPJoin(D^c_{u_1}, D^c_{u_2}, J)$
28         if $G.getColumn(c) \neq 0$ then
29             if $u_c = u_1$ then
30                 $PPJoin(D^c_{u_1}, D^{c-1}_{u_1}, J)$
31             else
32                 $PPJoin(D^c_{u_2}, D^{c-1}_{u_1}, J)$
33     if $c \neq c_{-1}$ then
34         $t \leftarrow t + |D^c_{u_1}| + |D^c_{u_2}|$
35     $r_{-1} \leftarrow r$, $c_{-1} \leftarrow c$
36 $\sigma \leftarrow |J|/(|D_{u_1}| + |D_{u_2}|)$
37 if $\sigma \geq \epsilon_u$ then return $\sigma$
38 else return $0$
```

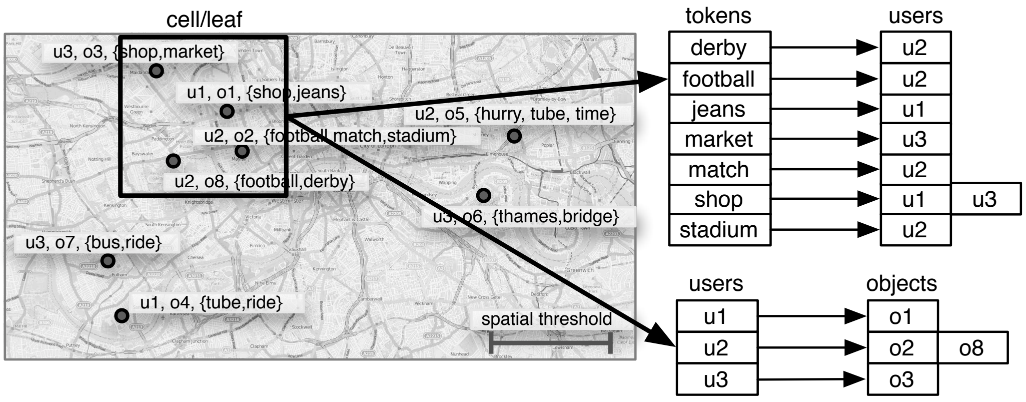


Figure 4.3: Spatio-textual structure for S-PPJ-F and S-PPJ-D.

ALGORITHM 8: *S-PPJ-F* Algorithm

```
Input: $D$, $U$, $\epsilon_{doc}$, $\epsilon_{loc}$, $\epsilon_u$
Output: Pair of matched users $R$
1  $R \leftarrow \emptyset$
2  $G \leftarrow initialiseSTGridIndex(D, \epsilon_{loc})$
3  foreach $u \in U$ do
4      foreach $c \in C_u$ do
5          $T \leftarrow calculateTokens(u, c)$
6          foreach $c' \in G.getRelevantCells(c)$ do
7              foreach $t \in T$ do
8                  foreach $u' \in G.getTokenUsers(c', t)$ do
9                      $M^u_{u'}.add(c)$, $M^{u'}_{u'}.add(c')$
10     $G.addUser(u)$
11     foreach $u' \in M.keys()$ do
12         $m \leftarrow \sum_{c \in M^u_{u'}} |D^c_u| + \sum_{c' \in M^{u'}_{u'}} |D^{c'}_{u'}|$
13         $\bar{\sigma} \leftarrow \frac{m}{|D_u| + |D_{u'}|}$
14         if $\bar{\sigma} \geq \epsilon_u$ then
15             $\sigma \leftarrow PPJ\text{-}B(D_u, D_{u'}, G, \epsilon_{doc}, \epsilon_{loc}, \epsilon_u)$
16             if $\sigma \geq \epsilon_u$ then
17                 $R.add(\langle u', u \rangle)$
18 return $R$
```

ALGORITHM 9: *S-PPJ-D* Algorithm

```
Input: $Tree$, $\epsilon_{doc}$, $\epsilon_{loc}$, $\epsilon_u$
Output: Pair of matched users $R$
1  $R \leftarrow \emptyset$
2  $I \leftarrow constructSpatioTextualIndex(Tree)$
3  foreach $u \in I.getUsers()$ do
4      foreach $l \in I.getLeafNodes(u)$ do
5          $T \leftarrow I.calculateTokens(u, l)$
6          foreach $l' \in I.getRelevantLeafNodes(l)$ do
7              foreach $t \in T$ do
8                  foreach $u' \in U_t^{l'}$ such that $u < u'$ do
9                      $M_{u'}^{u}.add(l)$, $M_{u'}^{u'}.add(l')$
10     foreach $u' \in M.keys()$ do
11         $m \leftarrow \sum_{l \in M_{u'}^{u}} |D_u^l| + \sum_{l' \in M_{u'}^{u'}} |D_{u'}^{l'}|$
12         $\bar{\sigma} \leftarrow \frac{m}{|D_u| + |D_{u'}|}$
13         if $\bar{\sigma} \geq \epsilon_u$ then
14             $\sigma \leftarrow PPJ\text{-}D(D_u, D_{u'}, I, \epsilon_{doc}, \epsilon_{loc}, \epsilon_u)$
15             if $\sigma \geq \epsilon_u$ then
16                 $R.add(\langle u', u \rangle)$
17 return $R$
```

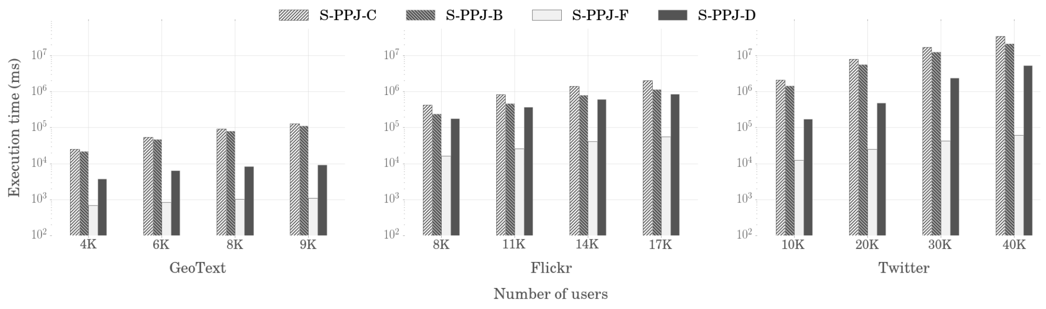


Figure 4.4: Scalability results for the GeoText, Flickr and Twitter datasets (parameter defaults: GeoText: $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.3$, $\epsilon_u = 0.3$; Flickr $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.5$, $\epsilon_u = 0.5$; Twitter: $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.4$, $\epsilon_u = 0.4$).

ALGORITHM 10: PPJ-D Algorithm

```
Input: $D_{u_1}$, $D_{u_2}$, $I$, $\epsilon_{doc}$, $\epsilon_{loc}$, $\epsilon_u$
Output: Similarity score for users $u_1$, $u_2$
$\beta \leftarrow (1 - \epsilon_u) \cdot (|D_{u_1}| + |D_{u_2}|)$
$J \leftarrow \emptyset$                                        // joined objects
$L_1 \leftarrow I.getLeafs(u_1)$                                 // sorted
$L_2 \leftarrow I.getLeafs(u_2)$
$i_1 \leftarrow 0$, $i_2 \leftarrow 0$
$t \leftarrow 0$
while $i_i < |L_1|$ or $i_2 < |L_2|$ do
    if $L_1[i_i] \leq L_2[i_2]$ then
        foreach $l_2 \in I.getRelevantLeafs(l_1)$ do
            if $l_2 \geq l_1$ and $l_2 \in L_2$ then
                $A \leftarrow I.extend(l_1, \epsilon_{loc}) \cap I.extend(l_2, \epsilon_{loc})$
                $PPJoin(D_{u_1}^{l_1} \cap A, D_{u_2}^{l_2} \cap A, J)$
        $t \leftarrow t + |D_{u_1}^{l_1}|$
    else if $L_{u_2}[i_2] \leq L_1[i_1]$ then
        $l_2 \leftarrow L_2[i]$
        foreach $l_1 \in I.getRelevantLeafs(l_2)$ do
            if $l_1 > l_2$ and $l_1 \in L_1$ then
                $A \leftarrow I.extend(l_1, \epsilon_{loc}) \cap I.extend(l_2, \epsilon_{loc})$
                $PPJoin(D_{u_1}^{l_1} \cap A, D_{u_2}^{l_2} \cap A, J)$
        $t \leftarrow t + |D_{u_2}^{l_2}|$
    if $t - |J| > \beta$ then
        return 0
    if $L_1[i_i] <= L_2[i_2]$ then $i_1 \leftarrow i_1 + 1$
    if $L_2[i_2] <= L_1[i_1]$ then $i_2 \leftarrow i_2 + 1$
$\sigma \leftarrow |J|/(|D_{u_1}| + |D_{u_2}|)$
if $\sigma \geq \epsilon_u$ then return $\sigma$
else return 0
```

Dataset	Objects	Users	Tokens per Object	Objects per Token	Objects per User
Twitter	9724579	40000	2.08 (1.43)	6.25 (141.80)	243.11 (344.86)
Flickr	1189335	17008	7.90 (8.03)	26.45 (1236.50)	69.93 (346.53)
GeoText	165733	9461	1.64 (1.01)	3.53 (39.36)	17.52 (12.99)

Table 4.1: Experimentation datasets, number of objects and users, and mean (standard deviation) for descriptive metrics.

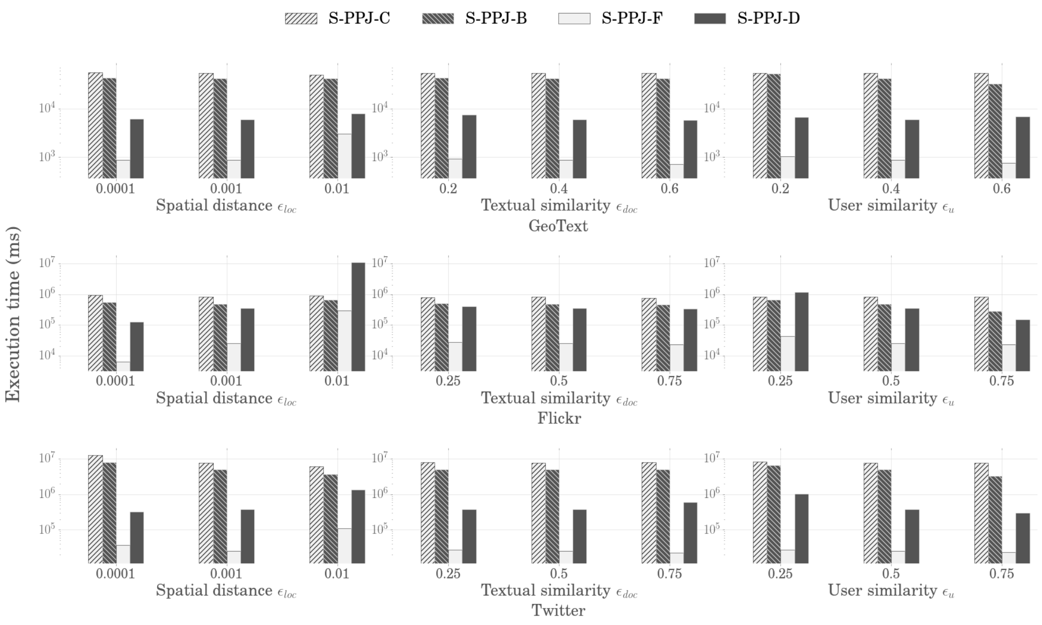


Figure 4.5: Varying the similarity thresholds (GeoText: 6, 000 users, 107, 941 objects; Flickr: 11, 000 users, 74, 8745 objects; Twitter: 20, 000 users, 4, 988, 090 objects).

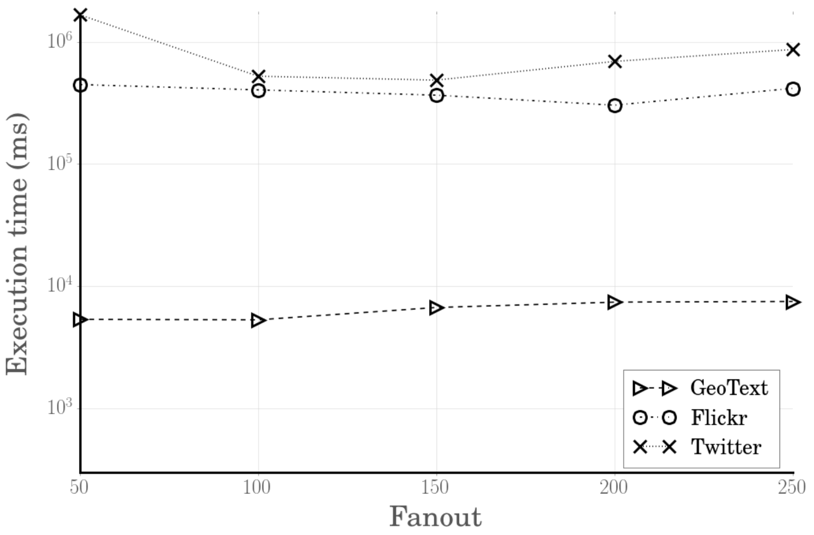


Figure 4.6: Tuning the R-Tree fanout parameter.

Chapter 5

Hub Labels on the Database for Large-Scale Graphs

5.1 Introduction

Answering distance queries on graphs is one of the most well-studied problems on algorithmic theory, mainly due to its wide range of applications. Although a lot of recent research focused exclusively on transportation networks (cf. [Bas14] for the most recent overview) the emergence of social networks has generated massive unweighted graphs of interconnected entities. On such networks, the distance between two vertices is an indication of their closeness, i.e., for finding users closely related to each other or extracting information about existing communities within the social media users. Although we may always use a breadth first search (BFS) to calculate the distance between any two vertices on such graphs, that approach cannot facilitate fast-enough queries on main memory or be easily adapted to secondary storage solutions.

Moreover, most of the excellent preprocessing techniques available for road networks cannot be adapted to large-scale graphs, such as social or collaboration networks. So far, the most promising approach for this type of graphs builds on the 2-hop labeling or hub labeling (HL) algorithm [GPPR01], [CHKZ02], in which we store a two-part label $L(v)$ for every vertex v: a forward label $L_f(v)$ and a backward label $L_b(v)$. These labels are then used to very fast answer point-to-point shortest-path queries. This technique has been adapted successfully to road networks [ADGW11, ADGW12, DGW13, AIKK14] and quite recently has also been extended to undirected, unweighted graphs [AIY13, DGPW14, JFWX14]. The HL method has also been applied for one-to-many, many-to-many and kNN queries in road networks [DGW11, DW13] and kNN and RkNN queries in the context of social networks in [EP14b].

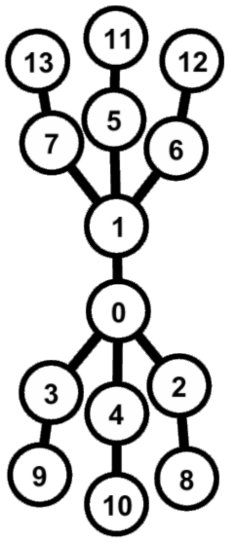
Although the aforementioned technique is extremely efficient on providing very fast computation of shortest-path distances on main memory, there are very few works that try to replicate those algorithms on secondary storage. HLDB [DW12] stores the calculated hub labels for continental road networks on a commercial database system and translated the typical HL distance query between two vertices to plain SQL commands. Moreover, it showed how to efficiently answer kNN queries and *k-best via points*, again with SQL queries. Recently, HopDB [JFWX14] proposed a customized solution that utilizes secondary storage, also during preprocessing. Unfortunately, both methods are not flawless: HLDB has only been tested on road networks where the size of labels is relatively small (<100) and therefore its speed would seriously degrade for large-scale graphs due to the much larger size of the labels, whereas HopDB answers only vertex-to-vertex queries and is a customized C++ solution that cannot plug-in on existing database systems and hence has limited practical applicability.

Considering the drawbacks of previous methods, this work presents a database framework that may service multiple distance queries on massive large-scale graphs. Our pure-SQL *COLD* framework (COmpressed Labels on the Database) may answer multiple distance queries (point-to-point, kNN) in addition to RkNN and *one-to-many* queries not handled by previous methods, rendering it a complete database solution for a variety of practical applications on massive, large-scale graphs. Our extensive experimentation will show that COLD outperforms previous solutions, including specialized graph databases, on all aspects (including query performance and memory requirements) while servicing a larger variety of distance queries, making it the best overall solution for servicing scalable, real-time applications operating on large-scale graphs. In addition, COLD is implemented entirely on a popular, open-source database engine with no third-party extensions and thus our results are easily reproducible by anyone.

The outline of this work is as follows. Section 5.2 describes our novel COLD framework and its implementation details. Experiments establishing the benefits of our approach are provided in Section 5.3.

5.2 Contribution

In this section we will present our *COLD* (COmpressed Labels on the Database) database framework that may answer multiple distance queries (point-to-point, kNN, RkNN and *one-to-many*) for large-scale graphs using SQL commands. Since, the COLD framework builds on the previous works of HLDB



Vertex	Hub Labels (h,d)
0	(0,0)
1	(0,1), (1,0)
2	(0,1), (2,0)
3	(0,1), (3,0)
4	**(0,1), (4,0)**
5	(0,2), (1,1), (5,0)
6	(0,2), (1,1), (6,0)
7	(0,2), (1,1), (7,0)
8	(0,2), (2,1), (8,0)
9	(0,2), (3,1), (9,0)
10	**(0,2), (4,1), (10,0)**
11	(0,3), (1,2), (5,1), (11,0)
12	**(0,3), (1,2), (6,1), (12,0)**
13	(0,3), (1,2), (7,1), (13,0)

Figure 5.1 & Table 5.1: A sample Graph G and the created hub-labels

[ADF$^+$12] and [EP14b] we will follow the notation and running example presented there, for highlighting the necessary concepts and challenges for adapting those previous works, (i) in the context of large-scale graphs for [ADF$^+$12] and (ii) within the boundaries of a relational database management system (RDBMS) for [EP14b]. To this end, we chose PostgreSQL [Pos15] for our implementation, since it is a very popular, open-source, extensible RDBMS. Although we will use some PostgreSQL-specific data-types and SQL extensions, we do not use any third-party extensions but only features included in its standard installation.

5.2.1 Implementation

Our COLD framework assumes that we have a correct hub labeling (HL) framework that generates hub-labels for the undirected, unweighted graphs we wish to query. Although COLD will work with any correct HL algorithm, in this work we use the [AIY15] implementation of the PLL algorithm of [AIY13] to generate the necessary labels. To highlight the results of this process, the labels for the undirected, unweighted graph G of Figure 5.1 are shown in Table 5.1. Throughout this work, we will refer to those labels as the *forward labels*. The forward label $L(v)$ for a vertex v is an array of pairs $(u, dist(v, u)$ sorted by hub u. Since our work also focuses on snapshot kNN and RkNN queries, there also some target objects $P \in V$ that do not change over time. For our specific running example we assume that $P = \{4, 10, 12\}$

Table 5.2: The *forward* table used in HLDB for the sample graph G

v	hub	dist
...	...	...
2	0	1
2	2	0
...	...	...
7	0	2
7	1	1
7	7	0
...	...	...

Table 5.3: The *forwcold* table used for COLD for the sample graph G

v	hubs	dists
...	...	...
2	$\{0, 2\}$	$\{1, 0\}$
...	...	...
7	$\{0, 1, 7\}$	$\{2, 1, 0\}$
...	...	...

Code 5.1: V2v query for HLDB

```
SELECT MIN(n1.dist+n2.dist)
FROM forward n1, forward n2
WHERE n1.v = s
AND n2.v = t
AND n1.hub = n2.hub;
```

Code 5.2: V2v query for COLD

```
SELECT MIN(n1.dist+n2.dist) FROM
/* Expand the hubs, dists arrays */
 (SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS dist
FROM forwcold WHERE v = s) n1,
 (SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS dist
FROM forwcold WHERE v = t) n2
WHERE n1.hub=n2.hub;
```

and thus, we highlight the respective entries of Table 5.1.

5.2.1.1 Vertex-to-Vertex (v2v) queries.

To find the network distance $dist(s, t)$ between two vertices s and t, a HL query must find the hub $v \in L(s) \cap L(t)$ that minimizes the sum $dist(s, v) + dist(v, t)$. For our sample graph G, e.g., the minimum distance between vertices 2 and 7 is $d(2, 7) = 3$, using the hub 0. To translate this HL query into SQL commands, in HLDB [ADF$^+$12] forward labels are stored in a database table denoted *forward* where the labels of vertex v are stored as triples of the form $(v, hub, dist(v, hub))$ (see Table 5.2). The table *forward* has the combination of (v, hub) as the primary key and is clustered according to those columns, so that "*all rows corresponding to the same label are stored together to minimize random accesses to the database*" [ADF$^+$12]. Then we can find the distances between any two vertices s and t by the SQL query of Code 5.1.

Although the HLDB vertex-to-vertex (v2v) query is very simple, there is one major drawback. For such a query, HLDB has to fetch from secondary storage a total of $|L(s)| + |L(t)|$ rows. Although this is practical for

Table 5.4: Necessary data structures for the sample graph G, $P = \{4, 10, 12\}$ and *one-to-many*, kNN and RkNN queries

Hub	**Backw. Lbls (to-many)**	**kNN Backw. Labels (k=2)**	**RkNN Backw. Labels (k=1)**	**Obj.**	**kNN Result (k=1) (Obj., dist)**
0	(4,1), (10,2), (12,3)	(4,1), (10,2)	(4,1), (12,3)	**4**	(10,**1**)
1	(12,2)	(12,2)	(12,2)		
4	(4,0), (10,1)	(4,0),(10,1)	(4,0), (10,1)	**10**	(4,**1**)
6	(12,1)	(12,1)	(12,1)		
10	(10,0)	(10,0)	(10,0)	**12**	(4,**4**)
12	(12,0)	(12,0)	(12,0)		

road networks where the forward labels have less than 100 hubs per vertex [ADGW12], it cannot scale for large-scale graphs where the forward labels have thousand of hubs per vertex. Moreover, on such graphs the *forward* DB table and the corresponding primary key index will become too large, which is also an important disadvantage. To this end, we take advantage of the fact that PostgreSQL has an array data type that allows columns of a DB table to be defined as variable-length arrays. Hence, in COLD we store hubs and distances for a vertex (both ordered by hub) as arrays in two separate columns (i.e., hubs and dists) in a single row. The resulting *forwcold* compressed DB table is shown in Table 5.3. This approach not only emulates exactly how labels are stored on main-memory for fast v2v queries but also has considerable advantages: (i) The *forwcold* DB table has exactly $|V|$ rows (ii) The *forwcold* DB table has the column v as primary key without needing a composite key. This alone facilitates faster queries. Moreover the size of the corresponding index will be much smaller. In fact, our experimentation will show that the primary-key index for *forwcold* may be $> 4,400\times$ smaller than the index size of HLDB. (iii) For a v2v query, COLD needs to access exactly two rows, regardless of the sizes of $|L(s)|$ and $|L(t)|$. This way, we efficiently minimized the secondary-storage utilization, even working inside a database. The resulting SQL query for COLD is shown in Code 5.2. There we exploit the fact that PostgreSQL guarantees that parallel unnesting for hubs and distances for each nested query will be in sync, i.e., each pair (hub, dist) is expanded correctly since for the same v the respective arrays have the same number of elements[1].

5.2.1.2 Additional queries overview.

For answering more complex (kNN, RkNN and *one-to-many*) distance queries on a HL framework for a set of target objects P, we need to build some additional data structures from the forward labels (for undirected graphs).

[1]http://stackoverflow.com/questions/23830991/parallel-unnest-and-sort-order-in-postgresql

Then to answer the respective query we only need to combine the forward labels $L(q)$ of query vertex q, with the respective data structure explained in the following. Those data structures are summarized in Table 5.4.

For answering *one-to-many* queries, i.e., calculate distances between a source vertex q and all objects in P, we need to build the *backward labels-to-many* by basically ordering the forward labels of the target objects by hub [DGW11] and then by distance for the same hub. For kNN queries we only need to keep at most the k-best pairs (of smallest distances) per hub from the backward labels-to-many to create the *kNN backward labels* [ADF+12]. In our specific example, the kNN backward labels for $k = 2$ and hub 0, do not contain the pair $(12, 3)$. Finally, for RkNN queries, we must first calculate the *kNN Results* (i.e., the NN of the object 4 is the object 10 with distance 1) and then we build the RkNN backward labels, based on the observation that *"we need to access those pairs from the backward labels-to-many to a specific object, if and only if those distances are equal or smaller than the distance of the kNN of this object"* [EP14b]. In our specific example, the RkNN backward labels for $k = 1$ and hub 0, do not contain the pair (10,2) since the NN of object 10 (the object 4) is within distance 1. Although for our small graph the differences between the individual data structures seem minimal, for larger graphs those differences become very prominent. This was also showcased by the theoretical analysis provided in [EP14b] which showed that backward labels-to-many will have on average $D \cdot |HL|$ pairs, the kNN backward labels have at most $k \cdot |V|$ pairs and the RkNN backward labels have on average $\varepsilon \cdot D \cdot |HL|$ pairs where ε may be < 0.01 for specific datasets and experimental settings. Moreover, Efentakis et al. [EP14b] have shown how these additional data structures may be constructed from the forward labels in main-memory, requiring less than few seconds, even for the larger tested datasets.

5.2.1.3 kNN queries.

To translate the HL kNN query into SQL, HLDB stores kNN backward labels in a separate DB table denoted *knntab* that stores triples of the form $(hub, dist, obj)$ (see Table 5.5). The respective table *knntab* has the combination of $(hub, dist, obj)$ as a composite primary key and is clustered according to those columns. Note that in HLDB, we cannot use the combination of $(hub, dist)$ as a primary key, because especially in large scale graphs we will have a lot of distance ties even for k-entries for the same hub. Then we can can answer a kNN query from vertex q by the SQL query of Code 5.3. Again, the kNN HLDB query has the same drawbacks as before, i.e., it has to retrieve $|L(q)|$ rows from *forward* and $k \cdot |L(q)|$ rows from *knntab* tables, for a total of $(k + 1) \cdot |L(q)|$ rows retrieved from secondary storage. Moreover in a

Table 5.5: The *knntab* table used in HLDB for the sample graph G, $k = 2$ and $P = \{4, 10, 12\}$

hub	**dist**	**obj**
0	1	4
0	2	10
1	2	12
...	...	...

Table 5.6: The *knntab* table used in COLD for the sample graph G, $k = 2$ and $P = \{4, 10, 12\}$

hub	**dist**	**objs**
0	1	{4}
0	2	{10}
1	2	{12}
...	...	...

Code 5.3: *k*NN query for HLDB

```
SELECT MIN(n1.dist+n2.dist),
n2.obj FROM
forward n1, knntab n2
WHERE n1.v = q
AND n1.hub = n2.hub
GROUP BY n2.obj
ORDER BY MIN(n1.dist+n2.dist)
LIMIT k;
```

Code 5.4: *k*NN query for COLD

```
SELECT MIN(n1.dist+n2.dist),
UNNEST(objs) AS obj FROM
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS dist
FROM forwcold WHERE v = q) n1,
/* Get k-entries per hub,dist */
(SELECT hub, dist ,objs[1:k]
FROM knncold) n2
WHERE n1.hub=n2.hub
GROUP BY obj
ORDER BY MIN(n1.dist+n2.dist)
LIMIT k;
```

database, it makes sense to create one large *knntab* table for the maximum value *kmax* of k (e.g., for $k = 16$) that may be serviced by the DB framework and that same table will be used for all *k*NN queries up to $k = kmax$. In that case, the HLDB framework will have to retrieve $(kmax + 1) \cdot |L(q)|$ rows for every *k*NN query regardless of the value of k.

To remedy the HLDB drawbacks, COLD creates the *knncold* DB table (Table 5.6) that has the columns $(hub, dist, objs)$, whereas objects are grouped and ordered per hub and distance (the column *objs* is an array). Although for our sample graph G, the DB tables *knntab* and *knncold* seem identical, COLD's method offers several advantages: (i) We can now use the combination of $(hub, dist)$ as a primary key, which makes the respective index significantly smaller and faster and (ii) In case of many distance ties (common to large-scale graphs) and one large *knncold* DB table that services all *k*NN queries for values of k up to the maximum value *kmax* , we only need to fetch the first *k-objs* entries (i.e., `objs[1:k]`) per hub and dist, which makes the later sorting faster (see Code 5.4).

Code 5.5: *One-to-many query* for COLD

```
SELECT MIN(n1.dist+n2.dist),
UNNEST(objs) AS obj FROM
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS dist
FROM forwcold WHERE v = q) n1,
objcold n2
WHERE n1.hub=n2.hub
GROUP BY obj;
```

Code 5.6: RkNN query for COLD

```
SELECT n3.id2,n3.dist FROM
/* Nested n3 query is a modified
 one-many-query to revcold */
(SELECT MIN(n1.dist+n2.dist) AS dist,
UNNEST(objs) AS obj FROM
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS dist
FROM forwcold WHERE v = q) n1,
revcold n2
WHERE n1.hub=n2.hub
GROUP BY obj
ORDER BY obj,MIN(n1.dist+n2.dist)
) n3,
/* Join with the knnres table */
(SELECT obj, dists[k] AS dist
FROM knnres) n4
WHERE n3.obj=n4.obj
AND n3.dist<=n4.dist
ORDER BY n3.obj;
```

5.2.1.4 One-to-many queries.

Similar to how COLD handles kNN queries, for one-to-many queries, COLD stores the *backward labels-to-many* in a new *objcold* DB table that has an identical format to *knncold*, i.e., it has three columns (hub, $dist$, $objs$) whereas objects are grouped and ordered per hub and distance. *Objcold* also uses the combination of (hub, $dist$) as a primary key. The resulting *one-to-many* query (Code 5.5) is quite similar to COLD's kNN query, but (i) it operates on the larger *objcold* DB table (ii) It does not have the `ORDER BY ... LIMIT k` clause and (iii) We use the entire *objs* array per hub and distance instead of `objs[1:k]`. Note that HLDB cannot possibly support such queries because it will need to retrieve on average $|L(q)|$ rows from the *forward* table and a total of $|L(q)| \cdot D \cdot (|HL|/|V|)$ [EP14b] rows from the corresponding *objlab* table, which will be prohibitively slow for very large datasets.

5.2.1.5 RkNN queries.

Table 5.7: The *knnres* table used in COLD for RkNN queries, the sample graph G, $k = 1$ and $P = \{4, 10, 12\}$

obj	**dists**	**objs**
4	{1}	{10}
10	{1}	{4}
12	{4}	{4}

For RkNN queries, COLD stores the RkNN backward labels in a separate *revcold* DB table that has an identical format to previous *knncold* and *objcold* DB tables, i.e., three columns $(hub, dist, objs)$ where objects are grouped and ordered per hub and distance and the combination of $(hub, dist)$ used as a primary key. COLD also stores the *kNN Results*, i.e., the kNN of all objects in another *knnres* DB table that has the format $(obj, dists, objs,)$ where obj is the primary key and *objs* and *dists* are arrays (both ordered by distance). Therefore the kNN of object p is the `objs[k]` within distance `dists[k]` of the respective row for p. Again it makes sense to build a *knnres* DB table for a max value of *kmax* that may service RkNN queries for varying values of k. As a result, during the RkNN COLD query, we will have to use an additional `JOIN` between the *revcold* and *knnres* DB tables. The resulting query is shown in Code 5.6.

We see that even the more complex RkNN query in COLD requires just a few lines of SQL code that will work on any recent PostgreSQL version without any need of third-party extensions or specialized index structures. In fact, all DB tables in COLD, use only standard B-tree based primary key indexes, without any additional modifications. To satisfy this strict requirement, we had to effectively compress the index sizes by grouping rows per vertex (*forcold* table) or object (*knnres* table), or by hub and distance for *knncold*, *objcold* and *rknncold*. And although we used PostgreSQL specific SQL extensions for expanding the stored arrays, latest versions of other commercial databases (e.g., Oracle, SQL Server) support similar array data-types. Hence, it would be quite easy to port COLD to other database vendors, as well.

In a nutshell, in this section we covered all the implementation details of our novel, pure-SQL COLD framework that may answer multiple distance queries (v2v, kNN, RkNN and one-to-many) entirely within a database. We also presented the actual queries used and the way the necessary data structures are stored within the database, so that our results are easily reproducible by anyone. Although our focus was on query efficiency, it is important to note that once we create the *forcold* table, all the adjoining DB tables within COLD can also be created by SQL commands. The resulting queries were omitted due to space restrictions. This fact also testifies that COLD is really a pure SQL framework for servicing multiple distance queries on large-scale graphs. We also provided the necessary theoretical details why the COLD framework outperforms previous solutions. This will be further quantified in the following experimental section.

Table 5.8: Networks graphs statistics

Graph	| V |	| E |	Avg degr.	| HL | / | V |	PLL Preproc. Time (s)
Facebook	4,039	88,234	22	26	0.03
NotreDame	325,729	1,090,108	3	55	6
Gowalla	196,591	950,327	5	100	13
Youtube	1,134,890	2,987,624	3	167	123
Slashdot1	77,360	469,180	6	204	11
Slashdot2	82,168	504,230	6	216	13
Citeseer1	268,495	1,156,647	4	408	110
Amazon	334,863	925,872	3	689	230
DBLP	540,486	15,245,729	28	3,628	5,720
Citeseer2	434,102	16,036,720	37	4,457	5,946

5.3 Experimental Evaluation

To evaluate the performance of COLD on various large-scale graphs, we conducted experiments on a workstation with a 4-core Intel i7-4771 processor clocked at 3.5GHz and 32 GB of RAM, running Ubuntu 14.04. We compare our COLD framework with a custom implementation of HLDB in PostgreSQL and with $Neo4j$, a well-known, popular graph database.

We use the same network graphs as our previous work of [EP14b] that are taken from the Stanford Large Network Dataset Collection [LK14] and the 10th Dimacs Implementation Challenge website [BMSW13]. All graphs are undirected, unweighted and strongly connected. We used collaboration graphs (DBLP, Citeseer1, Citeseer2) [GSS08], social networks (Facebook [ML12], Slashdot1 and Slashdot2 [LLDM09]), networks with ground-truth communities (Amazon, Youtube) [YL12], web graphs (Notre Dame) [AJB99] and location-based social networks (Gowalla) [CML11]. The graphs' average degree is between 3 and 37 and the PLL algorithm creates $26 - 4,457$ labels per vertex, requiring $0.03 - 5,950s$ for the hub labels' construction (see Table 5.8).

COLD and HLDB were implemented in PostgeSQL 9.3.6, 64bit with reasonable settings (8192 MB *shared buffers*, 64 MB *temp buffers*). We also used Neo4j Server v2.1.5. The Neo4j queries were formulated using *Cypher*, Neo4j's declarative query language and we report query times as they were returned by the server. Although Cypher may theoretically facilitate *one-to-many* queries (besides vertex-to-vertex), testing Neo4j with our datasets and the same number of target vertices we tested COLD with, resulted in a "java.lang.Stack OverflowError". Providing the server with additional resources[2] had no positive effect and thus there are no results to report for *one-to-many* queries and Neo4j.

We conducted experiments belonging to 4 query types: (i) *vertex-to-*

[2]http://neo4j.com/developer/guide-performance-tuning/

vertex, (ii) kNN , (iii) RkNN and (iv) *one-to-many*. For each experiment, we used 10,000 random start vertices, reporting the average running time. Before each experiment, we restart the PostgreSQL and Neo4j servers for clearing their internal cache and we also clear the operating system's cache for accurate benchmarking. All charts are plotted in logarithmic scale.

5.3.1 Performance on HDD

In our first round of experiments, we ran experiments on an HDD. We used a Seagate Barracude ST3000DM001 SATA3 7200*rpm* with 64Mb cache.

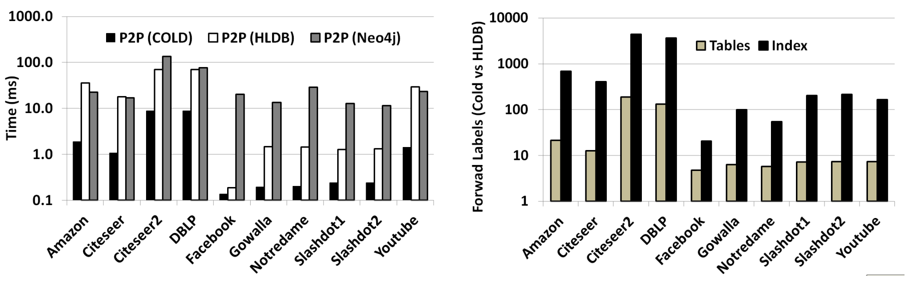
5.3.1.1 Vertex-to-vertex.

Figure 5.2a shows results for vertex-to-vertex (v2v) queries for COLD, HLDB and Neo4j. Results show that COLD is consistently 5 - 20.7× faster than HLDB, with this difference amplified for the Citeseer1, Amazon and Youtube datasets (16.8, 19.1 and 20.7 respectively). Moreover, COLD is also 9 - 149× (for the $facebook$ dataset) faster than Neo4j, which exhibits stable performance for all datasets, but is slower from both COLD and HLDB. For all datasets, COLD requires less than $9ms$ for answering v2v queries.

Figure 5.2b shows the difference in memory size for the DB tables *forcold* (COLD) and *forward* (HLDB) and their respective primary-key (PK) indexes. Results show that the size of the PK index in COLD is $3,600$ - $4,444$× smaller than for HLDB (for DBLP and Citeseer2 respectively). As expected, the difference in index sizes is almost identical to the $|HL|/|V|$ ratio, since *forcold* table has $|V|$ rows and *forward* has $|HL|$ rows. Likewise, the corresponding tables are 131 - 188× smaller for COLD. Thus, the techniques used for compressing the forward labels in COLD clearly achieve a considerable reduction in memory size, rendering our proposed framework suitable for real-world scenarios.

5.3.1.2 kNN.

Figure 5.3a shows the speedup of COLD compared to HLDB in the case of kNN queries for $D = 0.01$ and $k = \{1, 2, 4, 8, 16\}$. As described in Section 5.2.1.3, we have created two DB tables for each framework (COLD, HLDB), one for $kmax =$ 4 and one for $kmax = 16$. Then the DB table for $kmax = 4$ is used for answering kNN queries for $k = 1$, $k = 2$ and $k = 4$ and the kNN table for $kmax = 16$ is used for answering kNN queries for $k = 8$ and $k = 16$. Results show that for $k = 1$, COLD is 5 - 19× faster for the five largest datasets (Amazon, Citeseer,Citeseer2, DBLP. Youtube) and although this speedup degrades for larger values of k, COLD remains



(a) Vertex-to-vertex query times (b) Memory size's difference between COLD and HLDB

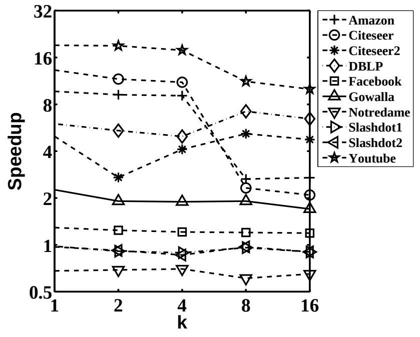
Figure 5.2: Experiments on HDD for *vertex-to-vertex*

consistently 2 - 10× faster even for $k = 16$. For the smaller datasets, performance between COLD and HLDB is quite similar, with COLD performing better on Facebook and Gowalla, while HLDB performs only marginally better for Slashdot1, Slashdot2 and Notredame. In all cases, COLD answers kNN queries for all datasets in less than $26ms$ even for $k = 16$.

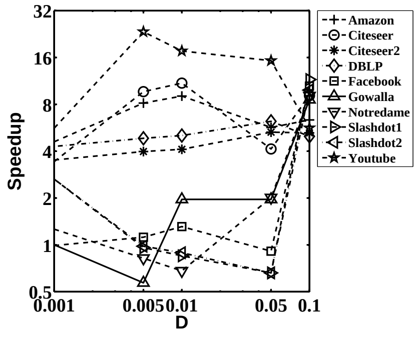
In our second set of kNN experiments, we assess the performance of COLD vs HLDB for varying values of D. For each value for D, we have build separate versions of *knntab* (HLDB) and *knncold* (COLD) DB tables for $D \cdot |V|$ objects selected at random from each dataset and $kmax = 4$. Figure 5.3b shows results for $k = 4$ and $D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$. Again, for the five largest datasets COLD is consistently 3.5 - 23.4× faster than HLDB, whereas even for the smaller datasets, COLD is consistently 8.6 - 11.5 faster than HLDB for the largest value of D (for $D = 0.1$). Moreover, COLD may answer kNN queries for $k = 4$ on all datasets and all values of D in less than $15ms$.

5.3.1.3 RkNN.

For RkNN experiments, we only report the COLD's performance, since there is no other SQL framework that supports these queries. In out first experiment, we report the performance of COLD for $D = 0.01$ and $k = \{1, 2, 4, 8, 16\}$. For all those queries we have built one version of the *knnres* DB table for $kmax = 16$ (see Section 5.2.1.5) and 3 separate *revcold* tables for $kmax = \{1, 4, 16\}$. As expected, for RkNN queries and $k = 1$ we use the *revcold* table built for $kmax = 1$, for $k = 2$, $k = 4$ we use the *revcold* table built for $kmax = 4$ and for $k = 8$, $k = 16$ we use the *revcold* table built for $kmax = 16$. Figure 5.4a presents the results. In all cases, COLD provides excellent query times that are below $20ms$ for $k = 1$ in all datasets and never exceed $85ms$ even for $k = 16$.

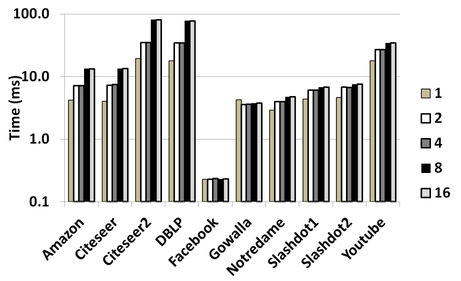


(a) kNN Speedup of COLD vs HLDB for $D = 0.01$ and varying values of k

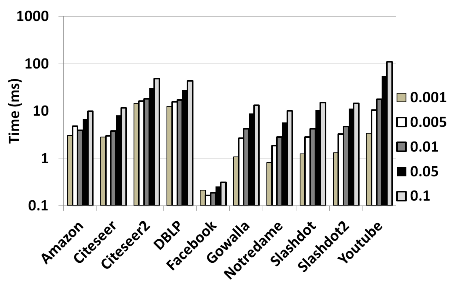


(b) Speedup of COLD vs HLDB for $k = 4$ and varying values of D

Figure 5.3: kNN Experiments on HDD for COLD and HLDB



(a) COLD RkNN query times for $D = 0.01$ and varying values of k



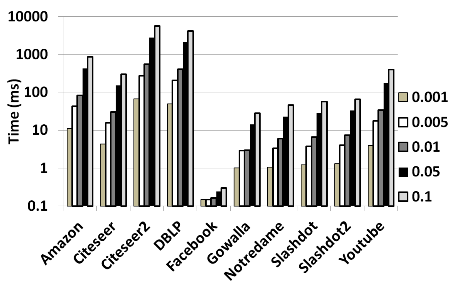
(b) COLD RkNN query times for for $k = 1$ and varying values of D

Figure 5.4: RkNN Experiments on HDD for COLD

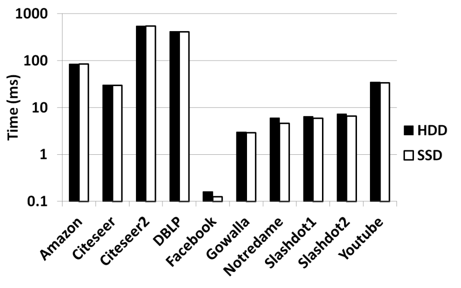
In our second set of RkNN experiments, we assess the performance of COLD for varying values of D. Figure 5.4b presents results for $k = 1$ (as this is the typical case for RkNN queries) and $D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$. Results show that although COLD's performance degrades for larger values of D, RkNN query times are below $65ms$ for all datasets and values of D, with the exception of Youtube and $D = 0.1$ ($109.3ms$). Thus, COLD offers excellent and stable performance in RkNN queries for all all datasets and tested values of k and D.

5.3.1.4 *One-to-Many.*

Again, COLD is the only SQL framework that supports *one-to-many queries*. Figure 5.5a presents the corresponding results for varying values of D ($D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$). COLD answers such queries in less than a sec-



(a) One-to-Many experiments for COLD varying values of D



(b) COLD One-to-Many HDD vs SSD

Figure 5.5: One-to-many experiments for COLD

ond for all datasets and values of D, except the Citeseer2 and DBLP datasets (those with the highest $|HL|/|V|$ ratio) that require $5601ms$ and $4170ms$ respectively, for $D = 0.1$. For such high values of D, the *one-to-many* query reaches the complexity of an *one-to-all* query and as expected, it cannot be any faster on a secondary storage device. Note that even specialized graph databases like Neo4j cannot support this type of queries for more than a 1,000 target objects, whereas COLD answers *one-to-many queries* to $110,000$ target objects in the Youtube dataset in $401ms$, with a simple SQL query. This alone, is a great achievement.

5.3.2 Performance on SSD

After establishing the performance characteristic of COLD in the HDD, in our second round of experiments, we repeat some of the previous experiments on an SSD to measure the impact of the secondary-storage device type to results. The SSD used is a Crucial CT512MX100SSD1 MX100 512GB 2.5" SATA3.

5.3.2.1 Vertex-to-vertex performance.

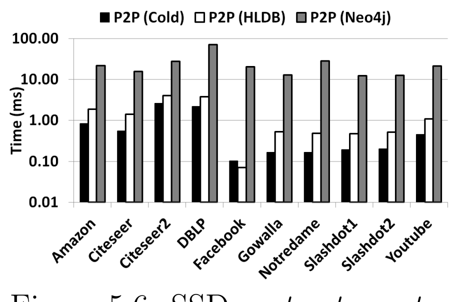
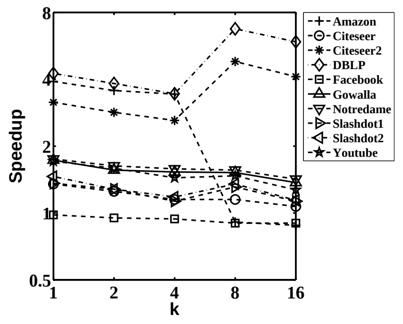
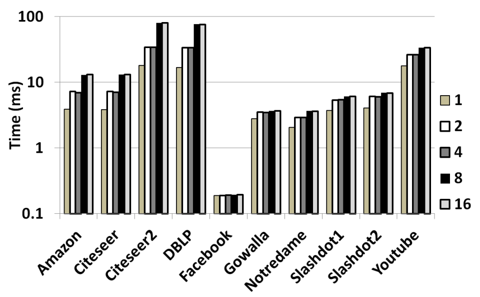


Figure 5.6: SSD *vertex-to-vertex*

Although the usage of SSD favors HLDB more than COLD (see Figure 5.6), COLD is consistently 1.5 - 3.2× faster than HLDB (except Facebook, which is the smallest of datasets). The SSD has almost no impact on Neo4j and thus, COLD is now 11-199× faster than *Neo4j* on all datasets. Note, than on the



(a) kNN Speedup of COLD vs HLDB for $D = 0.01$ and varying values of k



(b) COLD RkNN query times for $D = 0.01$ and varying values of k

Figure 5.7: kNN and RkNN SSD performance

SSD, COLD requires less than $0.9ms$ for all datasets and vertex-to-vertex queries, except the Citeseer2 and DBLP datasets (those with the highest $|HL|/|V|$ ratio). But even then, vertex-to-vertex queries take less than $2.6ms$ for COLD.

5.3.2.2 kNN performance.

Figure 5.7a shows the performance speedup of COLD compared to HLDB in the case of kNN queries running on the SSD, for $D = 0.01$ and varying value of k. Again, although the SSD lowers the performance gap between COLD and HLDB, COLD is still faster on all datasets (except Facebook). In fact, COLD is 3 - 6.75× faster than HLDB for the high $|HL|/|V|$ ratio datasets (Citeseer2, HLDB) requiring less than $24.6ms$ even for $k = 16$.

5.3.2.3 RkNN.

Figure 5.7b presents the results of the RkNN query time performance on COLD for $D = 0.01$ and varying value of k. Results show that SSD usage accelerates COLD by only 20% at most, which clearly demonstrates that COLD effectively minimized secondary storage utilization and thus adding a better secondary storage provides minimal benefits for such queries.

5.3.2.4 One-to-Many performance.

Finally, Figure 5.5b compares *one-to-many* queries on HDD and SSD for COLD. Again, the SSD usage accelerates COLD by only 2-30% with further testifies the optimal secondary storage utilization of COLD.

5.3.3 Summary

Our experimentation has shown that our proposed COLD framework outperforms previous state-of-the-art HLDB in all performance benchmarks, including query performance, memory size and scalability. On the HDD, COLD is $9 - 149\times$ faster for *vertex-to-vertex* queries and $10 - 20\times$ faster for kNN queries and the largest datasets. On the SSD, COLD is $2 - 3\times$ faster than HLDB for *vertex-to-vertex* and up to $6.75\times$ faster for kNN queries. COLD also requires up to $4,444\times$ less storage space (indexes) and up to $188\times$ less storage space (DB tables) used for storing forward labels and thus COLD is suitable for real-world deployment. Even specialized graph databases like Neo4j are outperformed by COLD, which is up to $199\times$ faster. But, most importantly COLD may service additional (RkNN, one-to-many) queries, not handled by any other previous secondary-storage solutions, while providing excellent query times even on standard hard drives. This fact demonstrates that COLD achieves maximum secondary-storage utilization that it will be very hard to beat by any forthcoming pure-SQL framework.

Chapter 6

TwitterViz: Visualizing and Exploring the Twittersphere

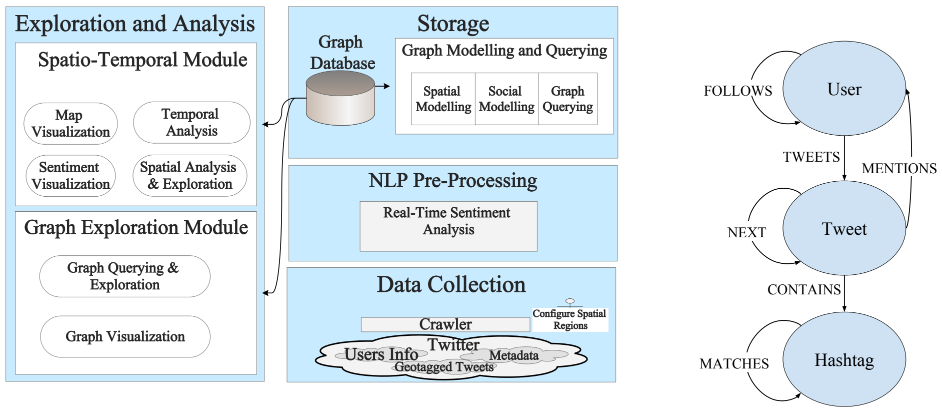
6.1 Introduction

Micro-blogging platforms, especially Twitter, have become a very popular communication tool, where millions of users share opinions on different aspects of everyday life. Twitter reports over 100 million active users and 500 million tweets exchanged every day. This huge amount of data and the fact that they are offered publicly in real time make their management and analysis challenging.

Many research studies have been conducted in order to determine whether Twitter can actually give insights as to how people behave. Such studies have focused on analyzing a variety of spatio-temporal phenomena (e.g. [KP12, SGAF13]), as well as topics, sentiments and social interactions (e.g. [QECC12, QCC12]). Typically, these works focus on specific problems and examine specific parts of the Twittersphere.

A recent survey of approaches for Twitter analytics can be found in [GSZS14]. It identifies the need for integrated solutions that provide a unified framework to be used by researchers and practitioners across disciplines, and it suggests the support of the following components for this purpose: (a) a focused crawler to allow for configuration by the user, (b) a pre-processor for the processing of tweets based on specific needs, (c) a defined data model that allows the efficient execution of complex queries, (d) the support of a query language and (e) the informative spatial as well as graph visualization.

TwitterViz supports all of these components. The crawler allows spatial configuration in order to focus on specific geographic areas. The pre-processor uses natural language processing (NLP) in real time and the pro-



(a) TwitterViz architecture (b) TwitterViz Data Model

Figure 6.1: TwitterViz Architecture and Data Model for querying the Twittersphere

cessed information is stored in a graph database with a defined data model as well as the support of a powerful graph query language. In addition, the visualization of the data both based on spatio-temporal characteristics as well as graph characteristics renders *TwitterViz* a complete solution for the management of Twitter data in order to provide useful analytics. *TwitterViz* provides a framework to the non-expert user for exploring and analyzing the Twittersphere using simple, unobtrusive yet powerful tools.

6.2 The TwitterViz System

TwitterViz comprises a modular pipeline that supports data collection, storage, analysis and visualization of Twitter data. Figure 6.1a shows the architecture of the system. We briefly describe each module below.

Data Collection. The *data collection* module supports the crawling of tweets. Tweets are collected from specific geographic regions based on users' preferences along with information about "followers" relationships. The collection of these data is subject to limitations, but as long as the geographic regions selected by the user are on the country level, the restrictions are usually not violated. In case of violations there are mechanisms that allow the smooth recovery of the system.

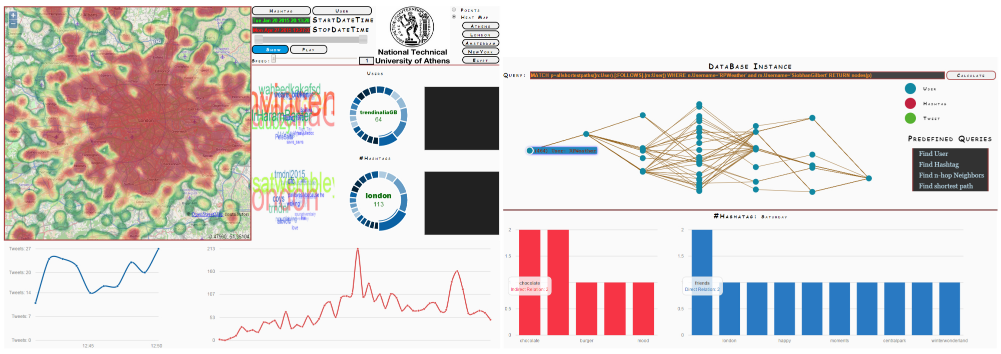
Sentiment Analysis. NLP for sentiment analysis is conducted and each tweet is then tagged with a score denoting its sentiment. The pre-processing

uses the *AlchemyAPI*[1] tool which is also used in various research works (e.g. [QCC12]).
Storage. The *storage module* consists of a *Neo4j*[2] graph database, which naturally fits the overtly relationship-centered domain of social networks. The defined data model, depicted in Figure 6.1b, enables the Twitter graph construction using a variety of relationships, rendering it a complete model for the support of complex queries.

The *Storage Example* describes how the information is stored in the database once a tweet arrives. *Storage Example*: Once a tweet arrives, the `USER` who has created the `TWEET` is added to the database and appropriate `FOLLOWS` relationships are added/updated. The temporal aspect is maintained by the `NEXT` relationship between two tweets by the same user. The `HASHTAGS` that are `CONTAINED` in the `TWEET` also have a special relationship called `MATCHES` with a property to count the frequency of the co-occurrence of `HASHTAGS` that appear in the same `TWEET`. By using this data model and the specific graph database, we can achieve the physical representation of this model in the database, since *Neo4j* offers *native graph storage and processing.* In this way, graph traversal queries can be achieved very fast. In addition, *TwitterViz* employs the *Neo4J Spatial* extension, that leverages an R-tree [Gut84b] for the indexing of the tweets based on their coordinates. In this way, we can perform spatial queries on tweets in an efficient manner.
Visualization and Analysis. The user interface of the *TwitterViz* framework, shown in Figure 6.2, consists of two main views: (a) the *Spatio-Temporal Analysis View* and (b) the *Graph Analysis View.* Both can be used simultaneously, independently or in combination with each other. In future versions, we intend to provide additionally an API that would allow third-party applications to reuse the results of the analysis creating custom visualizations for specific needs.
Spatio-Temporal Analysis. The user is given a variety of tools for spatio-temporal analysis and exploration of Twitter data, both on the map as well as a variety of charts. Spatial indexing is used to speedup range queries focusing on tweets in specific areas. The user interface allows for temporal visualization and analysis of tweets, as well as the simulation of the temporal evolution of tweets created during a specific time window. All tools can be used in combination with each other in order to reach to useful conclusions. Figure 6.2a shows the *spatio-temporal analysis* interface, which supports the following operations:

- Range queries on the map to visualize tweets from specific areas.

[1]http://www.alchemyapi.com
[2]http://www.neo4j.org/



(a) Spatio-Temporal Analysis Interface (b) Graph Mining Interface

Figure 6.2: *TwitterViz* User Interface

- Visualization of sentiment on tweets on the map in speficic geographic areas using a defined visual syntax. The user can investigate how the sentiments change in specific areas as well as how they change in time, by also applying other restrictions based on the social network's structure.

- Visualization of a user's followers' tweets on the map, combining information from the graph.

- Visualization and study of the temporal evolution of tweets in user-defined time windows.

- Analysis of the spatio-temporal distribution of tweets.

- Presentation of a variety of real-time statistics on the streaming data.

All of the supported operations can be combined with each other using restrictions, allowing a powerful spatio-temporal and social analysis based on the advanced visualization offered by *TwitterViz*.
Graph Mining. The *graph mining* module offers to the non-expert user off-the-shelf advanced queries for the exploration of the Twitter graph. In addition, the expert can formulate her own *Cypher* queries[3]. The visualization of the graph allows for exploration of query results. In addition, charts are used to visualize specific relationships on the graphs such as *hashtags*. The user is thus given powerful tools to explore the Twittersphere, visualize and analyze data as well as extract meaningful information. Figure 6.2b shows the *graph mining* interface that supports the following operations:

[3] *Cypher* is the graph query language used in *Neo4j*

- Defined *Cypher* queries for the non-expert for graph exploration, like *shortest-path* queries, *n-Hop traversal* queries and queries for locating specific nodes.

- Support for custom queries on the graph for more complex analysis, such as pattern matching queries on the graph. Custom queries can be formulated easily and can combine the tweets' geo-social characteristics. Such example queries could be:

  - Find the tweets with hashtag "parthenon" that are within $0.5Km$ from the Athens historical center (substituting *x*,*y* with actual coordinates).

    ```
    START n=node:tweetWKT('withinDistance:[x,y, 0.5]')
    MATCH (n)-[]-(h:HashTag)
    WHERE h.Hashtag='parthenon'
    RETURN n,h
    ```

  - Find the *top-10* users in New York based on how many other users "follow" them.

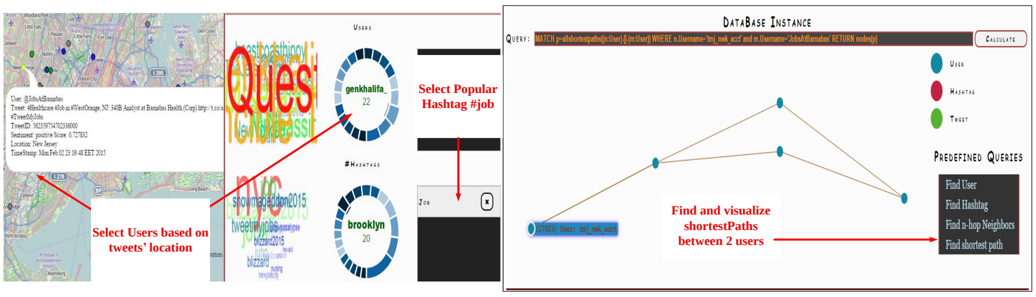
    ```
    MATCH (n:User)<-[:FOLLOWS]-(m:User), (n)-[]->(t:Tweet)
    WHERE t.Region='NewYork' WITH n,count(m) AS total
    RETURN n ORDER BY total DESC limit 10
    ```

- Visualization of the results for all of the queries on the graph.

- Presentation of a variety of statistics for real-time graph analysis.

6.3 Demonstration

A use-case scenario that benefits from the use of *TwitterViz* is the following: We want to examine whether users who tweet from the same spatial neighborhood and who use the same *hashtags* in their tweets are close in the followers graph. This kind of scenario can be used to verify the results of a geo-social query as in [APP13] or a kRNN query as in [EP13b]. We expect that users who use the same hashtags are close in the followers graph (probably one follower of the other). We pick a very popular hashtag from an area in NYC *job* and we use TwitterViz to choose two random users who used *job* in their tweets. *TwitterViz* visualizes the resulting subgraph, showing that $u1$ and $u2$ are three hops apart. Figure 6.3 shows how we use *TwitterViz* spatial and graph analysis capabilities in order to gain insights.


User: @JobsAtBarnabas
Tweet: #Healthcare #Job in #WestOrange, NJ: 340B Analyst at Barnabas Health (Corp) http://t.co/a
#TweetMyJobs
TweetID: 562359754702336000
Sentiment: positive Score: 0.727832
Location: New Jersey
TimeStamp: Mon Feb 02 23:19:48 EET 2015

```
MATCH p=allshortestpaths((n:User)-[]-(m:User)) WHERE n.Username='tmj_nwk_acct' and m.Username='JobsAtBarnabas' RETURN nodes(p)
```

(17623) User: tmj_nwk_acct

Figure 6.3: Use Case: Find paths between co-tagged users

The capabilities of *TwitterViz* for spatio-temporal and graph analysis and visualization are apparent. Scenarios such as the above, that leverage the use of spatio-temporal exploration and sentiment analysis, and combine the findings with graph exploration in order to reach to useful results can be used for decision making as well as research. A current prototype of *TwitterViz* is available online[4] among with a video demonstration.

[4]https://web.imis.athena-innovation.gr/redmine/projects/twittervizdemo

Chapter 7

Conclusions and Directions for Future Work

In this thesis we have described works that aim in enhancing location-based services and efficiently execute respective queries. We have introduced a new query, the k-Relevant Nearest Neighbor query (k-RNN), that manages to efficiently combine a spatial and a graph index in order to provide answers to relevant and close by POIs, enhancing the user experience and satisfaction. We have used state of the art algorithms from graph theory in order to optimize our proposed algorithms. We have introduced a new type of join query, the *Spatio-Textual Point-Set Similarity Join* (`STPSJoin`), that seeks pairs of entities that contain similar spatio-textual objects. Combining research from similarity joins and spatial joins we proposed efficient data structures and algorithms to support this type of query. Based on the observation that user-generated content is generated in massive amounts, we aim in proposing a secondary-storage solution to conduct distance-based queries on large-scale graphs. Our in-database framework, namely *COLD*, manages to outperform all other secondary storage solutions, rendering it a complete database system to be used off-the shelf. This allows the use of efficient location-based services by third-party application without the need for custom-based solutions. Since the majority of user-generated content is being made available by social networks, there is an ever increasing need to harness this information and gain valuable insights. Researchers and practitioners on the field seek in analyzing social network data. In spite of this fact, the research in the area has not produced a unified solution for the collection, analysis, querying and visualization based on this kind of data. Our proposed system *TwitterViz* is a complete solution that manages to perform spatial and spatio-temporal analysis and visualization of Twitter data, as well as querying on the Twittersphere, both in real time and offline. This tool can be used by a variety of practitioners in the field, as well as researchers for the support of enhanced location-based services such as k-*R*NN, spatio-textual and Geo-Social queries. What follows is a brief summary of each of the contributions of this PhD thesis, as well as directions for future work.

7.1 Relevant Nearest Neighbor Queries

The scope of this work is to introduce k-RNN queries and to create efficient methods for processing them, i.e., a version of the NN problem that also considers the relevance between query points. Relevance is introduced as a means to navigate a "forest" of POIs by retracing the steps of other people in similar situations, i.e., where did people typically go next. Relevance in our case is defined as the co-occurrence of POIs in texts (Links of Interests - LOI). While a rather simplistic measure, it is adequate to define and evaluate the proposed approach. To solve the k-RNN problem, we define query processing methods that rely on a Spatial Grid and a Relevance Graph to capture the spatial and the relevance aspect of the data, respectively. Landmarks are introduced as a means to improve the k-RNN query performance by optimizing the costly search on the Relevance Graph. *GR-Link-LM** the best performing algorithm, uses the *ALT* algorithm, which combines landmarks and A* search to optimize the graph search. The proposed methods show a significant performance improvement and their performance comes close to that of hypothetical, "ideal" method.

Directions for future work include optimizing the spatial aspect of k-RNN query processing. The challenge is to create an index that more tightly integrates the processing of k-RNN queries. More efficient spatial access methods could be used and our goal is to make the Relevance Graph information part of the index. The specific data and relevance score computation turn this goal into quite a challenge. Additional work will focus on an empirical "relevance" assessment of the query results involving real-world data collected from the Web. Extracting POIs and LOIs from Web documents requires an accurate and powerful geoparsing/geocoding algorithm. Testing Google Places and other NER (named entity recognition) tools has yielded some promising initial results.

7.2 Spatio-Textual Similarity Search on Point Sets

This work studies the problem of similarity search on spatio-textual point sets. We formally define this problem as `STPSJoin`. `STPSJoin` can be employed to identify pairs of similar users, with respect to web documents such as tweets and photographs associated with these users.

In order to efficiently process `STPSJoin` we propose algorithms that leverage different spatio-textual indexes, and integrate an early termination pruning mechanism with a filter and refinement approach. We conducted large-scale experiments on real-world datasets for multiple values on the problem parameters. The better performing algorithm `S-PPJ-F` is orders of magnitude more efficient in terms of execution time than the baseline methods. Finally, `S-PPJ-D` shows improvement over the baseline methods for the case of data-driven partitioned databases, even though it is significantly outperformed by `S-PPJ-F`.

In the future, we plan to focus on distributed architectures in order to further enhance the efficiency of our methods. Furthermore, we intend to integrate additional characteristics in `STPSJoin` queries, which are often associated with web objects, such as temporal information.

7.3 Hub Labels on the Database for Large-Scale Graphs

This work presented COLD, a novel SQL framework for answering multiple exact distance queries on a database for large-scale graphs. Our results have proven solid. Not only the COLD framework answers RkNN and one-to-many queries not handled by previous approaches but it also outperforms previous solutions (including specialized graph databases) on all levels, including query performance, secondary storage utilization and scalability. Thus, COLD is the best overall pure SQL framework for querying huge, large-scale graphs.

Moreover, we have provided comprehensive details about our specific COLD implementation in a popular, open-source database system along with the actual SQL queries used in our implementation, so that our results may be easily reproducible by anyone. Hence, we hope to inspire other researchers to expand the COLD framework towards handling even more complex queries and test-cases.

7.4 Visualizing and Exploring the Twittersphere

The main motivation for the development of the *TwitterViz* framework stems from the fact that, although there exists a variety of applications that use data from twitter in order to provide useful analytics, no such a tool exists to provide a wide range of functionalities in a unified environment. *TwitterViz* manages to leverage both the spatio-temporal characteristics as well as the social aspects of the twitter data and is successful in assisting the non-expert in exploring the Twittersphere. We aim in extending *TwitterViz* to support more complex spatio-temporal and geo-social queries on the multi-modal graph (such as the queries proposed in [EP13b] and [APP13] in order to allow for a more rich analysis.

Bibliography

[AAP15] Nikos Armenatzoglou, Ritesh Ahuja, and Dimitris Papadias. Geo-social ranking: functions and query processing. *The VLDB Journal*, pages 1–17, 2015.

[ADF$^+$12] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Hldb: Location-based services in databases. In *SIGSPATIAL GIS*. ACM, November 2012.

[ADGW11] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, pages 230–241, 2011.

[ADGW12] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Hierarchical hub labelings for shortest paths. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 24–35, 2012.

[AIKK14] Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, January 5, 2014*, pages 147–154, 2014.

[AIY13] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, USA*, pages 349–360, 2013.

[AIY15] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Pruned landmark labeling [online]. `https://github.com/iwiwi/pruned-landmark-labeling`, 2015.

[AJB99] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabesi. The diameter of the world wide web. *CoRR*, cond-mat/9907038, 1999.

[AMV15] Deepak Ajwani, Ulrich Meyer, and David Veith. An i/o-efficient distance oracle for evolving real-world graphs. In *Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments, ALENEX 2015, San Diego, CA, USA, January 5, 2015*, pages 159–172, 2015.

[ANS11a] Marco D Adelfio, Sarana Nutanong, and Hanan Samet. Searching web documents as location sets. In *ACM SIGSPATIAL*, 2011.

[ANS11b] Marco D Adelfio, Sarana Nutanong, and Hanan Samet. Similarity search on a large collection of point sets. In *ACM SIGSPATIAL*, 2011.

[APP13] Nikos Armenatzoglou, Stavros Papadopoulos, and Dimitris Papadias. A general framework for geo-social query processing. *Proc. VLDB Conf.*, 6(10):913–924, August 2013.

[Bas14] Bast, Hannah and Delling, Daniel and Goldberg, Andrew and Müller-Hannemann, Matthias and Pajor, Thomas and Sanders, Peter and Wagner, Dorothea and Werneck, Renato. Route Planning in Transportation Networks. Technical report, Microsoft Research, 2014.

[BCR11] Jaime Ballesteros, Ariel Cary, and Naphtali Rishe. Spsjoin: Parallel spatial similarity joins. In *ACM SIGSPATIAL*, 2011.

[BGM12] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. Spatio-textual similarity joins. *PVLDB*, pages 1–12, 2012.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, 1993.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. SIGMOD conf.*, pages 322–331, 1990.

[BMS07] Roberto J Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.

[BMSW13] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2013.

[BNNK14] Felix Borutta, Mario A. Nascimento, Johannes Niedermayer, and Peer Kröger. Monochromatic rknn queries in time-dependent road networks. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '14, pages 26–33, New York, NY, USA, 2014. ACM.

[BZWM15] Jie Bao, Yu Zheng, David Wilkie, and Mohamed F. Mokbel. Recommendations in location-based social networks: a survey. *GeoInformatica*, 19(3):525–565, 2015.

[CCJ10] Xin Cao, Gao Cong, and Christian S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1-2):373–384, 2010.

[CCJO11] Xin Cao, Gao Cong, Christian S Jensen, and Beng Chin Ooi. Collective spatial keyword querying. In *SIGMOD*. ACM, 2011.

[CCJW13] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[CGK06] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*. IEEE, 2006.

[CHKZ02] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 937–946, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[CJW09] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[Clo05] Paul Clough. Extracting metadata for spatially-aware information retrieval on the internet. In *Proc. Geographic Information Retrieval Workshop*, pages 25–30. ACM, 2005.

[CML11] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 1082–1090, 2011.

[CSM06] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. Efficient query processing in geographic web search engines. In *Proc. SIGMOD conf.*, pages 277–288, 2006.

[CTH$^+$10] Justin Cranshaw, Eran Toch, Jason I. Hong, Aniket Kittur, and Norman M. Sadeh. Bridging the gap between physical location and online social networks. In *UbiComp*, 2010.

[DFHR08] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *Proc. 24th ICDE conf.*, pages 656–665, 2008.

[DGPW11] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In *Proceedings of the 10th international conference on Experimental algorithms*, SEA'11, pages 376–387, Berlin, Heidelberg, 2011.

[DGPW14] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust distance queries on massive networks. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 321–333, 2014.

[DGW11] Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Faster batched shortest paths in road networks. In *ATMOS*, pages 52–63, 2011.

[DGW13] Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hub label compression. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, pages 18–29, 2013.

[DKKLN11] Peter DeScioli, Robert Kurzban, Elizabeth N Koch, and David Liben-Nowell. Best friends alliances, friend ranking, and the myspace social network. *Perspectives on Psychological Science*, 6(1):6–8, 2011.

[DW07] Daniel Delling and Dorothea Wagner. Landmark-based routing in dynamic graphs. In *Proc. of the 6th Int'l conf. on Experimental algorithms*, WEA'07, pages 52–65, Berlin, Heidelberg, 2007. Springer-Verlag.

[DW12] Daniel Delling and Renato Fonseca F. Werneck. Better bounds for graph bisection. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 407–418, 2012.

[DW13] Daniel Delling and Renato F. Werneck. Customizable point-of-interest queries in road networks. In *21st SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013, Orlando, FL, USA, November 5-8, 2013*, pages 490–493, 2013.

[EAPD] Hariton Efstathiades, Demetris Antoniades, George Pallis, and Marios D Dikaiakos. Identification of key locations based on online social network activity.

[EASV15] Christodoulos Efstathiades, Helias Antoniou, Dimitrios Skoutas, and Yannis Vassiliou. Twitterviz: Visualizing and exploring the twittersphere. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings*, pages 503–507, 2015.

[EBSP] Christodoulos Efstathiades, Alexandros Belesiotis, Dimitrios Skoutas, and Dieter Pfoser. Similarity search on spatio-textual point sets.

[EEP] Christodoulos Efstathiades, Alexandros Efentakis, and Dieter Pfoser. Efficient processing of relevant nearest neighbor queries.

[EEP15] Alexandros Efentakis, Christodoulos Efstathiades, and Dieter Pfoser. COLD. revisiting hub labels on the database for large-scale graphs. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings*, pages 22–39, 2015.

[EM97] Thomas Eiter and Heikki Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.

[EN06] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems (5th Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[EOSX10] Jacob Eisenstein, Brendan O'Connor, Noah A. Smith, and Eric P. Xing. A latent variable model for geographic lexical variation. In *EMNLP*, 2010.

[EP13a] Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '13, pages 25–30, New York, NY, USA, 2013.

[EP13b] Christodoulos Efstathiades and Dieter Pfoser. User-contributed relevance and nearest neighbor queries. In *Proc. 13th SSTD Symp.*, 312-329, 2013.

[EP14a] Alexandros Efentakis and Dieter Pfoser. GRASP. extending graph separators for the single-source shortest-path problem. In AndreasS. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 358–370. Springer Berlin Heidelberg, 2014.

[EP14b] Alexandros Efentakis and Dieter Pfoser. ReHub. Extending hub labels for reverse k-nearest neighbor queries on large-scale networks. *CoRR*, abs/1504.01497, 2014.

[EPV11] Alexandros Efentakis, Dieter Pfoser, and Agnès Voisard. Efficient data management in support of shortest-path computation. In *Proc. of the 4th ACM SIGSPATIAL Int'l Workshop on Computational Transportation Science*, CTS '11, pages 28–33, New York, NY, USA, 2011. ACM.

[EPV14] Alexandros Efentakis, Dieter Pfoser, and Yannis Vassiliou. SALT. A unified framework for all shortest-path query variants on road networks. *CoRR*, abs/1411.0257, 2014.

[ETP12] Alexandros Efentakis, Dimitris Theodorakis, and Dieter Pfoser. Crowdsourcing computing resources for shortest-path computation. In *Proc. of the 20th Int'l conf. on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 434–437, New York, NY, USA, 2012. ACM.

[Fag99] Ronald Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[FHM05] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: A new abstraction for information management. *SIGMOD Rec.*, 34(4):27–33, December 2005.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[FLZ$^+$12] Ju Fan, Guoliang Li, Lizhu Zhou, Shanshan Chen, and Jun Hu. Seal: Spatio-textual similarity search. *PVLDB*, 5(9):824–835, May 2012.

[G9̈4] Ralf Hartmut Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4):357–399, October 1994.

[GBSW10] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proc. of the 19th ACM Int'l conf. on Information and knowledge management*, CIKM '10, pages 499–508, New York, NY, USA, 2010. ACM.

[GH05] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proc. 16th SODA conf.*, pages 156–165, 2005.

[GPPR01] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 210–219, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[GSS08] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In J. Ian Munro and Dorothea Wagner, editors, *ALENEX*, pages 90–100. SIAM, 2008.

[GSWY13] Ashish Goel, Aneesh Sharma, Dong Wang, and Zhijun Yin. Discovering similar users on twitter. In *MLG*, 2013.

[GSZS14] Oshini Goonetilleke, Timos Sellis, Xiuzhen Zhang, and Saket Sathe. Twitter analytics: A big data management perspective. *SIGKDD Explor. Newsl.*, 16(1):11–20, September 2014.

[Gut84a] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. SIGMOD conf.*, pages 47–57. ACM, 1984.

[Gut84b] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[GW05] Andrew V. Goldberg and Renato F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In *Proc. of the Seventh Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, pages 26–40, 2005.

[HNR68] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[HS99] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. on Database Syst.*, 24(2):265–318, 1999.

[JFWX14] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB*, 7(12):1203–1214, 2014.

[JS07] Edwin H Jacox and Hanan Samet. Spatial join techniques. *TODS*, 32(1):7, 2007.

[JW03] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proc. 12th WWW conf.*, pages 271–279, 2003.

[KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998.

[KMS06] Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In *IN: 9TH DIMACS IMPLEMENTATION CHALLENGE*, 2006.

[KOTZ04] Nick Koudas, Beng Chin Ooi, Kian-Lee Tan, and Rui Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 804–815. VLDB Endowment, 2004.

[KP12] Felix Kling and Alexei Pozdnoukhov. When a city tells a story: urban topic analysis. In *Proc. of the 20th Int'l Conf. on Advances in GIS*, SIGSPATIAL '12, pages 482–485, New York, NY, USA, 2012. ACM.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[LLDM09] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[LLF12] Sitong Liu, Guoliang Li, and Jianhua Feng. Star-join: Spatio-textual similarity join. In *CIKM*, 2012.

[LLZ$^+$11] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lee, and Xufa Wang. IR-tree: An efficient index for geographic document search. *IEEE Trans. on Knowl. and Data Eng.*, 23(4):585–599, 2011.

[LWK$^+$14] Yu Liu, Fahui Wang, Chaogui Kang, Yong Gao, and Yongmei Lu. Analyzing relatedness by toponym co-occurrences on web pages. *Transactions in GIS*, 18(1):89–107, 2014.

[LWWF13] Cheng Long, Raymond Chi-Wing Wong, Ke Wang, and Ada Wai-Chee Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, 2013.

[LZX$^+$08] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *ACM SIGSPATIAL*, 2008.

[ML12] Julian J. McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 548–556, 2012.

[MLTM15] K. Mouratidis, Jing Li, Yu Tang, and N. Mamoulis. Joint search by social and spatial proximity. *Knowledge and Data Engineering, IEEE Transactions on*, 27(3):781–793, 2015.

[MSA05] Bruno Martins, Mário J. Silva, and Leonardo Andrade. Indexing and ranking in geo-ir systems. In *Proc. Geographic Information Retrieval Workshop*, pages 31–34, 2005.

[MSM10] Jens Maue, Peter Sanders, and Domagoj Matijevic. Goal-directed shortest-path queries using precomputed cluster distances. *J. Exp. Algorithmics*, 14:2:3.2–2:3.27, January 2010.

[NZTK08] Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. The v*-diagram: A query-dependent approach to moving knn queries. *Proc. VLDB Endow.*, 1(1):1095–1106, August 2008.

[PBCG09] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *Proc. of the 18th ACM conf. on Information and knowledge management*, CIKM '09, pages 867–876, New York, NY, USA, 2009. ACM.

[PKZT01] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *Proc. 7th SSTD symp.*, pages 443–459. Springer-Verlag, 2001.

[Pos15] PostgreSQL. The world's most advanced open source database [online]. `http://www.postgresql.org/`, 2015.

[QCC12] Daniele Quercia, Licia Capra, and Jon Crowcroft. The social world of twitter: Topics, geography, and emotions. In John G.

Breslin, Nicole B. Ellison, James G. Shanahan, and Zeynep Tufekci, editors, *ICWSM*. The AAAI Press, 2012.

[QECC11] D. Quercia, J. Ellis, L. Capra, and J. Crowcroft. In the mood for being influential on twitter. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 307–314, 2011.

[QECC12] Daniele Quercia, Jonathan Ellis, Licia Capra, and Jon Crowcroft. Tracking "gross community happiness" from tweets. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 965–968, New York, NY, USA, 2012. ACM.

[RB01] Jan Ramon and Maurice Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.

[RJGJN11] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Proc. 12th SSTD symp.*, pages 205–222, 2011.

[RLS14] Jinfeng Rao, Jimmy Lin, and Hanan Samet. Partitioning strategies for spatio-textual similarity join. In *BigSpatial*, 2014.

[SGAF13] Christian Sengstock, Michael Gertz, Hamed Abdelhaq, and Florian Flatow. Reliable spatio-temporal signal extraction and exploration from human activity records. In *Advances in Spatial and Temporal Databases*, volume 8098 of *Lecture Notes in Computer Science*, pages 484–489. Springer Berlin Heidelberg, 2013.

[SIT09] Maytham Safar, Dariush Ibrahimi, and David Taniar. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Systems*, 15(5):295–308, 2009.

[SK04] Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In *SIGMOD*, 2004.

[SPK13] Georgios Skoumas, Dieter Pfoser, and Anastasios Kyrillidis. On quantifying qualitative geospatial data: A probabilistic approach. In *Proc. 2nd ACM SIGSPATIAL Int'l GEOCROWD Workshop*, pages 71–78, 2013.

[SQZZ15] Yu Sun, Jianzhong Qi, Yu Zheng, and Rui Zhang. K-nearest neighbor temporal aggregate queries. In *Proc. 18th Int'l EDBT Conf.*, pages 493–504, 2015.

[TACGBn$^+$11] Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proc. 20th CIKM conf.*, pages 1785–1794, 2011.

[TSF$^+$15] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.

[VJJS05] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark S. Spatio-textual indexing for geographical search on the web. In *Proc. 5th SSTD symp.*, pages 218–235, 2005.

[XWL$^+$11] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. *TODS*, 36(3):15:1–15:41, August 2011.

[XZLX10] Xiangye Xiao, Yu Zheng, Qiong Luo, and Xing Xie. Finding similar users using category-based location history. In *ACM SIGSPATIAL*, 2010.

[YL12] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 745–754, 2012.

[YLL$^+$10] Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S. Tseng. Mining user similarity from semantic trajectories. In *LBSN*, 2010.

[YPMT06] Man Lung Yiu, D. Papadias, Nikos Mamoulis, and Yufei Tao. Reverse nearest neighbors in large graphs. *Knowledge* *and Data Engineering, IEEE Transactions on*, 18(4):540–553, April 2006.

[ZCM+09] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony KH Tung, and Masaru Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, 2009.

[ZLTZ13] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: An efficient index for knn search on road networks. In *Proceedings of the 22nd ACM International Conference on Conference on Information Knowledge Management*, CIKM '13, pages 39–48, New York, NY, USA, 2013. ACM.

[ZOT10] Dongxiang Zhang, Beng Chin Ooi, and Anthony KH Tung. Locating mapped resources in web 2.0. In *ICDE*, 2010.

[ZXW+05] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *Proc. 14th CIKM conf.*, pages 155–162, 2005.

[ZZM+11] Yu Zheng, Lizhu Zhang, Zhengxin Ma, Xing Xie, and Wei-Ying Ma. Recommending friends and locations based on individual location history. *TWEB*, 5(1):5, 2011.