MONASH University

Query Processing in Location-Based Social Networks

Ammar Sohail

A thesis submitted for the degree of Doctor of Philosophy at

Monash University in 2018

Clayton School of Information Technology

Copyright Notice

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: 

Print Name: Ammar Sohail

Date: 01 September 2018

Publications during Enrolment

1. Ammar Sohail, Ghulam Murtaza, and David Taniar. **Retrieving Top-k Famous Places in Location-Based Social Networks**. In *27th Australasian Database Conference, ADC 2016:17-31* (ERA B).

2. Ammar Sohail, Park Jeongho, Andreas Züfle and David Taniar **Query Processing in Location-based Social Networks**. In *International Workshop on Social Computing*, (IWSC):1379-1381, 2017, co-located with *26th International World Wide Web Conference,* (WWW) 2017 (ERA A).

3. Ammar Sohail, Muhammad Aamir Cheema and David Taniar **Social-Aware Spatial Top-k and Skyline Queries**. In *The Computer Journal*,62,2018 (ERA A*).

4. Ammar Sohail, Arif Hidayat, Muhammad Aamir Cheema and David Taniar **Location-Aware Group Preference Queries in Social-Networks**. In *29th Australasian Database Conference, 2018:53-67*

5. Ammar Sohail, Muhammad Aamir Cheema and David Taniar **Geo-Social Temporal Top-k Queries in Location-Based Social Networks**. In *WWW Journal* (ERA A) (Submitted and Under Review).

Thesis including published works declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 4 original papers published in peer reviewed journals or conferences and 1 paper has been submitted and is under review. The core theme of the thesis is to study geo-social queries on social networks. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology, Monash University, under the supervision of Dr. Muhammad Aamir Cheema, Assoc Professor David Taniar and Dr. Iqbal Gondal.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapter 3, 4, 5 and 6 my contribution to the work involved the following:

Thesis Chapter	Publication Title	Status (Published in press, accepted or returned for revision, submitted)	Nature and % of student contribution	Co-author name(s) Nature and % of Co-author's contribution	Co-author(s), Monash student Y/N
3	**1.** Retrieving Top-k Famous Places in Location-Based Social Networks **2.** Query Processing in Location-based Social Networks	Published Published	1. 80% problem formulation, technique, algorithm, experiment 2. 75% problem formulation, technique, algorithm, experiment	1. Ghulam Murtaza, input into paper 10% 2. David Taniar, input into paper 10% 1. Andreas Zufle, input into technique and algorithm 5% 2. Park Jeong-ho, input into writing code 15% 3. David Taniar, input into paper 5%	No No No No No
4	Social-Aware Spatial Top-k and Skyline Queries	Published	80% problem formulation, technique, algorithm, experiment	1. Muhammmad Aamir Cheema, input into technique and algorithm 10% 2. David Taniar, input into Paper 10%	No No
5	Location-Aware Group Preference Queries in Social-Networks	Published	70% problem formulation, technique, algorithm, experiment	1. Arif Hidayat, input into technique 10% 2. Muhammmad Aamir Cheema, input into technique and algorithm 10% 3. David Taniar, input into Paper 10%	Yes No No
6	Geo-Social Temporal Top-k Queries in Location-Based Social Networks	Submitted	80% problem formulation, technique, algorithm, experiment	1. Muhammmad Aamir Cheema, input into technique and algorithm 10% 2. David Taniar, input into Paper 10%	No No

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student signature: [redacted] Date: 01 September 2018

The undersigned hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor signature: [redacted] Date: 01 September 2018

Acknowledgements

First and foremost, I would like to thank God for all the blessings He bestowed upon me and for providing me persistent guidance throughout my life. Without His blessings, I would have not been able to finish my thesis.

I would like to express my special appreciation and thanks to my advisor Dr. Muhamamd Aamir Cheema, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a researcher. Your advice on both research as well as on my career has been priceless. Besides my advisor, I would like to thank the rest of my thesis committee: Assoc Professor David Taniar and Dr. Iqbal Gondal, for their insightful comments and encouragement.

Also, I am thankful to Monash University for the financial support and arrangements during my PhD study.

I thank my fellow research mates, Arif Hidayat, Anasthasia Agnes Haryanto, Chaluka Salgado, Nasser Allheeib, Tenindra Abeywickrama, Utari Wijayanti and Zhou Shou, for the stimulating discussions and for all the fun we have had in the last few years.

A special thanks to my family. Words cannot express how grateful I am to my mother and father, for all of the sacrifices that you have made on my behalf and your prayer for me was what sustained me thus far. An extra special thanks to my sincere father, Tariq Sohail Sheikh, for always loving and supporting me. Finally, I would like to express appreciation to my beloved wife Rauqia Ammar who spent sleepless nights with and was always there to support in

the moments when there was no one to answer my queries. Last but not least, a loving thanks to my siblings and my in-laws for their prayers and support and a very very special thanks to my daughter, Zara Ammar, for always being with me and putting a big smile on my face during stressful times.

Abstract

Recent advances in location-acquisition techniques and GPS-embedded mobile devices have essentially enhanced the user experience in location-based services. The widespread of these technologies and devices have made location information an integrated part of social networks resulting in the emergence of new concept called Location-Based Social Networks (LBSNs). Similarly, these LBSNs allow people to record and share their location and are a rich source of information which can be exploited to study people's various attributes and characteristics.

Location-based social services usually process various types of queries such as Geo-Social Top-k queries, Geo-Social Nearest Neighbours queries and Geo-Social Skyline queries to offer many interesting services such as in disaster management, public health, security, tourism, marketing etc. Undoubtedly, social connections play a vital role in our daily lives to enable us in making right decisions in various activities and events and thus impose some influence on us. In our daily life, besides our own preference, we usually turn to our friends for opinions of songs, restaurants or movies.

Specifically, we studied four different Location-Based queries (Top-k, Skyline, Aggregate Nearest Neighbour and Temporal) on Social Networks in this work. In the first work, we formalize a new problem namely, Top-k famous places TkFP query and propose efficient query processing techniques. We proposed efficient indexing techniques which facilitate combine filtering of both social and spatial components. Similarly, we proposed effective query processing algorithms to produce top-k query results. We conducted exhaustive

experiments on real and synthetic datasets to demonstrate the effectiveness of our techniques.

We also extended our aforementioned work to solve Socio-Spatial Skyline Query (SSSQ) which does not require any scoring function and user also does not need to have adequate domain knowledge to decide good value of preference parameter between social and spatial components. In particular, we introduce three different techniques i.e., I) Social- First II) Spatial-First and III) Hybrid; and efficient indexing and pruning methods to answer the query efficiently.

In the third work, we studied a problem of finding top-k places considering their distance from a group of query users and popularity of the place among each query user's social connections (e.g., the number of check-ins at the place by each q's friends). We presented Branch-and-Bound approach to solving the problem followed by several optimization techniques to further improve its performance.

In the fourth work, we added a third dimension (time) to our first work and studied a problem of finding top-k places considering their distance from the query user q and popularity of the place among q's social connections during a certain period of time. At first, we presented two different approaches i.e., Social-First and Spatial-First to solve our problem and then we propose our main algorithm called Hybrid. We conducted an exhaustive evaluation of the proposed schemes using real dataset and demonstrate the effectiveness the approaches. Our experiments showed that our main algorithm outperforms the other two.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Online social networks provide a rich source of valuable information that can be used for tracking public opinions, political trends, disseminate news, or just to socialise by finding the right group of people to communicate with [1]. Worldwide penetration of online social networks is ever increasing and it is estimated that 72 percent (around 2.5 billion) of worldwide internet users use social networks, which is 3 percent up from 69 percent in 2016 [2]. Online activities involving social media are one of the most popular online activities with very high people engagement rates and are opening new horizons of mobile possibilities.

Moreover, North America ranks first among regions where social media is highly popular, with a social media penetration rate of 66 percent. In 2016, more than 81 percent of the United States population had a social media profile. In another recent report, it is shown that almost 79 percent Australians use online social networks and nearly 60 percent connect through social media everyday [3]. The immense popularity of social media has resulted in the generation of large volume of social data and extracting information from

these social networks is widely practiced by commercial companies, government agencies and individuals . For instance, the data obtained from social networks have been used to monitor disease outbreaks [4, 5]. Similarly, the social media integration in businesses has recently reached the highest level recorded. More than 51 percent of the small and 58 percent of the medium businesses have a social media presence, while for large businesses the rate has reached to 85 % [4, 5]. These statistics show that businesses extensively utilize the social network tools to obtain various benefits such as market research, increased revenue, recruitment ,brand development and networking etc [6]. The social networks have also been researched to assist law enforcement agencies in solving various criminal cases. For example, the US National Security Agency (NSA) uses social media data for various national security projects [7] such as identifying suspects of a crime. Similarly, social networks allowed Richmond City Policy Department to locate a homicide witness and compile data related to sixteen convenience store robberies that were not previously thought to be connected [8]. Further, it is reported in McKinsey report that companies can increase their productivity upto 25 % by incorporating social media technology in their business model. Also, Google revealed in recently conducted survey that 41 percent of professionals use social media search to find information, people and expertise more quickly [9].

Social networks are comprised of various types of entities and are usually represented as a complex graph where nodes represent various entities in the social network (such as users, places or pages) and the edges represent the relationships between different nodes. These relationships are not only limited to friendship relations but also contain other types of relationships such as `works-at`, `born-in`, and `studies-at` etc. In addition, the nodes may also contain spatial information such as a user's check-ins at different locations.

Since social networks are not only about friendships now a days, these are relatively complex structures than ordinary networks and in recent years, the increasing use of mobile devices, location-based services and advancements in location-acquisition technologies (e.g., GPS and Web 2.0), have made location information an essential part of such graphs [10]. This in return, results in the generation of new set of associations between users and locations like user-user, location-location and user-location as shown in Figure 1.1. In addition, these associations are rich sources of information which can be exploited to analyze people's every day behaviour to offer many interesting services such as mobile marketing [11, 12, 13], disaster relief [14, 15, 16] and traffic forecasting [17, 18].

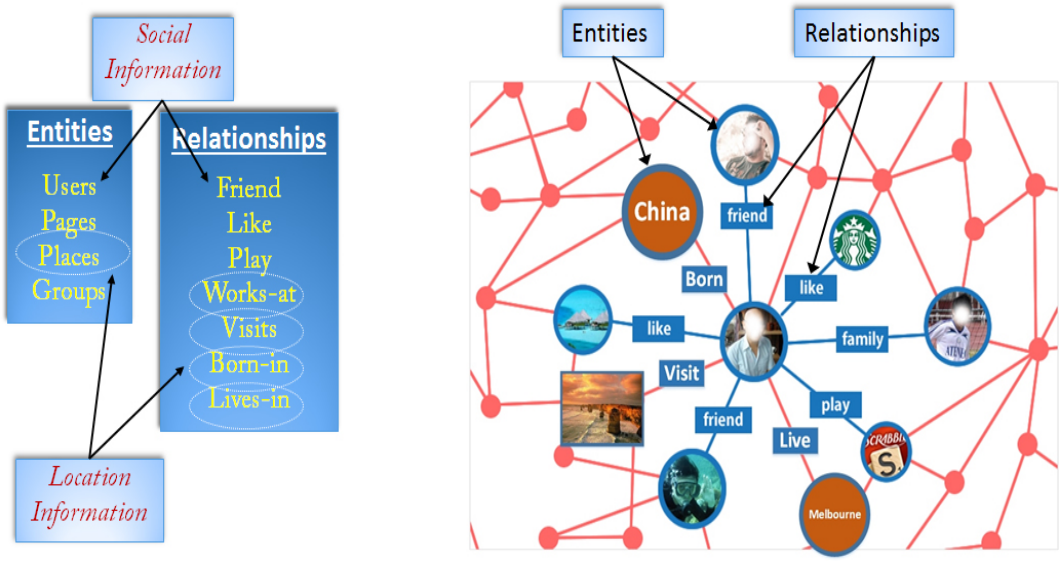


Figure 1.1: Locality Information in Social Networks

This fusion of social and geographical information has given rise to the notion of online social media known as location-based social networks (LBSNs) such as Facebook, Twitter, Foursquare etc. Such LBSNs allow people to record their locations. For example, an activity called `check-in`, through which people can record their visits to various places. Figure 1.2 depicts two different types of entities one of which also constitutes locality information (an entity

of type place) and the entities can be connected through a *check-in* relationship. Similarly, people can search for local places (e.g., *cafés*, restaurants) and different types of offers.

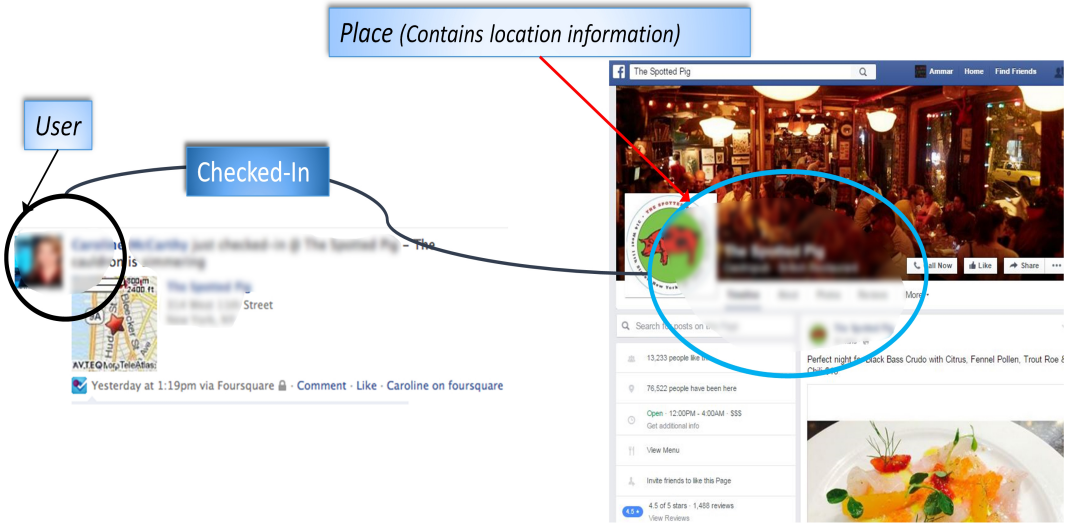


Figure 1.2: Example of Check-In activity

Lets consider another example of a Facebook user Alice who was born in Germany, works at Monash University and checks-in at a particular restaurant. Facebook records this information by linking Facebook pages for *Monash University* and *Germany* with Alice [19], e.g., Alice and Monash University are connected by an edge labelled `works-at` and Alice and Germany are connected with an edge labelled `born-in`. The check-in information records the places the user has visited.

1.1 Motivation

As stated earlier, social networks have applications in a variety of domains like law enforcement [8], national and global security [7, 20], emergency management [21], predicting/monitoring disease spread [4, 5], advertisement [22],

analytics [23] and many more.

Spatial data and social relationships in LBSNs provide a rich source of information which can be exploited to offer many interesting services. Consider the example of a German tourist visiting Melbourne. She may want to find a nearby pub which is popular (e.g., frequently visited) among people from Germany. This involves utilizing spatial information (i.e., nearby pub, check-ins) as well as social information (i.e., people who were `born-in` Germany). Similarly, a user may want to find nearby places that are most popular among her friends, e.g., the places most frequently visited by her friends. The ability to efficiently find and exploit relevant information from the social network data is at the backbone of all such applications. However, current technology to effectively and efficiently exploit social networks for search related purposes is not as mature as the advances achieved by the Web search engines. Specifically, the current technology suffers from the limitation where the search results do not consider geographical location of the query which leads to poor-quality and often incoherent results.

For example, consider a user who is interested in finding fast food restaurants close to her geographical location and issues a query "fast food restaurants". The search that retrieves the query results while considering the query location is known as location-aware search. Majority of the existing work for search on social networks do not handle location-aware search. Figure 1.3 illustrates an example where a query is issued by the user's computer on Facebook Graph Search in Monash University Clayton campus. However, the returned results contain the restaurants from Clayton in USA. Ideally, a location-aware social search engine would return the nearby restaurants from Monash University Clayton campus and in addition to that, may consider the user's social relationships in social network to further tailor the results containing restau-

rants that have been visited/liked by the people connected to her in the social network (for example, her colleagues, friends, people from the same country etc.).

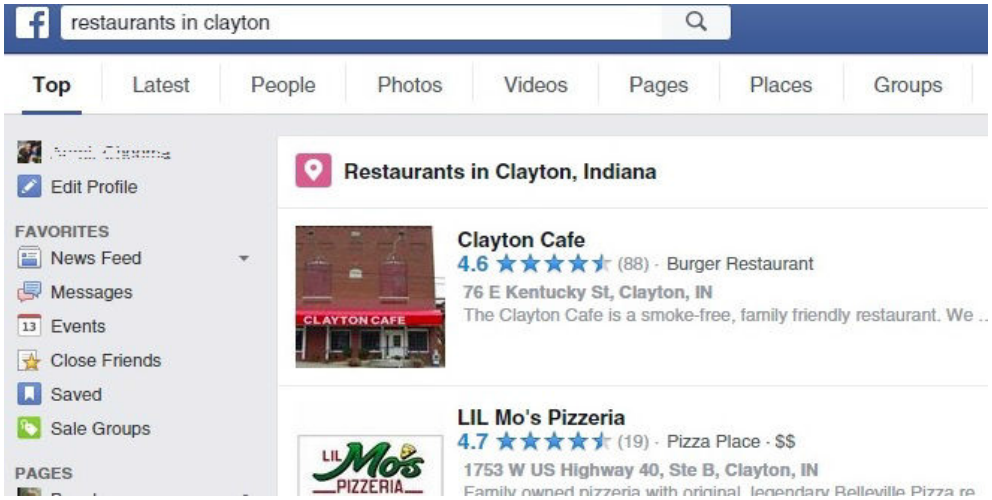


Figure 1.3: The user's current location is Clayton Australia but query returns the results from Clayton in Indiana, USA.

Facebook Graph search is based on Unicorn [19] which is an online in-memory social graph-aware indexing system. Unfortunately, Unicorn suffers from few limitations, for example, it cannot handle location information in social networks effectively. Because, it does not leverage highly developed spatial data structures, rather, it considers each location as a *Facebook Page* and represents the relationships such as lives-in between the page and users by creating edges. See Section 2 for more detail.
Similarly, previous works have various limitations and do not handle geo-social queries properly. Therefore, motivated by this, we focuses on geo-social queries in this thesis .

Existing relevant literature can be classified into various categories like, Geo-Social Top-k Queries, Geo-Social Skyline Queries, Geo-Social Group Queries and Geo-Social Temporal Queries. Next, we present definitions of the different

types of queries.

1.1.1 Geo-Social Queries

Geo-Social queries aim at extracting objects of interest (e.g., friends, restaurants) combining both the social relationships and the current location of objects under consideration. For example, a French immigrant recently moved to Toronto may be interested in finding social activities for french people. This involves exploiting social relationships between social media users (people who were born-in France) and their current location data (living in Toronto). Geo-Social queries can be further classified into the following types of queries.

1.1.1.1 Geo-Social Range Queries

A geo-social range query (range search) finds objects of interest that are within distance r from a given query location q based on given social criteria. For example, a query user q may be interested in finding her friends within the range of 2 kilometres.

1.1.1.2 Geo-Social KNN Queries

A k nearest neighbour query (kNN) retrieves k closest objects of interest to a given query point q. In other words, a kNN query returns a set of k objects such that there does not exist any other object which is closer to the q than any of the k returned objects. A kNN query is called NN query when $k = 1$. In geo-social context, given a query point q and a positive integer k, a kNN query retrieves k closest objects from q among the objects that satisfy a given social criterion. For example, a Samsung employee may be looking for 2 people in Melbourne close to her location who also work in Samsung to join her for a lunch.

1.1.1.3 Geo-Social Top-k Queries

A geo-social top-k query aims at retrieving a list of k objects (e.g., friends, restaurants) with highest scores, where the score of each object is computed using a user defined preference function based on given spatial and social relevance. The spatial relevance is based on how close the object is to the query q and social relevance is based on social relationships in the social network to be considered. For instance, a Canadian tourist may be looking for 2 restaurants that are popular among people from Germany and are close to her.

1.1.2 Geo-Social Skyline Queries

Like top-k query, a skyline query takes into account multiple attributes to retrieve objects of interest. However, a skyline query does not require a preference function to be specified by a query user. Given a query user q, an object o dominates another object o' if o is preferable for q than o' in at least one attribute and is not worse than o' on all other attributes. For example, a German tourist considers distance and popularity of a restaurant among customers who were born-in Germany to choose the restaurant for dinner. Thus, a restaurant x dominates restaurant y if it is closer to q than y and is socially more relevant for q than y. Informally, a *skyline query* returns every object o that is not dominated by any other object.

1.1.3 Geo-Social Temporal Queries

A geo-social temporal query aims at retrieving a list of objects (e.g., friends, restaurants) with highest ranking scores according to user's defined spatial, social and temporal criteria. For example, find recently opened fast food restaurants in Melbourne that became popular or retrieve a list of popular

amusement parks during Christmas holidays.

1.1.4 Geo-Social Group Queries

Given two datasets P (e.g., places) and Q (queries), a group (aggregate) nearest neighbor (GNN) query retrieves places of interest $p \in P$ with smallest aggregate distance(s) to the query points in Q. Assuming, for example, n users at locations $q_1, q_2, ...q_n$, a GNN query retrieves place $p \in P$ that minimizes the sum of distances $|p, qi|$ for $1 \leq i \leq n$ that the query users have to travel in order to meet there. Similarly, another GNN query may return the place $p \in P$ that minimizes the maximum distance that any query user has to travel. In geo-social context, given a set of query points Q, a GNN query retrieves a place $p \in P$ that minimizes the sum of distances $|p, qi|$ while satisfying a given social criterion. For example, some conference attendees would like to go out for dinner together and for this purpose, we may consider their respective locations and given social criterion to recommend restaurants.

All of the aforementioned queries have not been studied before and we are the first to study all these queries. We note that most of the existing studies either cannot handle this or are too inefficient. For details, see Section 2.

1.2 Research Questions and Contributions

The main focus of the research in this Ph.D. thesis is to study geo-social queries that have a variety of applications as discussed earlier. To overcome current limitations in previous work, we investigate and propose efficient indexing techniques and algorithms to process geographical and social components together. Specifically, we address following research questions in this thesis and make various contributions to each research question as described next.

1.2.1 *RQ-1: Location-Based Top-k Queries in Social Networks*

In first chapter, we formalize a new problem namely, *Top-k famous places (T_kFP)* query and propose efficient query processing techniques. Specifically, a T_kFP query retrieves top-k places (points of interest) ranked according to their spatial and social relevance to the query user where the spatial relevance is based on how close the place is to a given location and the social relevance is based on how frequently it is visited by the one-hop neighbors of the query user in the social graph. In addition, we propose three algorithms to efficiently answer the query by leveraging proposed index structures. Our extensive experimental study conducted on real and synthetic data sets demonstrates the effectiveness of proposed techniques. Our research [24] on this research question was published in *Australasian Database Conference* **(ADC)**, 2015.
In-addition to this, we develop a demonstration to show the real application of *Location-Based Top-k Queries* in social networks. This demonstration enables participants to view actual output of T_kFP query containing *top-k* places checked-in by friends in a given region. The demo [25] was published in *International Workshop on Social Computing* **(IWSC)** 2017, co-located with *26th International World Wide Web Conference* **(WWW)** 2017.

1.2.2 *RQ-2: Spatial Skyline Queries in Social Networks*

A *top-k* query uses a scoring function that combines social and spatial scores to rank the objects. However, users must have adequate domain knowledge to be able to decide upon a good value of α. In particular, it is not easy to define a scoring function (e.g., due to incompatible attributes, different distributions

of attributes, the inability of users to choose a good scoring function) [26]. Therefore, to complement our *Top-k famous places* query, we extend our work to study skyline queries which return those objects that are not dominated by any other object and are within given range r (i.e., $||q, p|| < r$). In order to answer such queries, we compute social and spatial scores of each place; and maps it to a space where x-coordinate refers to spatial score and y-coordinate refers to social score. Then, the set of places which are not dominated by any other place are returned. The intuition behind using a range r is that sometimes users are not interested in places that are too far. We propose three techniques to process the query and propose efficient algorithms and indexing techniques. Furthermore, we conduct an extensive experimental study on real and synthetic data sets and compare their performance with [27] and find that our algorithms are significantly better than the competitor. Our research [28] on this research question was published in **The Computer Journal** in 2018.

1.2.3 *RQ-3: Spatial Group Top-k Queries in Social Networks*

In many applications, a group of users may want to plan an activity and to find a point of interest (POI) for example, some conference attendees would like to go out for dinner together. For this purpose, we may consider their respective locations and social circles to recommend required POIs. Therefore, in our third work, we study a problem of finding top-k places considering their distance from the group of query users Q and popularity of the place among each query user $q_i \in Q$'s social connections (e.g., the number of check-ins at the place by each q's friends). To the best of our knowledge, we are the first to study the $SG\text{-}Top_k$ query that retrieves nearby places popular among a

particular group of users w.r.t. each query user $q_i \in Q$ in the social network. We present *Branch-and-Bound* algorithm to solve the problem followed by some proposed optimization techniques to further improve its performance. We conduct an exhaustive evaluation of the proposed schemes using real dataset and demonstrate the effectiveness of the schemes. Our research [29] on this research question was published in *Australasian Database Conference* **(ADC)**, 2018.

1.2.4 *RQ-4: Spatial Temporal Top-k Queries in Social Networks*

We present three different algorithms to answer such queries efficiently. In addition, we propose some index structures to store data which helps in pruning unnecessary objects.

In this study detailed in chapter four, we add a third dimension (time) to our first work and study a problem of finding top-k places considering their distance from the query user q and popularity of the place among q's social connections during a certain period of time. Consider an example of a visitor from Switzerland visiting Melbourne. She maybe keen in finding a nearby *café* which serves *Rösti* (a traditional Swedish hot cake) with coffee and has become popular (e.g., frequently visited) among people from Switzerland in last year. This involves utilizing spatial information (i.e., nearby *café*, check-ins), social information (i.e., people who were `born-in` Switzerland) as well as temporal information (i.e., *cafés* that are visited during last year). We formalize the problem as a *Geo-Social Temporal Top-k ($GSTT_k$)* query and propose three techniques to answer the query: I) Social-First, II) Spatial-First and III) Hybrid. To improve the query performance, we propose few index

structures (e.g., FCR-Tree, 3D Check-Ins RTree) which enable flexible data management and algorithmic design. We conduct extensive experiments using real and synthetic data sets to demonstrate the effectiveness of the proposed algorithms. Our research on this research question has been submitted to **WWW Journal** and currently is under review.

1.3 Thesis Organization

Below, we present the structure of the rest of the thesis.

- Chapter 2 presents a review of the related work
- Chapter 3 describes our work on top-k spatial queries in social networks
- Chapter 4 covers our work on spatial skyline queries in social networks
- Chapter 5 constitutes our work on spatial group top-k queries in social networks
- Chapter 6 covers our work on spatial temporal top-k queries in social networks
- Chapter 7 concludes our research and describes several possible directions for future work

Chapter 2

Literature Review

In this chapter, we provide some background on two types of queries that is, spatial only queries and geo-social queries. Since we studied geo-social queries in this thesis, we will also cover some background on spatial only queries to better understand the geo-social queries.

In Section 2.1, we present an overview of geo-social queries. In Section 2.2, we describe kNN queries and Section 2.3 presents an overview of the work involving top-k queries which return upto k objects based on given criteria. Section 2.4 provides literature review on skyline queries where a query returns non-dominated objects without requiring user-defined preference criteria. A brief literature on group queries is presented in section 2.5 where the query involves more than one query objects. In section 2.6, we give an overview on temporal queries where such queries retrieve results based on given temporal properties and in sections 2.7 and 2.8, we present an overview of bulk loading techniques and some other related work.

2.1 Geo-Social Queries

The integration of geo-spatial data into social networks has enabled users to utilize social networks to assist them in day to day decision making. Using these loacation-based social networks, users can easily share their geo-spatial locations and location-related contents in the physical world via online platforms. For example, a user with a mobile phone can share comments with her friends about a restaurant at which she has dined in. Other users can expand their social networks using friend suggestions derived from overlapped location histories. For instance, people who constantly hike on the same mountain can be put in contact. The location dimension bridges the gap between the physical world and the digital online social networking services, giving rise to new opportunities and challenges in traditional recommender systems. [30]

This gives rise to new types of queries called, *Geo-Social* queries which aim at extracting objects of interest (e.g., friends, restaurants) combining both the social relationships and the current location of objects under consideration. For example, a French immigrant recently moved to Toronto is interested in finding social activities for french people. Geo-Social query processing is an emerging field and getting attention of research community these days [31, 32, 33, 34]. Below, we provide comprehensive overview of some of the works done so far in this domain.

In [35], an algebraic model to answer geo-social queries is proposed. This model consists of set of operators to query the geo-social data. They replicate the graphs to represent spatial and social components which can be very large in terms of social networks thus, making query processing more cumbersome. Huang et al [36] studied a geo-social query that retrieves a set of nearby friends of a user that share common interests, without providing concrete query pro-

cessing algorithms. [37] defined a new query namely, *Geo-Social Circle of Friends* to retrieve the group of friends in geo-social settings whose members are close to each other based on their geographical and social circumstances such as for group sports, social gathering and community services. Yang et al [38] introduced another type of group query by extending the work presented in [37] namely, *Social-Spatial Group Query* (SSGQ) which is useful for impromptu activity planning. In addition, nearest neighbour queries have been widely applied in location-based social networks recently[39, 40, 41, 42, 43].

Nikos et al [44] proposed a framework to process geo-social queries. Specifically, it segregates social, geographical and query processing components. Let's assume a user wants to find k nearest friends in a given region. To process the query, first they get all friends of her, followed by fetching all users in given region. Subsequently, the intersection is performed on these two sets to answer the query. The key limitation involved here is that the number of acquaintances and number of users in a given region can be very large thus, performing intersection is not efficient. Figure 2.1 illustrates the proposed framework to process the geo-social query. It is clearly shown that social and spatial components are processed separately followed by combining the results before release.

In addition to the proposed framework, they proposed some algorithms for primitive queries to support social and spatial query processing modules like $GetFriends(u)$ returns the set of friends of u. Similarly, $AreFriends(u_i, u_j)$ returns true if given users are friends and false otherwise. On the other hand, for the spatial component, $RangeUsers(u, r)$ which returns users in radius r from query user q and $NearestUsers(q, k)$ which fetch k users nearest to q are proposed. They also proposed a $NearestStarGroup$ query by combin-

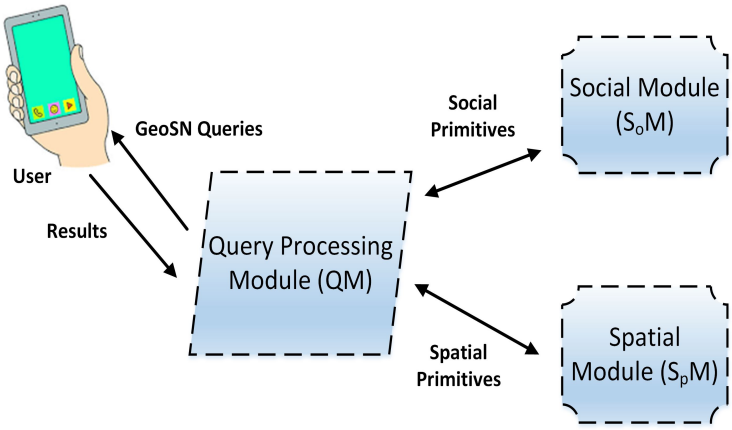


Figure 2.1: Geo-Social Framework

ing primitive queries to return a user group which makes a sub-graph of social network of user u provided that the aggregate euclidean distance is minimized.

Eunjoon et al [45] proposed a model to predict the mobility of the users namely, *Periodic and Social Mobility model* which consists of three components: i) a model of spatial locations visited by a user quite often ii) a model of temporal patterns and iii) a model of movements influenced by her social circle. The proposed model also adheres the limitation of not being capable of processing social and spatial components together to return the query results.

Another effort to manage the social and spatial data is made by Yerach et al [46]. They studied a problem of extracting life patterns of a user from her location history to answer the query which involves the *Sporadic events* in one's daily life. For example, a taxi driver visits many places in a day but seldom visits a same place in the day. Further, they imitated the social and spatial network graphs into socio-spatial graph by connecting these with life pattern edges and embedded the location information into it. Such graphs

are being constructed separately for each user and location. Thus, making it impossible to prune location and social components together.

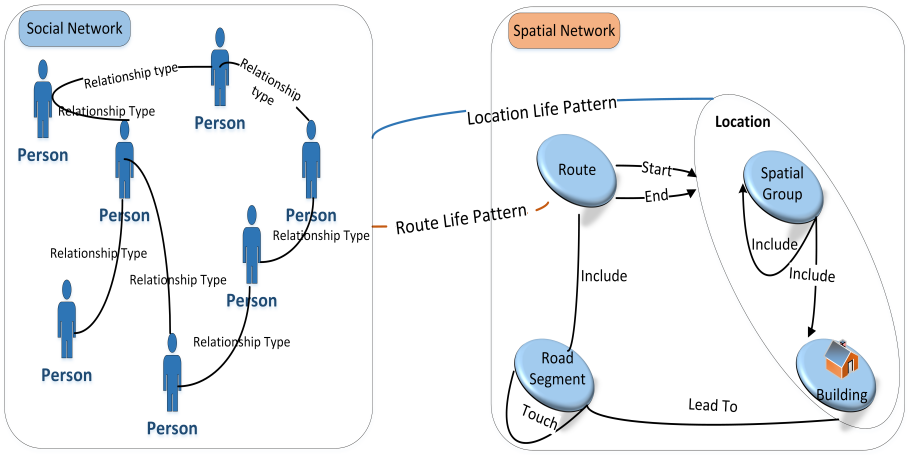


Figure 2.2: Example of Social-Spatial Graph

Possibly, Facebook Graph search is considered as the most advanced and State-of-the-Art social network search system which allows to search including advanced filters on social relationships, for example, find German people who work in Samsung and live in Melbourne. The search system is based on Unicorn [19] which is an online in-memory social graph-aware indexing system supporting search on very large graphs (social networks). Unfortunately, Unicorn suffers from few limitations, for example, it cannot handle location information in social networks effectively. Because, it does not leverage highly developed spatial data structures, rather, it considers each location as a *Facebook Page* and represents the relationships such as lives-in between the page and users by creating edges.

Initially, old search on Facebook was keyword based in which the results were retrieved according to the user's entered keywords. Later on, they started developing a new search product called *Typehead*; which delivered results based

on *"prefix matching"*. But these two techniques exhibit lack of performance due to limited capacity of index size. Thus, they decided to develop a new technique based on inverted index paradigm for graph search and finally ended up with *Unicorn*. There are three major components of unicorn as follows:

- Index: a many-to-many relationship which maps entities to the attributes
- A framework to construct the index on data and updates
- A system to retrieve results based on different type of constraints

In-addition, Unicorn divides index into many instances (verticals) based on different entity types (users, pages, facility points) since, each of them requires different ranking scores. Each entity is maintained by separate Unicorn vertical. A *Top Aggregator* receives a query and returns results after coordinating with all verticals to combine results based on their ranking scores. The scoring process works on one entity at a time which is independent of scores assigned to other entities. For example, if we want to search for restaurants which are famous (are being liked) among Monash employees; firstly, a search on *"user"* vertical is performed to fetch the Monash employees and then a search with these users is performed on *"facility points"* vertical to retrieve restaurants famous among them. Figure 2.3 illustrates the framework of unicorn which consists of three layers: i) top aggregator ii) vertical aggregator and iii) index server.

Let's take another example of finding "friends of my friends who live in Melbourne". Initially, Unicorn retrieves a set of entities (nodes) that are connected through an edge type "my-friends" who lives in Melbourne. In second step, it takes first result set and then fetches the nodes who are the friends

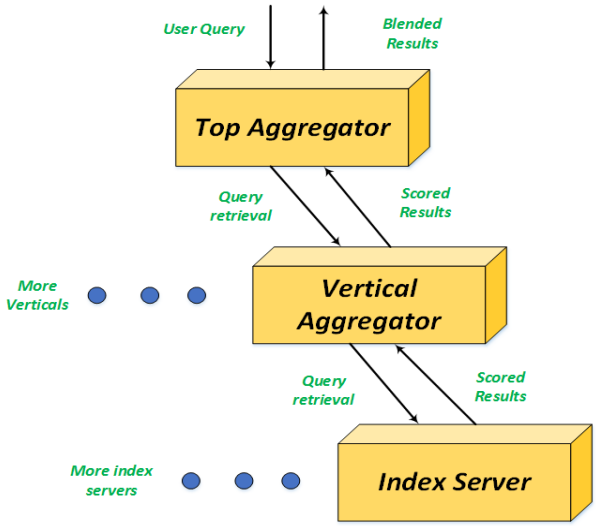


Figure 2.3: Unicorn Framework

of first result set. Although, this architecture can handle large datasets like Facebook and has huge index size, but it still needs to process social and spatial components individually and is unable to embed location information in social graph.

2.2 kNN Queries

A k nearest neighbour query (kNN) [47] retrieves k closest objects of interest to a given query point q. In other words, a kNN query returns a set of k objects such that there does not exist any other object which is closer to the q than any of the k returned objects. A kNN query is called NN query when $k = 1$. The concept and implications of kNN queries have been studied in various fields such as data mining, spatial databases, geo-social databases and computational geometry. These studies have been carried out in various environments such as Euclidean space, road networks and indoor space [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61]. kNN queries have

also been investigated in various conditions such as in a case where certain data is not readily available due to some factors like delay, loss or equipment limitations [62, 63, 64, 65]. Further, such queries have been used in terrains environment, where the elevation information is taken into account [66, 67, 68, 69] and in snapshot queries, where the results are computed based on the current locations of the objects [70, 48, 71, 72, 73, 50, 74, 75]

In addition, kNN queries have been widely applied in location-based social networks recently [39, 40, 41, 42, 43]. Given a query user q with her location and social network information, objects to be retrieved k, a spatial distance constraint and an social constraint, a geo-social nearest neighbour query ($GSNNQ$) retrieves k nearest neighbours from the q satisfying the social constraint. For example, such as, retrieving a set of k nearest friends to the q's location.

2.3 Top-k Queries

kNN queries only consider distance parameter between query user and objects of interests. However, a top-k query considers multiple attributes of the objects of the interest where one of the attributes is the distance to the query point. Formally, a top-k query returns k objects of interests having the best scores. The score of each object is computed using a user's defined scoring function. A user's preference function is a linear function with a weight assigned to each attribute that shows the importance of the respective attribute.

Top-k queries have been extensively studied under various query models [76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87]. Fagin's algorithm (FA) [88], threshold algorithm (TA) (independently proposed in [88], [89], [90]) and no-random access (NRA) [88] propose some of the top-k processing algorithms

that combine multiple ranked lists and return the top-k objects. Ilyas et al [86] give a comprehensive survey of top-k query processing techniques.

In addition, Top-k queries have not only been studied with uncertain data, where exact data is not always available [91, 92, 93, 94], but also with various scoring functions [95, 96], such as monotonic or generic scoring functions. Recent works also incorporate textual relevance of the objects (top-k spatial keyword queries) by integrating inverted index on top of R*-tree [76].

Top-k queries have recently been applied on location-based social networks. A geo-social top-k query aims at retrieving a list of k objects of interest (e.g., friends, restaurants) with highest scores, where the score of each object is computed using a user defined preference function based on given spatial and social criteria. Consider an example of social circle of a query user q in Figure 2.4. Assume, we have a set of few places and are interested in retrieving Top-k places according to their social and spatial relevance.

Figure 2.5(a) illustrates the locations of all the places $P = \{p_1, p_2, p_3, p_4\}$. The query q's location shown in Figure 2.5(a) with $k = 2$ and range $r = 0.15$, has a set of friends $\{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$. The number in bracket next to each place is the check-in count made by q's friends. Figure 2.5(b) shows the Euclidean distances and the visitors of each place amongst q's friends. Lets assume $\alpha = 0.5$ (where α is a weightage constant), the score of the p_1 w.r.t. q is computed as $0.025 + 0 = 0.025$. Similarly, we have $Score(p_2) = 0.185$, $Score(p_3) = 0.205$ and $Score(p_4) = 0.115$. The result of the query q is (p_2, p_3).

Wu et al [27] presented a work involving geo-social queries and proposed a new query named as social-aware *top-k* spatial keyword query (SkSK) which retrieves a list of k objects ranked according to their spatial proximity, textual (e.g. restaurant has menu and different facilities) and social relevance. The

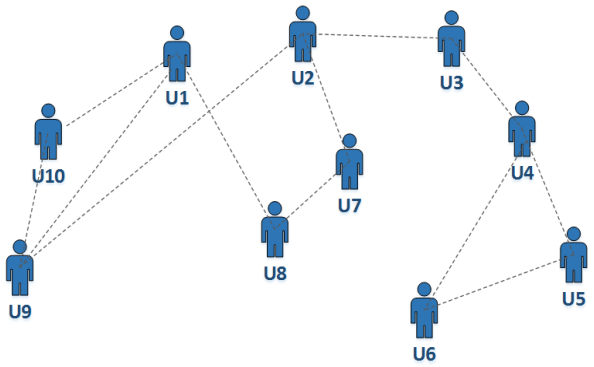
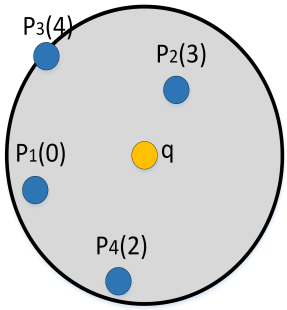


Figure 2.4: Social Network Example



(a) *Range = shaded area*

P	$\|\|q, p_i\|\|$	P_i.visitors	*Final Score*
P_1	0.10	-	*0.025*
P_2	0.08	u_1, u_3, u_9	*0.185*
P_3	0.14	u_5, u_6, u_2, u_7	*0.205*
P_4	0.12	u_7, u_{10}	*0.115*

(b) $\alpha = 0.5, k = 2, r = 0.15$
$F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$

Figure 2.5: Top-k Query Example

social relevance is defined as a function of the users who are "fans" of an object considering how close they are to the query user.

Another work is presented by Jiang et al [97] in which they proposed a method to find *top-k* local users in geo-social media data. First, they extract all tweets in the given range and rank each tweet based on number of replies/forwards to that tweet. For this, they build a tweet thread tree of each tweet and sum-up replies/forwards at each level. This tweets ranking is considered as social score of the user who initiated the tweet. Then, they compute spatial score of each user who has posted tweets in the range. However, their social scoring criteria is not applicable to our problem definition.

Further, they leveraged a *Hybrid Index* design based on Geohash encoding which is actually Quadtree [98] based. This hybrid structure is shown in Figure 2.6 which consists of two parts: forward index and inverted index. Forward index stores the keywords according to their geohash codes and associates each entry to the posting lists which is stored in inverted index. Specifically, each inverted index record refers to tweets in the database.

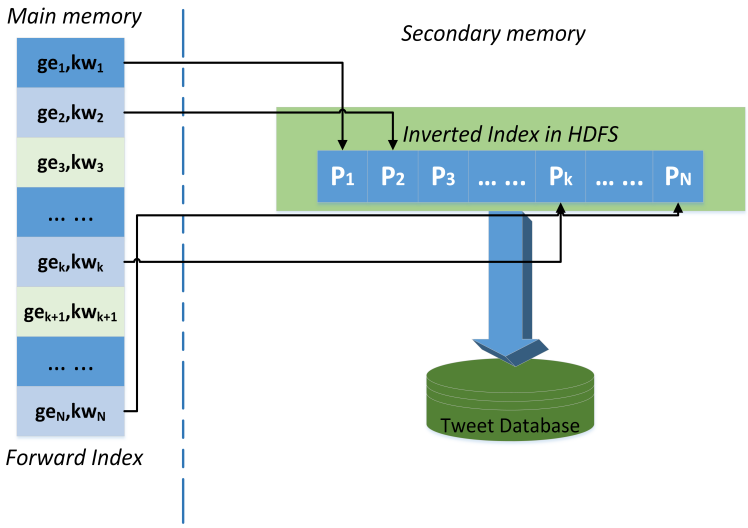


Figure 2.6: Index Structure

Li et al [43] presented a work on influence maximization problem [99]. Given a geo-social network, where each user has a spatial location, and a query q with a given spatial region r and an integer k, the location-aware influence maximization problem retrieves k initial users in given region as seeds to maximize the influence spread, meaning by, the maximum number of users in the query region that are influenced by selected seeds. To address this problem, they proposed two algorithms: i) **Expansion based method** which first checks the users in query region and then expand to their friends and ii) **Spatial based indexes** which divides whole space into the small regions and for every region, users are precomputed and maintained. However, for large

number of k these algorithms are quite expensive.

To elaborate further, users in geo-social network are connected to each other via some sort of social relationship and they also have their current location stored in it. Each user's probability to activate friends depends upon the degree of her vertex, i.e. the number of directly connected users. At first, every user is inactive and if gets selected, its state changes to *active* and also activates its acquaintances in same manner and this will proceed like a chain. The aim is to select k initial seeds who have maximum connectivity to activate maximum number of users. This work also has the same limitation of not being able to prune users together on the basis of their social and spatial feature like either users in a close proximity will be selected first and then will be processed to find the maximum connected user or vice versa.

Ahuja et al [100] presented a work on spatial keyword search by exploiting given user's social network. Generally, keyword search in social networks targets to fetch group of users forming a specific social structure [101, 102]. On the other hand, in spatial keyword search, the aim is to retrieve facility points that satisfy given spatial and textual criteria [103, 104]. In this work, they presented a *geo-social keyword search* which belongs to a class of $top-k$ queries.

For query processing, they proposed an index structure called *Geo-Social Keyword Index* (GSK Index) which is a hybrid index structure that stores users and facility points. The proposed index is grid based with height h and at each level, a cell constitutes all child cells and their contained objects. The key limitation of this approach is that to extract social or spatial information, either we have to find all users in given region first and then compute their social ties and textual relevance with query q or other way round.

2.4 Skyline Queries

A *top-k* query uses a scoring function that combines the scores of multiple attributes under consideration to rank the objects of interest. However, users must have adequate domain knowledge to be able to decide upon preference function. In particular, it is not easy to define a preference function (e.g., due to incompatible attributes, different distributions of attributes, the inability of users to choose a good scoring function) [26]. On the other hand, a skyline query considers multiple attributes same a top-k query does, however, it does not require the query user to specify her preference function. Formally, a skyline query retrieves all objects that are not dominated by any other object. Given a query user q, an object o dominates another object o' if o is preferable for q than o' in at least one attribute and is not worse than o' on other attributes.

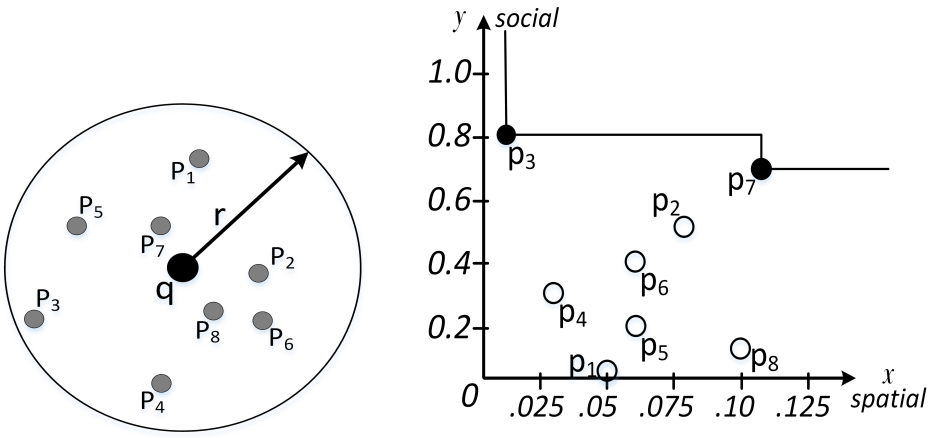
Consider an example in Figure 2.7 where we have a set of places $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ inside range $r = 0.15$, given by query q and a set of friends of q, $F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$. For each place, we compute its spatial and social scores based on its distance from query q and number of friends of q who checked-in at this place. Next, we map each place to a space where x-coordinate refers to spatial score and y-coordinate refers to social score as illustrated in Figure 2.7(b).

For Example in table 2.1, the social and spatial scores of place p_3 are 0.8 and 0.01 respectively and using these scores, p_3 is mapped to the space as shown in Figure 2.7(b). To retrieve query result, we utilize this space to find such places that are not dominated by any other place. For example, place p_7 dominates p_5 because $p_7.social > p_5.social$ and $p_7.spatial > p_5.spatial$. Hence, the *Skyline* query returns p_7 and p_3 which are not dominated by any

other place.

Id	dist(q,p)	p.spatial	Visitors	p.social
P_1	0.10	0.05	-	0
P_2	0.07	0.08	u_1, u_3, u_5 u_7, u_9	0.5
P_3	0.14	0.01	u_1, u_2, u_4 u_5, u_7, u_8 u_9, u_{10}	0.8
P_4	0.12	0.03	u_4, u_6, u_8	0.3
P_5	0.09	0.06	u_4, u_5	0.2
P_6	0.09	0.06	u_3, u_4, u_7, u_9	0.4
P_7	0.04	0.11	u_1, u_2, u_4, u_5 u_8, u_9, u_{10}	0.7
P_8	0.05	0.10	u_4	0.1

Table 2.1: Sample Dataset



(a) Places in Range (b) Mapping

Figure 2.7: Example: Skyline Query

There has been considerable work done in the literature regarding skyline queries [105, 106, 106, 107, 108]. These include computing skyline queries in partially ordered domain, in a distributed settings, in road networks, continuous skyline queries [109, 110, 111, 112, 113, 114, 115] and many more. The skyline operator was first introduced in [116] followed by many generic skyline computation algorithms such as *Block-Nested Loop* (BNL), *Divide and Con-*

quer (DC) approach which were proposed by same authors. Since then, skyline processing has appealed many researchers and has attracted the attention of many in database community. Additionally, a *Bitmap* algorithm was proposed in [117] to improve the original algorithms which involve low cardinality domains i.e., datasets with small number of discrete attributes. There exists few other solutions to process such datasets for instance, as proposed in [107]. Similarly, another approach was introduced in [116] known as *Index* algorithm which divides the dataset into d sorted lists for d optimized measures.

Another R-Tree [118] based approach known as *Nearest-Neighbour* (NN) was proposed in [119]. This approach starts with finding the nearest neighbour to the query and thus, objects dominated by the nearest neighbour can be pruned. [120] proposed a *Branch-and-Bound Search* (BBS) method to overcome the overlapping problem persists in (NN) algorithm. This approach is guaranteed to visit each page of the R-Tree at most once.

The above algorithms can be categorised into two classes: 1) Index-based algorithms and 2) Non-index-based algorithms. BBS, NN, *Index* algorithm and *Bitmap* algorithms fall into the first category. Similarly, second category is comprised of (SF), (DC), (BNL) and *Linear Elimination Sort* for skyline algorithm [108]. Index-based algorithms have advantage over non-index-based algorithms in a way that they only have to access a portion of the dataset to process the skyline while others have to access whole dataset atleast once.

Most of work on skyline computation does not consider social aspect. To the best of our knowledge, there exists only one work [121] in literature that considers both social and spatial aspects for skyline queries. They define skyline query as a set of users who are not dominated by any other user where a user u is said to be dominated by another user u' if u' is closer to the query location and u' is socially closer to the query user. The problem we study in

this thesis is fundamentally different as we focus on returning skyline places (instead of skyline users) where a place p is dominated by another place p' if p' is closer to the query user and the number of q's friends who visited p' is greater than the number of q's friend who visited p. For this, they use *Random walk with restart method (RWR)* to compute social distance which is very expensive and to optimize, they propose a method to approximate the results (social similarity) which does not return exact results.

2.5 Group Queries

Group (Aggregate) nearest neighbour query is one of the most useful operators for analyzing networks and graph data, e.g. social networks, traffic networks, and biological networks. Given two spatial datasets P (e.g., places) and Q (queries), a group nearest neighbor (GNN) query retrieves the point(s) of P with the smallest aggregate distance(s) to points in Q. Let's assume n users at different locations $q_1, q_2 ... q_n$, a *GNN* query outputs the place $p \in P$ that minimizes the sum of distances $|p, q_i|$ for $1 \leq i \leq n$ that the users have to travel in order to meet there. Similarly, in another variant of *GNN* query, the users might be keen in finding a point $p \in P$ that minimizes the maximum distance that they have to travel.

In many applications, a group of users may want to plan an activity to find a point of interest (POI) for example, some conference attendees would like to go out for dinner together. For this purpose, we may consider their respective locations and social circles to recommend required POIs. Group (aggregate) nearest neighbour queries are first presented by Papadias et al. [122] and proposed three different methods MQM (multi query method), SPM (single point method) and MBM (minimum bounding method). The propsed

methods are designed for Euclidean space and are not suitable for criteria involving users' preference. In 2007, Yiu et al. [123] introduces a new query called *top-k* spatial preference query which returns top-k objects whose ranking is defined by other objects around them. Yuan et al. [124] proposed a new query which returns top-k objects based on distance and objects ratings. Many previous studies [123, 125] have proposed to integrate POI properties into POI recommendations however, these works do not consider user preferences.

To improve the efficiency of the algorithms proposed in [123], Rocha-Junior et al. [126] proposed an approach to process spatial preference query based on the materialization technique that leads to significant savings in both computational and I/O costs. Tsatsanifos et al. [127] presented an SRT-index to process top-k spatio-textual preference queries. A spatio-textual preference score is defined for each feature object that takes into account a non-spatial score and the textual similarity to user-specified keywords, but it does not consider the user's location and group preference. The problem of processing spatial preference query on road networks is studied in [128, 129].

To determine the top-k objects using group (aggregate) function, Fagin has given an algorithm *"Fagin's Algorithm or FA"* [130] which is much more efficient than naive. However, the threshold algorithm *"TA"* [89] which is remarkably simple algorithm and is essentially more optimal. Unlike FA, which requires large buffers (whose size may grow exponentially as the database size grows), TA requires only a small, constant-size buffer.

2.6 Temporal Queries

Temporal dynamics and how they impact upon various components of information retrieval (IR) systems have received a large share of attention in

the last decade. In particular, the study of relevance in information retrieval can now be framed within temporal information retrieval approaches, which explain how user behavior vary with time, and how we can use them in order to improve retrieval effectiveness. Temporal queries retrieve query results based on given temporal properties. It is noteworthy that time dimension has strong influence in many domains for example, Topic Detection and Tracking, Spatial queries, Information retrieval, Top-k queries, Geo-Textual queries [131, 132, 133]. Consider an example of a visitor from Switzerland visiting Melbourne. She may be keen in finding a nearby *café* which serves *Rösti* (a traditional Swedish hot cake) with coffee and has become popular (e.g., frequently visited) among people from Switzerland in last year. This involves utilizing spatial information (i.e., nearby *café*, check-ins), social information (i.e., people who were `born-in` Switzerland) as well as temporal information (i.e., *cafés* that are visited during last year).

A specific research work involves incorporating temporal influence into recommender system for better understanding of users' temporal preferences. For example, some early work [134, 135] discovered the dynamics of user preference or interest over time. Recently, researchers started investigating periodic patterns of user preferences (e.g., weekend night interests). One solution is to add a time dimension to user-item matrix and apply techniques introduced in [136]. Work proposed in [137] offers time-aware recommendations using a user-based collaborative filtering method. However, none of the proposed works exploit user's social circle to recommend point of interests (POIs).

POI recommendation is an important task in LBSNs. [138] discussed how to use memory-based techniques to recommend POIs. Further, to improve the techniques, they leveraged other dimensions to capture more information e.g., temporal effects [137]. Similarly, [139] introduced a factorization based POI

recommendation model which incorporates the temporal matching between user movement and POI popularity. They recommend POIs based on user's personal check-in history and explores user's trip trajectories to recommend POIs. However, none of these works consider social relationship to retrieve query results.

Huiji et al [140] investigated the temporal effects on location recommendation on LBSNs. In this work, they introduced a location recommendation framework based on the temporal properties of user movement observed from a real-world LBSN dataset. For temporal, they model daily patterns of check-ins of a user and proposed a User -Location matrix where an entry is check-ins count to that place by the user. They maintain many instances of the matrix based on time distribution. Since their method is for recommendation, it cannot be applied on exact query results retrieval.

2.7 Bulk Loading Techniques

Bulk loading refers to the process of creating an index from scratch for a given data set. In an early attempt on bulk insertion for an R-tree, the data items to be inserted are first sorted by their spatial proximity (e.g., the Hilbert value of the center) and then packed into blocks of B rectangles [141]. These blocks are then inserted one at a time using standard insertion algorithm. Intuitively, the algorithm should give an insertion speed-up of B (as a block of B data items is inserted at a time), but it is likely to increase overlap. Thus, produces a worse index in terms of query performance.

There is another work on the bulk insertion which uses a STLT (small-tree-large-tree) approach [142]. The STLT constructs an R-tree (small tree) from the data set and inserts it into the target Rtree (large tree). To insert a

small tree into a large tree, it chooses an appropriate location to maintain the balance of the resulting large tree. However, this approach has the following shortcoming: if a small tree covers a large area, the node of a large tree into which a small tree is inserted needs to be enlarged to enclose it. This means the STLT only works well for highly skewed data sets [143].

A variant of STLT is the GBI (Generalized Bulk Insertion) technique [143]. In this work, the input data set is partitioned into a number of clusters by grouping spatially close data items into the same cluster. After clustering, from each of these clusters, R-trees are built. Finally, these R-trees are inserted into the target tree one at a time. Data items not included in any cluster are classified as outliers and inserted one by one using normal R-tree insertion. This work alleviated the limitation of the STLT which is highly dependent on data distribution. However, this suffers with the same problem of the R-trees being inserted may increase the overall overlap of the target R-tree.

Another class of bulk operations rely on a buffer strategy for dynamic R-trees [144]. This approach adopts the lazy buffering technique of the buffer tree [145]. Basic idea of their work is to attach buffers to the internal nodes of an R-tree in pre-calculated levels and keep the total size of the buffers to fit in the memory. Then, when an object is inserted, it is stored in the buffer until it gets full. When the buffer is full, data objects in the buffer are pushed down to buffer at the lower level.

2.8 Other related work

The work in geo-social networks can be classified into many categories and for the scope of typical thesis, it is almost impossible to cover all of those. For the sake of brevity, in this section, we will only cover some previous work in

the category of *Recommendation* in geo-social networks. The goal of *Recommendation* is to recommend a set of objects (e.g. friends, restaurants) that a target user might be interested in. Plenty of work has been done in this field but we will cover few of them here.

Ye et al [146] proposed a recommender technique which leverages the collaborative filtering module to aggregate multiple facets like i) user's preferences that can be extracted from her check-in history, ii) user's social associations, which can be extracted from her social network and iii) spatial distance between query q's current location and candidate facility points. Further, the probability of a facility point can be estimated by aggregating social and spatial weighing parameters and if the value of social weighing parameter is equal to 1, then it implies that probability completely depends on social factor. If both weighing parameters are zero, then recommendation only depends on user preferences.

The authors exploit outcome of two large datasets i.e., *Whrrl and Foursquare* by using different values of weighing factors and found that their model allows relatively high precision and recall. In addition, they found that spatial influences have more impact on the possibility of user visiting a facility point than of social associations user.

Since collaborative filtering technique cannot directly address all aspects related to how social and spatial information influence one's choice over many facility points, Liu et al [147] presented a model that combines different geographical factors to rank spatial component such as regional popularity and the Toblers first law of geography. In addition to this, they used a factor

which exploits explicit rating recommendation to implicit feedback by taking into account the skewed count of check-ins.

All aforementioned geo-social queries can also be classified into the *Continuous paradigm* in which the result of the query needs to be updated at all times by considering the dynamic nature of spatial and social data. Users can move to different location and/ social networks are being updated continuously. Some work has been presented to handle dynamic nature of these components.

Chen et al [148] proposed a new type of query namely, *Temporal Spatial-Keyword Top-k Subscription* (TaSK)query which considers following three components to evaluate the relevance of geo-textual objects, i) Text relevance, ii) Spatial distance and iii) Time stamp of the geo-textual object. This query continuously keeps track of the objects (e.g. tweets with spatial information) and updates its top-k result set accordingly. This query is issued by a newly published object only if the ranking score of new object is higher than the current found k_{th} object in result set.

They propose a concept of *Conditional Influence Region* (CIR) based on the notion of *continuous k nearest neighbours* queries. Moreover, *CIR* is leveraged to represent the query and to generate a filtering constraint in accordance to the spatial cell to determine whether the new object is a result of the query or not. Techniques to group and index *TaSK* queries are also devised so that these can be evaluated simultaneously.

Chapter 3

Location-Based Top-k Queries in Social Networks

In this chapter, we study location-based top-k queries in social networks and formalize a new problem namely, *Top-k famous places (T_kFP)* query and propose efficient query processing techniques. Specifically, a T_kFP query retrieves top-k places (points of interest) ranked according to their spatial and social relevance to the query user where the spatial relevance is based on how close a place is to a given location and the social relevance is based on how frequently it is visited by one-hop neighbors of the query user in the social graph. The outline of the chapter is as follows: Section 3.1 contains introduction of the chapter, Section 3.2 lists down the contributions we have made, Section 3.3 presents preliminaries, Section 3.4 describes the proposed techniques, Section 3.5 consists of experimental evaluation, Section 3.6 describes the demonstration built to visualize the query results and Section 3.7 concludes the chapter.

3.1 Introduction

A location-based social network is usually represented as a complex graph where nodes represent various entities in the social network (such as users, places or pages) and the edges represent relationships between different nodes. These relationships are not only limited to friendship relations but also contain other types of relationships such as `works-at`, `born-in` and `studies-at` etc [24]. In addition, the nodes and edges may also contain spatial information such as a user's check-ins at different locations [28]. Consider the example of a Facebook user Sarah who was born in USA, works at Monash University and checks-in at a particular restaurant. Facebook records this information by linking Facebook pages for *Monash University* and *USA* with Sarah [19], e.g., Sarah and Monash University are connected by an edge labelled `works-at` and, Sarah and USA are connected with an edge labelled `born-in`. The check-in information records the places the user has visited.

Spatial data and social relationships in LBSNs provide a rich source of information which can be exploited to offer many interesting services. Consider the example of a German tourist visiting Melbourne. She may want to find a nearby pub which is popular (e.g., frequently visited) among people from Germany. This involves utilizing spatial information (i.e., nearby pub, check-ins) as well as social information (i.e., people who were `born-in` Germany). Similarly, a user may want to find nearby places that are most popular among her friends, e.g., the places most frequently visited by her friends.

The applications of such queries are not only limited to traditional location-based social services. These can also be used in disaster management, public health, security, tourism, marketing etc.. For example, in public safety and crime prevention, law enforcement agencies may be keen on finding frequently

visited places by users who have tweeted about *Drugs, Burglary and Extortion* and have also joined some pages/groups containing/sharing information related to those crimes on social networks. The users are socially connected through an edge (entity) e.g., a tweet, a page or a group in social network and then agencies can exploit one-hop neighbours of the entities to find frequently visited places to raid and prevent drugs dissemination.

Although various types of queries have been studied on LBSNs [100, 31, 149, 46, 37, 38], to the best of our knowledge, none of existing techniques can be applied to answer queries like the above that aim at finding near by places that are popular among a particular group of users; satisfying a social constraint. Motivated by this, in this chapter, we formalize this problem as a *Top-k famous places (T_kFP)* query and propose efficient query processing techniques. Specifically, a T_kFP query retrieves top-k places (points of interest) ranked according to their spatial and social relevance to query user. The spatial relevance is based on how close a place is to a given query location and the social relevance is based on how frequently it is visited by one-hop neighbors of the query user in social graph. A formal definition is provided in Section 3.3.1.

In this chapter, we present three approaches to answer T_kFP query processing called, 1) Social-First, 2) Spatial-First and 3) Hybrid. The first two approaches separately process the social and spatial components of the query and do not require a specialized index. The third approach (*Hybrid*) is capable of processing social and spatial components simultaneously by utilizing a hybrid index specifically designed to handle T_kFP queries.

In addition to this, we develop a demonstration of the proposed query which enables participants to view the actual output of the T_kFP query containing top-k places checked-in by the friends of the query user in given range.

3.2 Contributions

We make the following contributions in this chapter.

1. To the best of our knowledge, we are the first to study the T_kFP queries that retrieves near by places popular among a particular group of users in the social network.

2. We propose a framework to process the query which consists of three approaches which enable flexible data management and algorithmic design. These approaches are, I) Social-First II) Spatial-First and III) Hybrid. The *Social-First* approach first processes the social component (e.g., friendship relations and their check-ins) and then processes the spatial component (e.g., places in given range) where as *Spatial-First* initially processes the spatial component followed by processing the social component. In contrast, the *Hybrid* is capable of processing both social and spatial components simultaneously to answer such queries.

3. We conduct an exhaustive evaluation of the proposed schemes using real and synthetic datasets and demonstrate the effectiveness of the proposed approaches. Since there is no prior algorithm available to solve T_kFP queries, we compare the three proposed algorithms with each other to evaluate their performance. Our experimental results show that *Hybrid* performs several times better than the other two.

4. We develop a demonstration of the proposed query which enables participants to view the actual output of the T_kFP query containing top-k places checked-in by the friends of the query user in given range. Our framework prototype adopts client-server model in which users submit

their queries through a web browser (client) and queries are then forwarded to the server. On server side, the *top-k* POIs are computed and finally results are sent back to the user.

3.3 Preliminaries

3.3.1 Problem Definition

Location Based Social Network (LBSN): A *location-based social network* consists of a set of entities U (e.g., users, Facebook Pages etc.) and a set of places P. The relationship between two entities u and v are indicated by a labeled edge where the label indicates the type of relationship (e.g., `friend`, `lives-in`). LBSN also records check-ins where a check-in of a user $u \in U$ at a particular place $p \in P$ indicates an instance that u had visited the place P.
Score of a place p: Given a query user q, and a range r, the score of a place $p \in P$ is 0 if $||q, p|| \geq r$ where $||q, p||$ is the Euclidean distance between query location and p. If $||q, p|| \leq r$, the score of p is a weighted sum of its spatial score (denoted as $p_{spatial}$) and its social score (denoted as p_{social}).

$$p.score = \alpha \times p_{spatial} + (1 - \alpha) \times p_{social} \tag{3.1}$$

where α is a parameter used to control the relative importance of spatial and social scores. The social score *p.social* is computed as follows. Let F_q denote the one-hop neighbors of the query user considering a particular relationship type, e.g., if the relationship is `born-in` and the query entity is the Facebook Page named Germany, then F_q is a set of users born in Germany. Although our techniques can be used on any type of relationship, for the ease of presentation, in the rest of the paper we only consider the friendship re-

lationships. In this context, F_q contains the friends of the query user q. Let $p.visitors$ denote the set of all users that visited (i.e., checked-in at) the place p. The social score p_{social} is computed as follows.

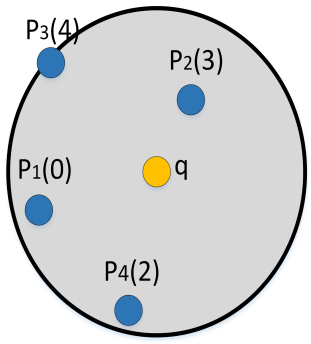
$$p_{social} = \frac{|F_q \cap p.visitors|}{|F_q|} \tag{3.2}$$

where $|X|$ denote the cardinality of a set X. Intuitively, p_{social} is the proportion of the friends of q who have visited the place p.

The spatial score $p_{spatial}$ is based on how close the place is to the query location. Formally, given a range r, $p_{spatial} = 0, (r - ||q, p||)$ where $||q, p||$ indicates Euclidean distance between the query location and p. Note that p_{social} is always between 0 to 1 and we normalize $p_{spatial}$ such that it is also within the range 0 to 1, e.g., the data space is normalized such that $||q, p|| \leq 1$ and $r \leq 1$.

Top-k Famous Places (T_kFP) Query: Given an LBSN, a T_kFP query q returns k places having highest scores, where score ($p.score$) of each place p is computed based on Equation 3.1.

Example 2.1: Figure 3.1(a) illustrates the locations of a set of places $P = \{p_1, p_2, p_3, p_4\}$. The query q shown in Figure 3.1(a) with $k = 2$ and range $r = 0.15$, has a set of friends $F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$. The number in bracket next to each place is the check-in count made by q's friends. Figure 3.1(b) shows the Euclidean distances and the visitors of each place amongst q's friends. Let us assume $\alpha = 0.5$, the score of the p_1 w.r.t. q is computed as $0.025 + 0 = 0.025$. Similarly, we have $Score(p_2) = 0.185$, $Score(p_3) = 0.205$ and $Score(p_4) = 0.115$. The result of the query q is (p_2, p_3) according to scoring function in equation 3.1.



(a) *Range = shaded area*

P	$\|\|q, p_i\|\|$	$P_{i.visitors}$	*Final Score*
P_1	0.10	-	*0.025*
P_2	0.08	u_1, u_3, u_9	*0.185*
P_3	0.14	u_5, u_6, u_2, u_7	*0.205*
P_4	0.12	u_7, u_{10}	*0.115*

(b) $\alpha = 0.5, k = 2, r = 0.15$
$F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$

Figure 3.1: Top-k Query Example

3.3.2 Framework Overview

The proposed framework consists of three approaches to answer T_kFP query: I) Social-First II) Spatial-First and III) Hybrid. The *Social-First* approach first processes the social component (e.g., friendship relations and their check-ins) and then processes the spatial component (e.g., places in given range) where as *Spatial-First* initially processes the spatial component followed by processing the social component. In contrast, the *Hybrid approach* is capable of processing both social and spatial components simultaneously to answer such queries. More specifically, it leverages two types of pre-processed information associated with each user $u \in U$, her check-in information (check-ins) and summary of her friends' check-ins information.

To the best of our knowledge, there is no unanimously accepted social or spatial storage implementation. Specifically, Facebook uses adjacency lists stored in Memcached [150] which is a distributed memory caching system, on the other hand, Twitter leverages the R*-Tree [151] spatial index. Further, Foursquare uses MongoDB [152], a document oriented database. Similarly, academics research has been adopting various kind of approaches such as [46]

uses adjacency list stored in Neo4j which is a graph based database, where as [35] utilizes relational tables for storing the friendship relations.

Similar to the existing work on KNN queries, we tailored the storage implementation for our technique in a way that suites our requirements. More specifically, we index places and users' check-in information by adopting R-Tree [153] spatial Index structure. Before presenting our technique, we present the definition of *Facility R-Tree, Check-in R-Tree and Friendship Index*.

Facility R-Tree: We create an R-tree where all places ($p \in P$) in the dataset are indexed based on their location coordinates.

Check-in R-Tree: For each user u, we create a *Check-In R-Tree* which indexes all check-ins of the u. This is a 2-dimensional R-tree containing the location coordinates information of each check-in. If a place p is visited by a user multiple times, it will be indexed as many times it was visited hence, *Check-in R-Tree* contains duplicate entries for the place p since many applications (e.g., which include ranking and recommendation of places) do require complete check-in information of users.

Friendship Index: For each user, her friends are indexed using B^+*-Tree* sorted on their IDs. This is used to efficiently retrieve the friends based on their IDs.

3.4 Proposed Technique

3.4.1 Social-First based Approach

Intuitively, *Social-First* approach, first processes the social component of given query q and computes the score of all the places $p \in P$ which have been checked-in by her friends $u \in F_q$. Next, using given range r, it processes the spatial component of the q and computes the score of the remaining places

$p \in P$ which are not checked-in and returns the set of *top-k* places based on their score and q's defined preference criteria α. We next describe the technique in detail with pseudocode given in Algorithm 1.

Initially, in the first loop of the algorithm, we compute the score of each place p in given range r, that has been visited by the friends $u \in F_q$ of query q while maintaining the score of current k_{th} place p based on their social and spatial scores by exploiting the *Check-in R-Tree* of each friend u. In-addition, in the second loop, the *Facility R-Tree* is exploited to compute the score of those places $p \in P$ in range r which are not visited by q's friends hence, their respective score only comprises of spatial component and their $(P_{social}) = 0$ which subsequently yields the *top-k* result set. Let us assume, the score of current k_{th} place p is $Score_k$, next lemma shows that if the $||q, p|| \geq (r - \frac{Score_k}{\alpha})$, we can prune that place p. Next, we introduce our first pruning rule in Lemma 3.1.

Lemma 3.1 *Every place p that has a distance $||q, p||$ from query q greater than current $(r - (Score_k/\alpha))$, cannot be in the Top-k places.*

Proof Given a query user q, a range r, preference factor α, a place p which is not checked-in by any user $u \in F_q$ has social component $(p_{social}) = 0$, by using equation 3.1 and equation 3.2 we get,

$$Score(p) = \alpha(r - ||q, p||) \ + 0 \tag{3.3}$$

To be the candidate for the *Top-k places*, a place p's score must be greater than current $Score_k$, hence

$$Socre_k \leq Socre(p) \tag{3.4}$$

By substituting the value of $Socre(p)$ from eq. 3.3,

$$Socre_k \leq \alpha(r - ||q,p||)$$
$$||q,p|| \leq (r - (Score_k/\alpha)) \tag{3.5}$$

Algorithm 1: Social-First

Input : Query q, range r, weight-age constant α, integer k,
Output: Result set R

```
1  foreach friend u in F_q do
2      Traverse check-in R-Tree of u ; // Accessing Check-in R-Tree by
       branch and bound method
3      foreach place p in r in Check-in R-Tree do
4          update social score and score of p;
5          update Score_k;
6      end
7  end
8  Traverse Facility R-Tree ; // Accessing Facility R-Tree by branch
   and bound method
9  foreach place p in range r in Facility R-Tree do
10     if p.dist ≤ (r − (Score_k/α)) ;          // from Lemma 3.1
11     then
12         compute Score(p);
13         update Score_k;
14     end
15 end
16 return Return R
```

3.4.2 Spatial-First based Approach

Initially, this approach starts with the processing of spatial component of the query q by computing the score of each place p in given range r regardless of the fact whether it is checked-in by any friend $u \in F_q$ of q or not. Moreover, it then computes the social score of each place p by first computing the number of friends checked-in to it by performing an intersection of the set of q's friends F_q and the set of visitors of the place V_p followed by computing the social score

p_{social} of p and then yields the result set. We next elaborate the technique in detail with pseudocode given in Algorithm 2.

Specifically, for each place $p \in P$ in range r, we compute the $Score(p)$ in ascending order of the distance of the place p from q. To achieve this, a $Heap$ is initialized with the root entry of *Facility R-Tree* with $||q, e||$ as a key to process spatial component first. Each entry is iteratively retrieved from $Heap$ and processed as follows. For each place p in range r, we first, compute social component of the score by counting the number of friends $u \in F_q$ who have visited the place p followed by the computation of the final score using equation 3.1. Let us assume, the score of current k_{th} place p is $Score_k$, next lemma shows that if the $||q, p|| \geq (r - \frac{(Score_k - (1-\alpha))}{\alpha})$, the process stops since every subsequent place p entry in $Heap$ is further than the current place p entry from q. Next, we introduce our second pruning rule in Lemma 3.2.

Lemma 3.2 *Every place p that has distance $||q, p||$ from query q greater than current $Score_k$, cannot be in the Top-k places.*

Proof Given a query user q, a range r, preference factor α, to be the candidate for the *Top-k places*, a place p's score must be greater than current $Score_k$, by using equation 3.1 and equation 3.2, we get,

$$Socre_k \leq Socre(p) \tag{3.6}$$

By substituting the value of $Socre(p)$, we get,

$$Socre_k \leq \alpha(r - ||q, p||) \; + \; (1 - \alpha)\left(\frac{F_q \cap V_p}{|F_q|}\right) \tag{3.7}$$

Since the maximum possible social score of given place can be 1, we get

$$
\begin{aligned}
Socre_k &\leq \alpha(r - ||q,p||) \ + \ (1-\alpha) * 1 \\
||q,p|| &\leq (r - \frac{(Socre_k - (1-\alpha))}{\alpha})
\end{aligned}
\tag{3.8}
$$

Algorithm 2: Spatial-First

Input : Query q, range r, weight-age constant α, integer k,
Output: Result set R

```
1  Traverse Facility R-Tree ; // Accessing Facility R-Tree by branch
   and bound method
2  foreach place p in r in Facility R-Tree do
3  |  if ||q,p|| ≥ (r − ((Score_k − (1 − α))/α)) ;      // from Lemma 3.2
4  |  then
5  |  |  return Result set R
6  |  end
7  |  count friends ← F_u ∩ V_p;
8  |  compute Score(p);
9  |  update Score_k ;
10 end
11 return Result set R
```

3.4.3 Hybrid Approach

3.4.3.1 Friends Check-ins R-Tree:

To Optimize, we propose a spatial indexing structure, the *Friends Check-ins R-tree*, that supports the simultaneous pruning of friends and places. It is an R-Tree based structure which is constructed for each user $u \in U$ and is able to prune the search space. *FCR-Tree* stores check-in information of each friend $u \in F_q$ of q, thus representing the check-in summary of all friends of q. The objects of *FCR-Tree* are the root MBRs of each friend's *Check-in R-Tree*. The update of the index in case of new check-in entry of any friend u, is not costly since these are being bulk updated after certain period of time.

Let's assume a user $u \in U$ where the friends of u are $F_u = \{u_1, u_2, u_3 \ldots u_{19}, u_{20}\}$. Figure 3.2 illustrates the idea behind the *FCR-Tree*. Next, we describe our proposed technique in detail.

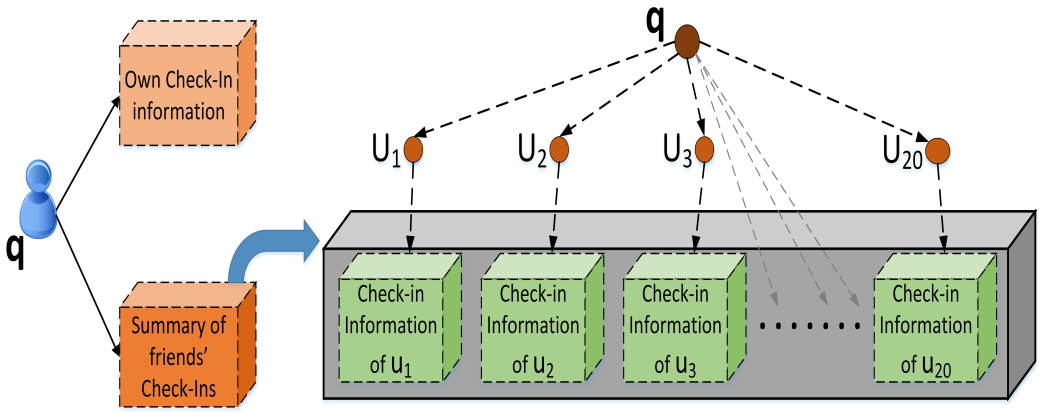


Figure 3.2: Summary of Friends' check-ins

In this approach, the score of each place p in range r is computed by processing the social and spatial components of query q together. To answer the T_kFP queries efficiently, this approach leverages the *Friends Check-ins R-Tree* of query q to prune the friends which have not visited the top-k places and *Grid* Spatial Index to prune the places in given range r which cannot be the candidate for the top-k result set. More specifically, to compute the social and spatial scores of a place p, this approach supports simultaneous pruning of q's friends set F_q and places $p \in P$ in given range r. We next elaborate the technique in detail with pseudocode given in Algorithm 3.

To achieve this, initially, grid portioning approach is employed to divide the area formed by given range r into small cells. Similarly, for each grid cell g_c, a set of places $P_{gc} \in P$ which lie inside the cell is maintained by using the *Facility R-Tree* and distance of the closest place p to q in a cell is recorded as the cell distance from q. In addition, a set of friends who might have visited a

cell denoted as V_{cell} is computed for each cell by exploiting *Friends Check-ins R-Tree* ($FCR-Tree$) of q and counting the number of overlapping objects of $FCR-Tree$ with the cell. Figure 3.3 illustrates an example of a cell and overlapping objects in which four objects are overlapping with the cell and therefore, the overlap count of the cell is 4.

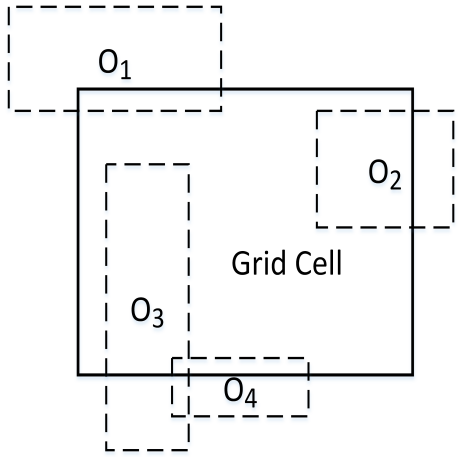


Figure 3.3: Cell Overlap

Once the overlap count and distance to the q for each cell g_c is computed, a ranking score of each cell is computed using equation 3.9 which serves as an upper bound on the score of any place in the cell. Moreover, to compute the score of a place p in range r, the places $p \in P_{gc}$ of the cell g_c with highest score are processed first. If the current k_{th} score is greater than the next cell's score, the process stops since all subsequent cells can not contain a place with higher ranking score.

$$Score_{cell} = \alpha(r - cell.distance) + (1-\alpha)\left(\frac{OverlapCount_{cell}}{|F_q|}\right) \tag{3.9}$$

Algorithm 3: Hybrid Algorithm

Input : Query q, range r, weight-age constant α, integer k,
Output: Result set R

```
1  Traverse Facility R-Tree ;          // Using branch and bound method
2  foreach place p in r in Facility R-Tree do
3  |  compute Score(p);
4  |  insert p in corresponding cell's places set;
5  |  update the distance of corresponding cell;
6  |  update Score_k;
7  end
8  Traverse FCR-Tree of q ;            // Using branch and bound method
9  foreach cell g_c do
10 |  compute overlapCcount(g_c) and Score(g_c);
11 end
12 foreach cell g_c do
13 |  if Score_gc ≤ Score_k then
14 |  |  return Result set R;
15 |  end
16 |  foreach place p in cell g_c do
17 |  |  compute check-in count of p ← V_cell ∩ V_p;
18 |  |  update Score(p) and Score_k;
19 |  end
20 end
21 return Result set R
```

3.5 Experiments

3.5.1 Experimental Setup

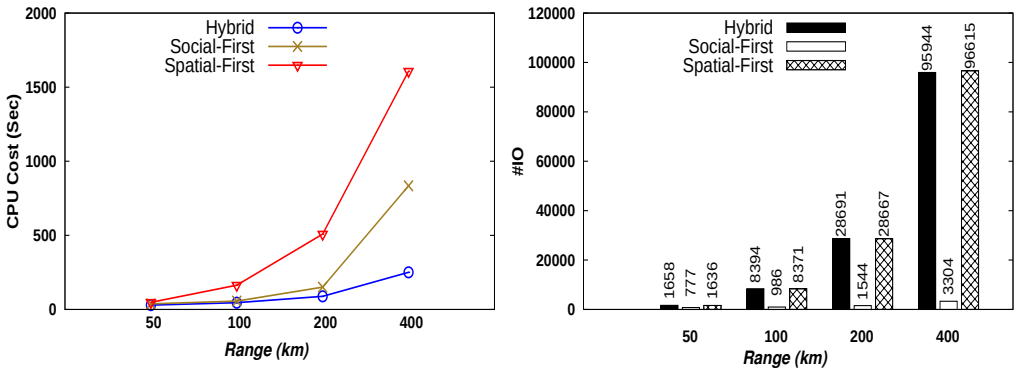
To the best of our knowledge, there is no prior algorithm to solve T_kFP queries therefore, we compare the three proposed algorithms with each other to evaluate their performance.

All algorithms were implemented in C++ and experiments were run on Intel Core $I3$ 2.4GHz PC with 8GB memory running on 64-bit Ubuntu Linux. Specifically, we use same real data set of *Gowalla* as of [45]. *Gowalla* is a location based social network which later was acquired by *Facebook*. It contains

196,591 users, 950,327 friendships, 6,442,890 check-ins and 1,280,956 checked-in places across the world. The page size of each *Facility R-Tree* index is set to 4096 Bytes and 1024 Bytes for *Check-in R-Tree* and *FCR-Tree* indexes. We randomly select 100 users and treat them as query points. The cost in experiments correspond to average cost of 100 queries. The default value of range r is 100 km and the default value of k is set to 10 unless mentioned otherwise.

3.5.2 Performance Evaluation

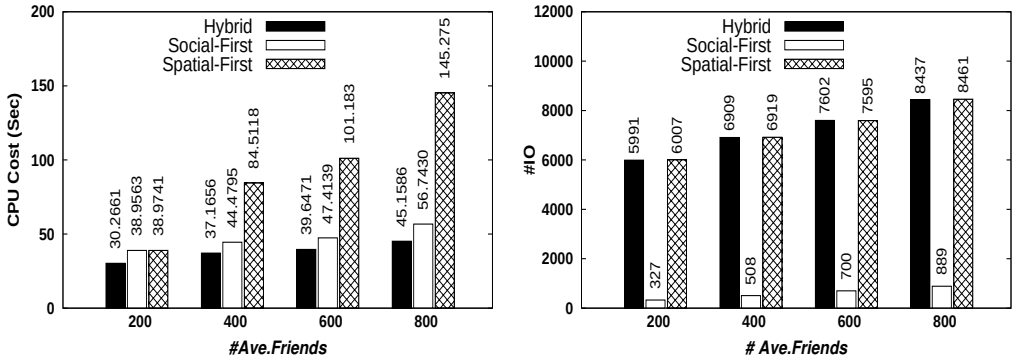
Effect of Range: We analyse the performance of our algorithms for various range values ranging from $10-400$ kms. The size of the area formed by given range determines the number of places it contains (ranging from $1500-94000$). Figure 3.4 shows that *Spatial-First* algorithm is most affected at bigger range values hence its performance deteriorates due to large number of places. Note that, the *Hybrid* algorithm performs better for bigger range since it is more likely to find *top-k* places after processing fewer cells. Figure 3.4(b) shows that I/O cost increases for bigger range values due to large number of places in range which result in higher index access rate.



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

Figure 3.4: Effect of varying range (number of places)

Effect of Average number of Friends: In Figure 3.5, we study the effect of the average number of friends of each query. Note that the size of *FCR-Tree* depends on the size of friends set of each user in data set which essentially affects the Hybrid algorithm. Further, *Spatial-First* algorithm is greatly affected by it because the intersection of two large sets i.e. visitors set and friends set is more expensive. Specifically, Figure 3.5(a), shows the CPU cost and Figure 3.5(b) shows the I/O cost of each method for varying average number of friends. The average number of places in given range r is 38319. Note that when average number of friends increases, the CPU and I/O cost of all three algorithms increases since each friend's check-in information is required to verify candidate places.

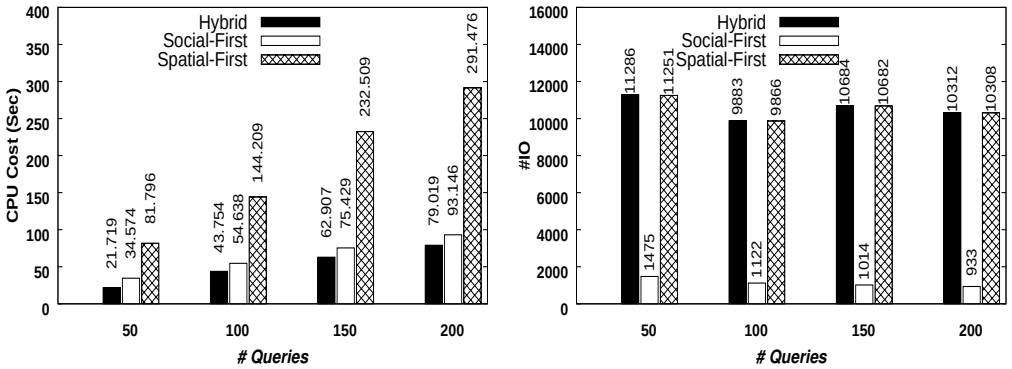


(a) CPU cost (b) I/O cost

Figure 3.5: Performance comparison on different number of friends

Effect of concurrent number of Queries: Geo-Social services seek to answer large number of incoming queries simultaneously due to the enormous size of registered users. Therefore, the number of concurrent queries ranging from 50 to 200 are analyzed for all three algorithm. In addition, each experiment involves average number of friends ranging from $750 - 1350$ and approximately 10,000 average number of places in given range r. All three algorithms need to traverse the *Facility-RTree* every time a T_kFP query is

issued to verify candidate place. The *Social-First* algorithm also needs to traverse the *Check-in R-Tree*. On the other hand, *Hybrid* algorithm leverages the *FCR-Tree* and both *Spatial-First and Hybrid* greatly rely on the visitors set of the places. In Figure 3.6, we report the CPU and I/O cost of each algorithm on *Gowalla* data set for different number of queries. As expected, the I/O cost of *Social-First* algorithm is less than the other two due to low dependency on indexes. Note that *Hybrid* is up to four times better than *Social-First* and *Spatial-First* algorithms.



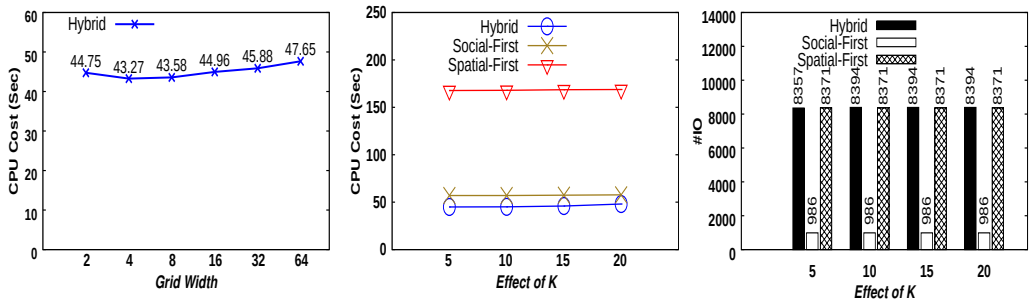
(a) CPU cost (b) I/O cost

Figure 3.6: Effect of number of Queries

Effect of Grid Size: In Figure 3.7(a), we study the effect of the size of grid partitioning ranges from $2 - 64$ on Hybrid algorithm. The size of grid affects the the CPU cost since the size of a cell defines how many places will be processed/pruned at once. Similarly, it also affects the termination condition on the algorithm. Note that the best CPU performance can be achieved by dividing the area into grid of size 4×4.

Effect of k: In previous experiments, the value of k is set to 10. Next, we analyze the performance of three algorithms for various values of k. Note that in Figure 3.7(b), all three algorithms are nearly independent of k. The reason is that, we have to update the result set every time we update the score of a

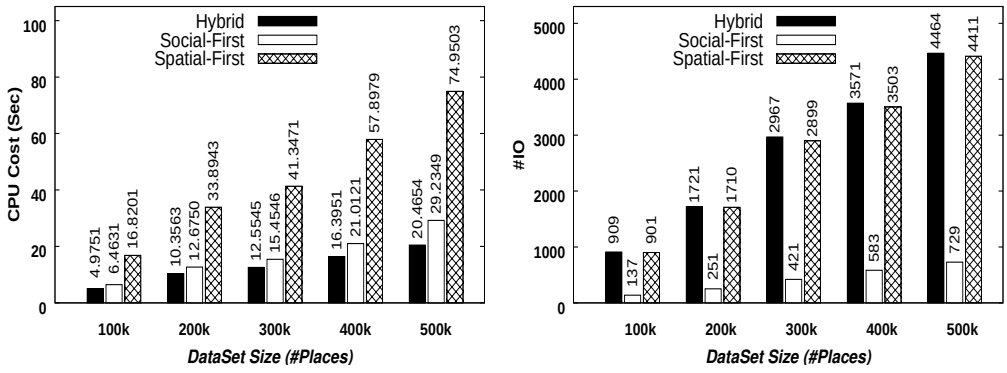
place. Therefore, the size of the result set does not impose great computation load. In terms of I/O cost, Figure 3.7(c) shows that all three algorithms do not get affected by the value of k.



(a) Grid Size (b) CPU cost (c) I/O cost

Figure 3.7: Effect of Grid Size and varying number of requested places (k)

Effect of Data Set Size: In Figure 3.8(a) and 3.8(b), we study the effect of data set size on the performance of the three algorithms. Specifically, we conduct experiments on synthetic data sets of different sizes containing places ranging from 100k to 500k. In Figure 3.8(a), note that the *Spatial-First* algorithm is most effected by number of places. Similarly, in Figure 3.8(b), *Hybrid and Spatial-First* have higher I/O cost due to the intersection performed on visitors set of places and friends set of query q.



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

Figure 3.8: Effect of varying data set sizes (number of places)

3.6 A Demo for Location-Based Top-k Queries in Social Networks

In this section, we develop a demonstration to show the real application of Location-Based Top-k Queries on Social Networks. This demonstration enables participants to view the actual output of the T_kFP query containing the *top-k* places checked-in by friends in given region.

This demonstration enables participants to view the actual output of the *TGS* query containing the *top-k* POIs checked-in by friends in given region. In addition to this, other information related to top-k POIs such as name, address, visitors count, friends detail can also be viewed using Google Maps in browser-based interface. On the server side, we process the TGS query to efficiently retrieve the top-k POIs. Further, communication between server and client is handled using standard HTTP post operations.

3.6.1 Framework and Query Processing Overview

Our framework prototype adopts client-server model in which users submit their queries through a web browser (client), and queries are then sent to the server. For each *TGS* query, the *top-k* POIs are computed and are sent back to the user. Figure 3.9 illustrates the architecture of our framework.

3.6.1.1 Client Side

The client side provides users with a mechanism to interact with the server side through a browser-based interface for submitting queries and to view the retrieved POIs. The client side component provides interaction with the map through Google Maps API. When a *TGS* query is submitted, the users

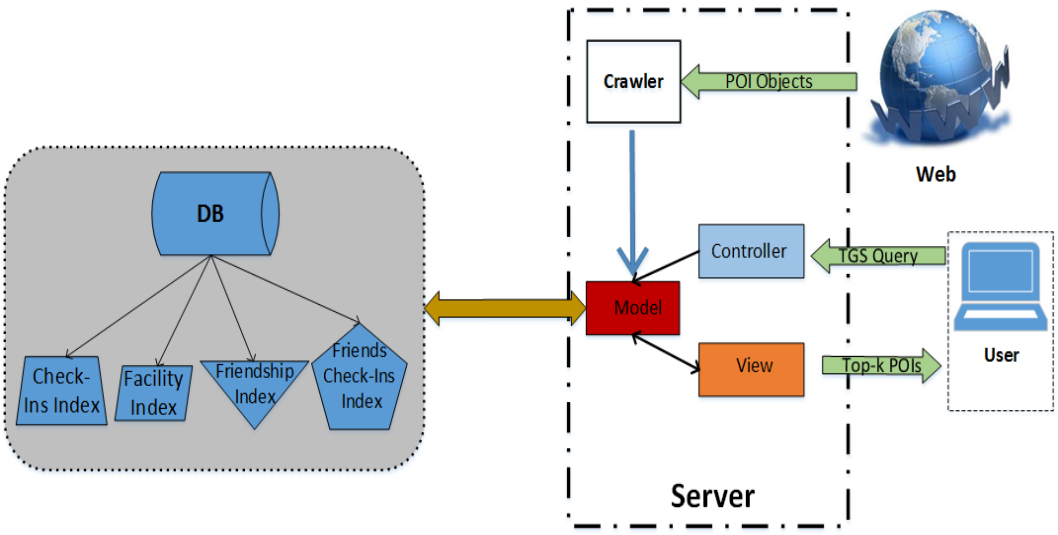


Figure 3.9: Framework Architecture

specify their *user id*, a location, a range (radius) and number of desired POIs k. Consequently, queries are sent to the server by *HTTP* post operation. After a query is processed at the server side, the retrieved top-k POIs are returned and displayed using Google Maps in the client side browser.

3.6.1.2 Server Side

Overview:

The web server is built using JSP and Apache Tomcat by applying MVC (model, view, controller) architecture. The MVC is divided into business logic and view logic. When a query is received by JSP server (Controller), it is forwarded to *Model* where the query processing algorithm implemented in Java is invoked to retrieved the result set which is then, forwarded to *View* to be sent to the client side browser.

Query Processing:

On server side, the *TGS* query can be answered by employing any of the three proposed approaches. Simply stated, *Social-First* approach first retrieves social

information (set of friends) of the query q and then for each friend, the score of all checked-in POIs (spatial component) in given range is computed and updated where as, *Spatial-First* approach starts with the processing of spatial component of the query q by retrieving all POIs in given range regardless of the fact whether these are checked-in by any friend of q or not. Moreover, it then computes the social score of each POI by computing the number of friends checked-in to it by performing an intersection of the set of q's friends and the set of visitors of the POI. In addition, *Hybrid* approach is capable of computing the score of each POI in range by processing social and spatial components of query q together.

In other words, It prunes the friends (which have not visited the top-k POIs) and the POIs in given range (which cannot be the candidate for the top-k) simultaneously. Specifically, the algorithm indexes each user's *check-in summary* using well-known spatial data structure i.e., *R-tree* [153] along with storing each user's friends' *check-in* summary in a separate index structure. Similarly, grid partitioning approach is employed to divide the region formed by given range r into small cells which facilitates in identifying those cells (including the POIs lie inside) that do not overlap with the objects of *friend's check-in summary R-tree* hence are pruned along with the objects (i.e., friends) which do not overlap with cells without further processing.

3.6.2 Demonstration Details

In this demonstration, participants will be able to experience how the system can be used for issuing *TGS* queries to retrieve *Top-k* POIs in friends circle. The client-side (browser) interfaces are shown in Figure 3.10 and 3.11.

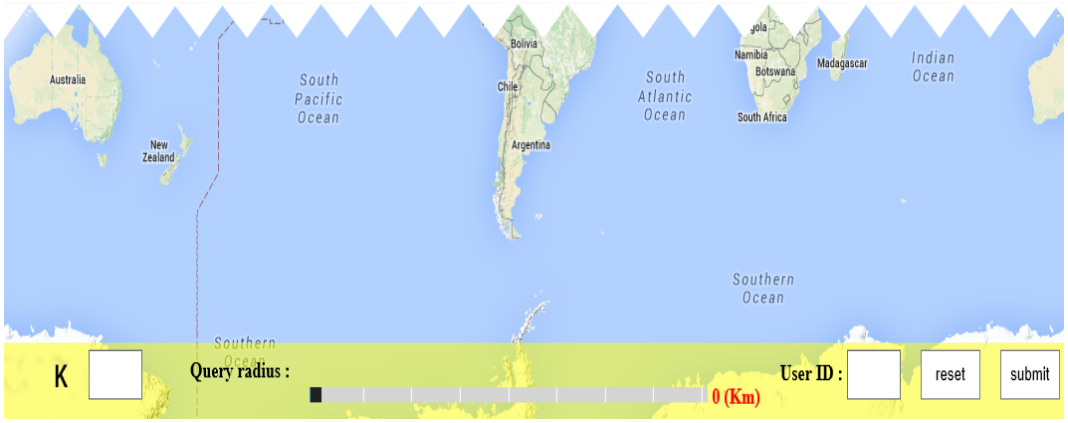


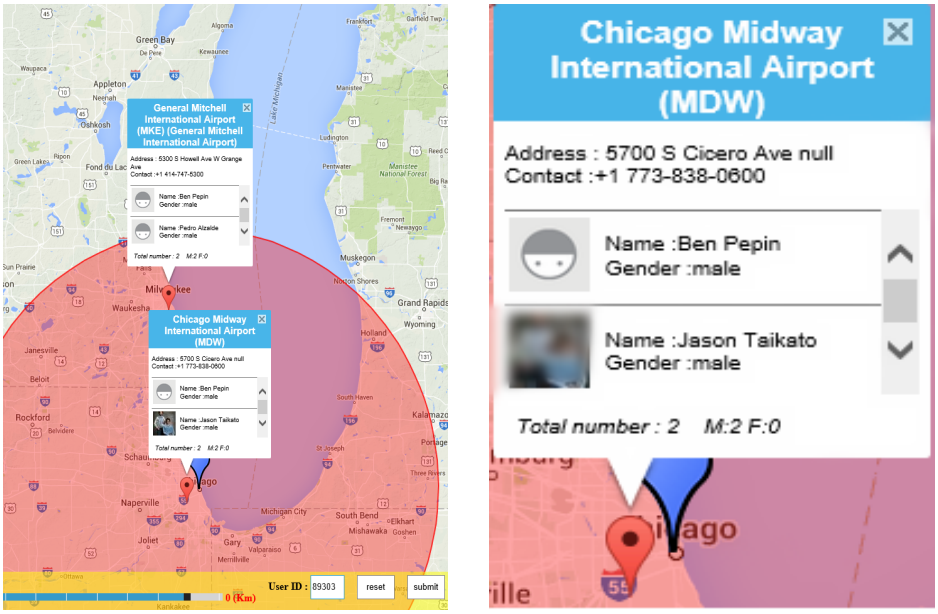
Figure 3.10: Main Interface

3.6.2.1 Submitting a TGS Query:

Initially, users specify a query User id. Using the *Foursquare API*, the framework focuses on that users hometown. Subsequently, users specify their location by clicking on Google Maps (the latitude and longitude of the selected location is obtained using Google Maps API), radius of the query region using the slide bar and number of desired POIs they are interested in to find.

3.6.2.2 Dataset:

We use real-world foursquare [154] data set for the demonstration. The data set contains 3473835 friendship relations, 33,278,683 check-ins, 266,909 users and 3,680,126 POIs. Each POI is represented by a unique id and has longitude and latitude information associated with it. The POI's id is used to retrieved additional information such as POI name and address. Similarly, each check-in entry has user id, longitude and latitude information of checked-in POI and time stamp at which it was issued.



(a) Query Result (b) POI Detail

Figure 3.11: T_kFP Query Result

3.7 Conclusions

In this chapter, we study location-based top-k queries in social networks and formalize a new problem namely, *Top-k famous places* T_kFP query which enriches the semantics of the conventional spatial query by introducing a social relevance component. A T_kFP query retrieves top-k places (points of interest) ranked according to their spatial and social relevance to the query user. We propose three approaches namely, 1) Social-First 2) Spatial-First and 3) Hybrid to efficiently process such queries. Further, an *FCR-Tree* index structure is proposed that integrates social information with spatial information. Results of empirical studies with an implementation demonstrate the effectiveness of the proposed approaches using real and synthetic datasets. In addition to this, we develop a demonstration of the proposed query which enables participants to view the actual output of the T_kFP query containing top-k places checked-in by the friends of the query user in given range.

Chapter 4

Spatial Skyline Queries in Social Networks

In this chapter, to complement our *Top-k famous places* query, we extend our work to study skyline queries that do not require a scoring function to retrieve desired objects within given range r (i.e., $||q, p|| < r$) that are not dominated by any other object. The outline of the chapter is as follows: Section 4.1 contains introduction of the chapter, Section 4.2 lists down the contributions we have made, Section 4.3 explains the problem definition, Section 4.4 describes the proposed techniques, Section 4.5 consists of experimental evaluation and Section 4.6 concludes the chapter.

4.1 Introduction

In *Top-k* queries, a user needs to define a scoring function that combines social and spatial scores to rank objects which may not be trivial (e.g., due to incompatible attributes, different distributions of attributes, the inability of users to choose a good scoring function) [26]. Motivated by this, to complement

our *Top-k famous places* T_kFP query presented in previous chapter, we study *skyline queries* that do not require a scoring function to retrieve desired objects and we formalize a new query called, *Socio-Spacial Skyline Query*.
A *Socio-Spacial Skyline Query* $SSSQ$ returns every place for which there does not exist any other place that has a better social score and better spatial score. In $SSSQ$ query, we do not need to have a scoring function therefore, skyline queries are natural and popular choice for the applications involving multi-criteria decision making [116, 119, 120, 155, 156, 157, 117].

Let us take an example of a German tourist visiting Melbourne who is looking for some restaurants that are close and are also popular among German people. An $SSSQ$ returns every restaurant p for which there does not exist any other restaurant p' that is closer to her location and is more popular among German people. A formal definition is provided in Section 4.3.
We present three approaches to solve $SSSQ$ queries namely, 1) Social-First, 2) Spatial-First and 3) Hybrid. The first two approaches separately process the social and spatial components of the queries and do not require a specialized index. However, the third approach (*Hybrid*) is capable of processing social and spatial components simultaneously.

4.2 Contributions

We make the following contributions in this chapter.

1. We extend our work to propose $SSSQ$ queries that return places which are not dominated by any other place.

2. We present a framework consisting of three approaches to solve $SSSQ$ queries, namely Social-First, Spatial-First and Hybrid. The first two approaches separately process the social and spatial components of the

queries and do not require a specialized index. However, the third approach (*Hybrid*) is capable of processing social and spatial components simultaneously by utilizing a hybrid index. For efficient retrieval and filtering of objects, we partition our skyline work space into a grid and map the objects in it using their social and spatial score. To further improve the pruning phase, we map each grid cell to skyline workspace grid as a 2-dimensional point and filter dominated ones along with all the objects that lie inside them.

3. We conduct an extensive evaluation of the proposed schemes using real and synthetic datasets and demonstrate the effectiveness of the proposed approaches and compare their performance with [27]. Our results show that our algorithms are significantly better than the competitor.

4.3 Problem Definition

In this work, we are interested in retrieving places in given range r based on their social and spatial scores. The *top-k* query uses a scoring function that combines social and spatial scores to rank the objects. However, users must have adequate domain knowledge to be able to decide upon a good value of α. In partucular, it is not easy to define a scoring function (e.g., due to incompatible attributes, different distributions of attributes, the inability of users to choose a good scoring function) [26].

Therefore, to complement our *Top-k famous places* query, we extend our work to study skyline queries which return the objects that are within the range r (i.e., $||q, p|| < r$) and are not dominated by any other object. In order to answer these queries, we compute p_{social} and $p_{spatial}$ scores of each place as defined in previous section (3.3.1). The intuition behind using a range r is

that sometimes users are not interested in places that are too far. Below we formally define our query.

Dominance. A place p is dominated by a place p' if $p_{social} \leq p'_{social}$ and $p_{spatial} \leq p'_{spatial}$ and for atleast one of the following two holds: $p_{social} < p'_{social}$ and $p_{spatial} < p'_{spatial}$. We denote the dominance relationship as $p' \prec p$ which implies that place p is dominated by place p'.

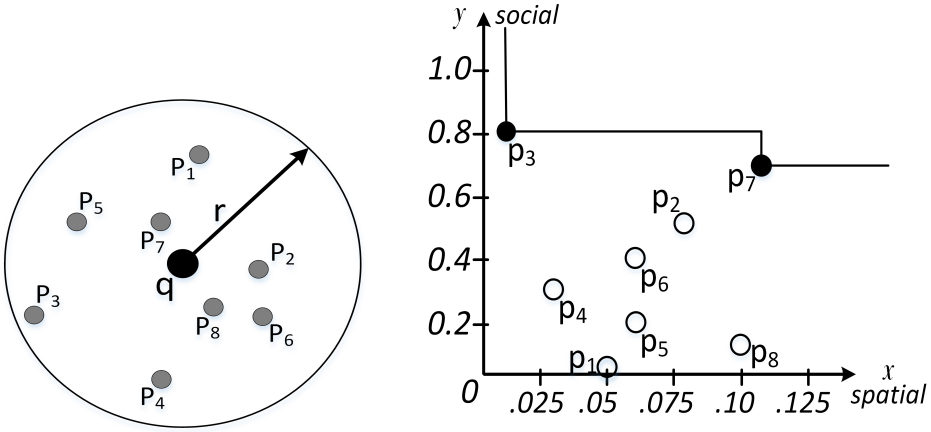
Socio-Spatial Skyline Query ($SSSQ$): Given a query q and a range r, an $SSSQ$ returns every place p for which $||q, p|| < r$ and p is not dominated by any other place p'.

We use an example given in Table 4.1 and in Figure 4.1 to illustrate the problem definition. Let us assume that we have a set of places $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ inside range $r = 0.15$, given by query q and a set of friends of q, $F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$. For each place, we compute its spatial and social scores based on its distance from query q and number of friends of q who checked-in at this place. Next, we map each place to a space where x-coordinate refers to spatial score and y-coordinate refers to social score as illustrated in Figure 4.1(b).

For Example in Table 4.1, the social and spatial scores of place p_3 are 0.8 and 0.01 respectively and using these scores, p_3 is mapped to the space as shown in Figure 4.1(b). To retrieve query result, we utilize this space to find such places that are not dominated by any other place. For example, place p_7 dominates p_5 because $p_7.social > p_5.social$ and $p_7.spatial > p_5.spatial$. Hence, the $SSSQ$ query returns p_7 and p_3 which are not dominated by any other place.

Id	dist(q,p)	p.spatial	Visitors	p.social
P_1	0.10	0.05	-	0
P_2	0.07	0.08	u_1, u_3, u_5 u_7, u_9	0.5
P_3	0.14	0.01	u_1, u_2, u_4 u_5, u_7, u_8 u_9, u_{10}	0.8
P_4	0.12	0.03	u_4, u_6, u_8	0.3
P_5	0.09	0.06	u_4, u_5	0.2
P_6	0.09	0.06	u_3, u_4, u_7, u_9	0.4
P_7	0.04	0.11	u_1, u_2, u_4, u_5 u_8, u_9, u_{10}	0.7
P_8	0.05	0.10	u_4	0.1

Table 4.1: Sample Dataset



(a) Places in Range (b) Mapping to 2D-Space

Figure 4.1: Mapping

4.4 Proposed Techniques

4.4.1 Social-First Based Algorithm

Social-First based approach accesses only those places that are visited by q's friends rather than accessing each place in range r. This approach first looks at the check-ins of each friend to compute social score of each visited place

$p \in P$ in given range r. Then we only use the visited places in the range to compute skyline places as described in algorithm 4.

Initially, in the first loop of the algorithm (at line 1), it exploits *Check-in R-Tree* of each friend $u \in F_q$ of query q to get the places in range r followed by computing social score of each candidate place p in r (at line 3). In-addition to this, we also compute spatial score of each candidate place p. Next, each candidate place p is accessed in descending order of the sum of two scores (at line 5) because accessing the places in this order guarantees that a place is skyline if and only if it is not dominated by any place in S [120], where S is the set of skyline places obtained so far. Then each candidate place p is examined for the dominance (at line 6). Finally, the *nearest neighbour* of query q is computed (at line 8) and is added to the skyline places if it is not checked-in by her friends. Below lemma 4.1 shows that *nearest neighbour* of query q is always a skyline object.

Lemma 4.1 *A nearest neighbour (NN) of the query is always a skyline place.*

Proof There cannot be any place p' that has a smaller distance than the nearest neighbour p of the query q. If there are more than one nearest neighbours, then the nearest neighbour with highest social score is not dominated by any other nearest neighbour and is considered as a skyline place.

4.4.2 Spatial-First Based Algorithm

This approach first gets all places in range r by issuing a range query on *Facility R-Tree* (at line 1) in algorithm 5. Then, in the first loop of the algorithm (at line 2), it computes spatial and social scores of each place in given range r.

Algorithm 4: Skyline: Social-First Algorithm

Input : Query q, Range r
Output: Skyline Result set S

```
1  foreach friend u of q do
2  |  Issue a range query on Check-in R-Tree;
3  |  foreach place p in range r do
4  |  |  update social score of p and add it to candidate places;
5  |  end
6  end
7  foreach candidate place p in descending order of p_social + p_spatial do
8  |  if place p is not dominated by any place p' in S then
9  |  |  insert p into skyline result set S
10 end
11 Compute NN and insert into skyline result set S;
12 return Result set S
```

Next, each place in range is accessed in descending order of the sum of two scores (at line 5) and then is examined for the dominance (at line 6). If the place is not dominated by skyline places obtained so far, it is inserted into the skyline places set S.

Spatial-First approach accesses only one R-tree index (i.e., *Facility R-Tree*) while *Social-First* approach has to access as many R-tree indices as the number of friends of query q. On Contrary, the down side of *Spatial-First* approach is that it retrieves all the places in given range r and computes their social score while *Social-First* approach computes the social score of only those places in the range r that are visited by q's friends. Next, we address the weakness of both in below section.

4.4.3 Hybrid Algorithm

This section focuses on our third approach (i.e., Hybrid) to process $SSSQ$ which is capable of processing both social and spatial aspects simultaneously. Before presenting the technique, first we describe our index and record keeping

Algorithm 5: Skyline: Spatial-First Algorithm

Input : Query q, range r
Output: Result set S

```
1  Issue a range query on facility R-Tree;
2  foreach place p in range r do
3  |  Compute spatial and social score;
4  |  Insert p into candidate places;
5  end
6  foreach candidate place p in descending order of p_social + p_spatial do
7  |  if place p is not dominated by any place p' in S then
8  |  |  insert p into skyline result set S;
9  |  end
10 end
11 return Result set S
```

structures.

4.4.3.1 Two Grids

We first introduce two grid indices that are employed to speed-up retrieval and pruning process.

1. Range Grid: This grid is built upon the region formed by given range r by splitting it into small cells as shown in Figure 4.2(a). Each cell has following information associated with it:

- Places that lie in the cell.
- Number of overlapping *Friends' Check-In R-tree* (FCR-Tree) object rectangles (root MBR of Check-In R-Trees) with that particular cell. As stated in Section 3.4.3.1, this information is used to compute a bound on the social scores of the places in the cell.

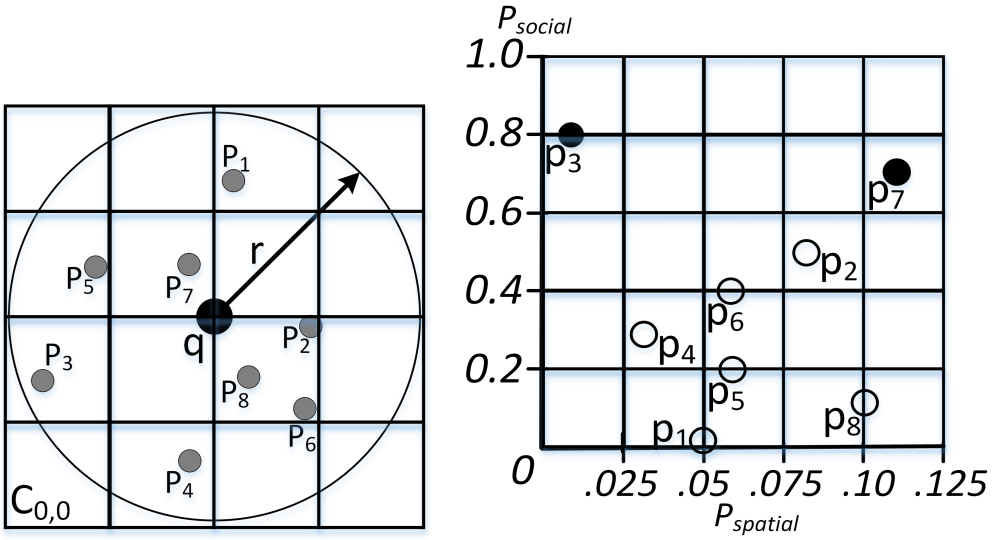
2. Skyline Workspace Grid: As described earlier, for each place inside range r, we compute social ($Psocial$) and spatial ($Pspatial$) scores and then

maps the place to a 2-dimensional space where *Psocial* is mapped along y-axis and *Pspatial* is mapped along x-axis. This 2-dimensional space is called *skyline workspace*.

Similar to Range Grid, we divide our skyline workspace into a grid as shown in Figure 4.2(b) to index each object based on its social and spatial scores. This aids in retrieving, examining dominance and filtering objects efficiently.

Please note that to avoid disambiguity, we denote a cell of range grid as a **cell** and a cell of skyline workspace grid as a **block** in rest of the chapter.

Figure 4.2(b) shows the mapping of all places in the range to their corresponding skyline workspace grid blocks based on their social and spatial scores. For example, denoting *bottom-left* block of the skyline workspace grid b_{ij} (where i is a row number staring with 0 and j is a column number starting with 0) as $b_{0,0}$, place p_3 is mapped to block $b_{3,0}$ and place p_7 is mapped to block $b_{3,4}$.

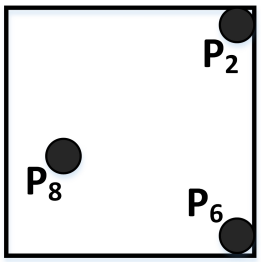


(a) Places in Range (b) Mapping to Skyline workspace

Figure 4.2: Sample Dataset and Skyline Mapping

4.4.3.2 Mapping Range Grid Cell

In-addition to the mapping of places to skyline grid blocks, each range grid cell C_{ij} is mapped to skyline workspace grid. To achieve this, first we compute social and spatial scores (i.e., c_{social}, $c_{spatial}$) of each cell. To understand further, let us take an example of range grid cell $C_{1,2}$ with three places (i.e., p_2, p_6, p_8) inside it as shown in Figure 4.3(a) with their social and spatial scores listed in Figure 4.3(b). Assuming, the cell $C_{1,2}$ overlaps with six objects of *FCR-Tree* that is considered as social score ($c_{social} = 0.6$) of the cell. In-addition, the spatial score of place P_8 that lies in the cell is largest among all and is considered as the cell's spatial score ($c_{spatial} = 0.10$). Therefore, these scores serve as an upper bound on scores of any place inside the cell and by using these scores, the cell is mapped to its corresponding skyline workspace gird block $b_{2,3}$ as a point $C_{1,2}(c_{social}, c_{spatial})$ as shown in Figure 4.4.



(a) Cell $C_{1,2}$

Id	P_{social}	$P_{spatial}$
P_2	0.5	0.08
P_6	0.4	0.06
P_8	0.1	0.10

(b) Places' scores

Figure 4.3: Social and Spatial score of a Range Grid Cell

Based on this, we can conclude that the cell point $C_{1,2}(0.10, 0.6)$ in skyline workspace clearly dominates all the places (i.e., p_2, p_6, p_8) that lie in it. In contrast, if cell $C_{1,2}$ in skyline workspace is dominated by any other object (e.g., p_7), the cell is immediately pruned along with all the places inside it due to having smaller social and spatial scores than the cell's. Therefore, the places lie inside the cell cannot be the part of skyline places hence, this pruning considerably improves the query processing time.

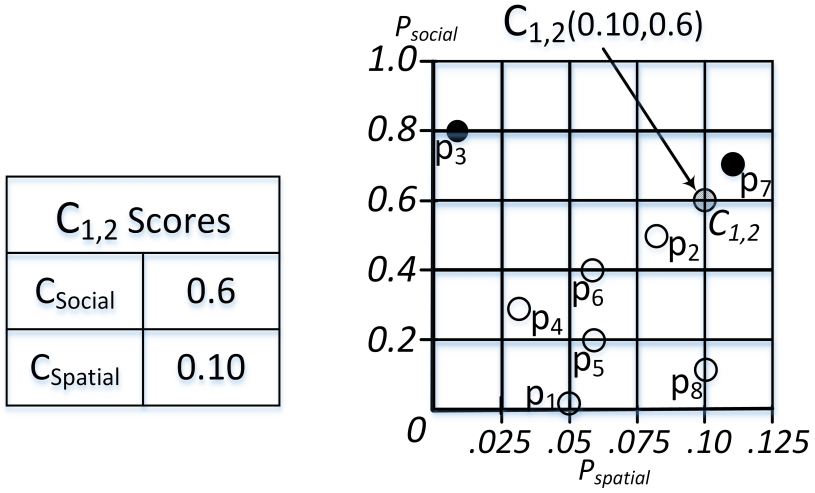


Figure 4.4: Range cell mapping to skyline workspace grid

Algorithm 6 describes the indexing of range and skyline workspace along with mapping of each range grid cell C_{ij} to the skyline workspace grid. Initially, we start by constructing a grid index in region formed by range r (at line 1). Then in the first loop (at line 3), we index each place in range to its corresponding range grid cell C along with updating the cell's spatial score ($c_{spatial}$)(at line 4). In-addition, the skyline workspace is divided into a grid (at line 5) and a range query is issued on *FCR-Tree* to get all the friends of query q who might have visited any place in the range (at line 6). Finally, in second loop (at line 7), for each range grid cell C, the upper bound (c_{social}) on social score of the places that lie in the cell is computed and then the cell is mapped to its corresponding skyline workspace block b using the cell's social and spatial scores.

Each block b_{ij} of skyline workspace grid is associated with two types of lists, one of which contains range grid cell objects that lie inside and the second one contains actual places p inside that block as shown in Figure 4.5.

Algorithm 6: Indexing and Mapping

Input : Query q, range r
Output: Range and Skyline Grids

```
1  Construct a range grid defined by range r;
2  Issue a range query on facility R-Tree;
3  foreach place p in r do
4  |  map it to corresponding cell C_ij and update spatial score of cell;
5  end
6  Construct skyline workspace grid;
7  Issue a range query on FCR-Tree to get q's friends;
8  foreach cell C_ij do
9  |  compute social score;
10 |  Map C_ij to corresponding block b_ij;
11 end
```

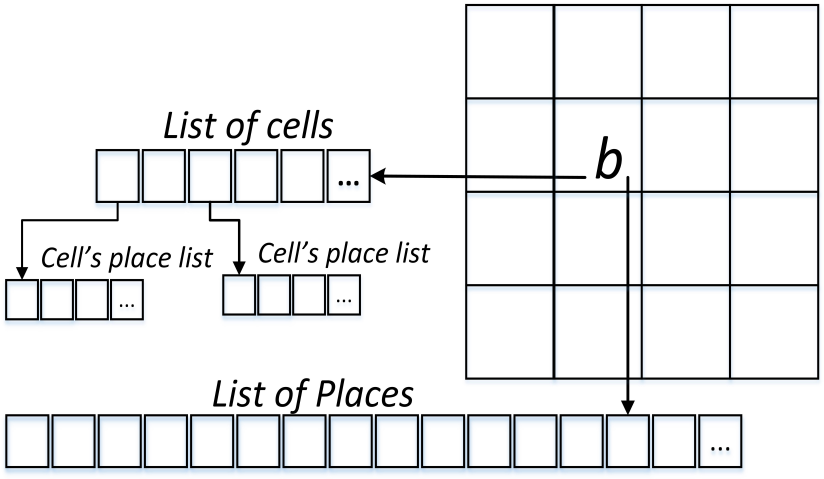


Figure 4.5: Grid Index and Record-keeping Structures

4.4.3.3 Computation Module

Intuition:

Let us assume that we have a place p in skyline workspace grid as illustrated in Figure 4.6. Note that the block $b_{2,2}$ is dominated by place p therefore, no place or range grid cell in the block can contain a skyline place p. Similarly, all the blocks in the shaded area R do not need to be accessed if we have already seen the place p.

To make blocks access efficient, we need to access them in a particular order where order is determined by *maxScore* which is defined in definition 4.1. For each block, we en-heap the blocks below and towards left of it. If a block is dominated, it is pruned.

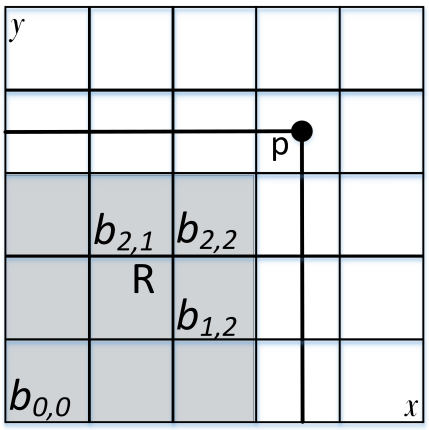


Figure 4.6: Dominated Blocks

Definition 4.1 ***maxScore(b)***

maxScore(b) *of any given block of skyline workspace grid is a summation of its top-right corner coordinates (i.e.,* P_{social}*,* $P_{spatial}$*).*

For example example in Figure 4.7, the *maxScore* of block $b_{4,4}$ is 2 (sum

of top-right corner coordinates i.e., 1,1) and the *maxScore* of $b_{3,3}$ is 0.8+0.8 = 1.6. Since the top-right corner of skyline workspace $P(1,1)$ has the highest *maxScore*, the block $b_{4,4}$ is selected first for processing. Now Consider two points p_1 and p_2 at the low-right and at the top-left corner of $b_{4,4}$ respectively. Note that points p_1 and p_2 have higher $maxScore$ than any object in shaded region. Precisely, for every unprocessed block b_{ij}, $maxScore(b_{ij}) \leq max(maxScore(b_{3,4}), maxScore(b_{4,3}))$. Consequently, the block to be processed after $b_{4,4}$ is either $b_{3,4}$ or $b_{4,3}$ and let us assume that $maxScore(b_{4,3} \leq maxScore(b_{3,4})$, $b_{3,4}$ is the second one to be processed. Further, the blocks with the third highest $maxScore$ is determined among $b_{4,3}$, $b_{3,3}$ and $b_{2,4}$. We next describe the technique in detail with pseudocode given in Algorithm 7.

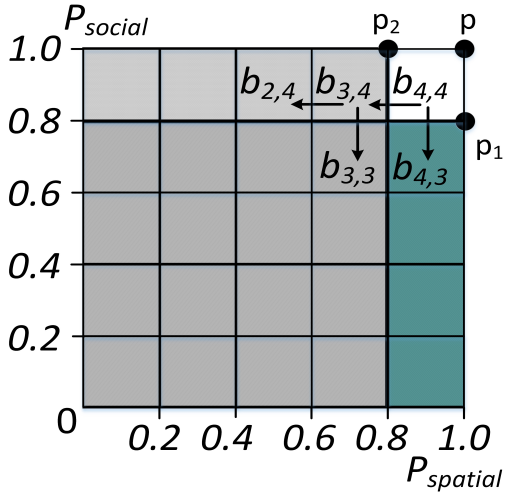


Figure 4.7: Block Visiting Order

Algorithm:

Initially, algorithm 7 invokes *Indexing and Mapping algorithm* (algorithm 6) to compute social and spatial scores of each range grid cell and to index them to their corresponding skyline workspace grid blocks (at line 1). Further, the

algorithm employs the method described above to process blocks in descending $maxScore(b)$ order, retaining the property of visiting the minimal set of blocks. To handle this, a *maxHeap* is initialized with the top-right block $b_{4,4}$ of skyline workspace grid with its $maxScore$ as a sorting key (at line 2).

Algorithm 7: Skyline: Hybrid

```
   Input  : query q, range r, grid size g
   Output: Result set S
 1 Invoke Indexing and Mapping(q, r);
 2 Insert top-right block b in Heap;
 3 while Heap is not empty do
 4 |   de-heap block b_ij;
 5 |   if b_ij is not dominated then
 6 |   |   foreach cell point c in b_ij do
 7 |   |   |   if c is not dominated then
 8 |   |   |   |   foreach place p in c do
 9 |   |   |   |   |   Compute social score;
10 |   |   |   |   |   insert it into respective block b_ij if it is not dominated;
11 |   |   |   |   end
12 |   |   |   end
13 |   |   end
14 |   |   foreach place p in b_ij in descending order of p_social + p_spatial do
15 |   |   |   if p is not dominated then
16 |   |   |   |   Insert in skyline result set S
17 |   |   |   end
18 |   |   end
19 |   |   if Adjacent blocks of b_ij are not dominated then
20 |   |   |   en-heap blocks;
21 |   |   end
22 |   end
23 end
24 return Result set S
```

Then, algorithm starts de-heaping the blocks iteratively (at line 3) and if a block is dominated by any skyline place p (at line 5), it is immediately pruned and consequently, the blocks below and left of it are not en-heaped. Since at this stage, only range grid cells are indexed to the skyline workspace, it first examines each cell object C_{ij} for dominance which lie inside the de-heaped

block (at line 6) and if a cell is dominated by any already found skyline place so far (at line 7), it is pruned. Consequently, all the places that lie in the pruned cell object are also discarded and are not processed further.

Further, if a cell object C_{ij} is not dominated, then for each place that lie in it (loop at line 8), the algorithm computes social score (P_{social}) of it (at line 9). Then, the place is indexed to its corresponding skyline workspace grid block b provided that the place is not dominated by any skyline place found so far (at line 10). However, if a corresponding block is dominated, it is marked to avoid being en-heaped in *maxHeap*. Subsequently, after indexing each place in the range to its corresponding blocks, the algorithm starts examining each place p that lie in the current de-heaped block for dominance (at line 12) in descending order of $P_{social} + P_{spatial}$ and updates the skyline result set. The algorithm also en-heaps the blocks below and to the left of current de-heaped block using their $maxScores$ provided that neither of them have been en-heaped before nor are they dominated by any skyline place found so far (at line 14). The algorithm terminates when all the blocks in *maxHeap* are examined and returns the skyline places (at line 16).

4.5 Experiments

4.5.1 Experimental Setup

In existing work, the techniques are either not applicable or cannot be efficiently extended to solve $SSSQ$ queries. However, although the paper [27] studies a different problem, we have implemented their algorithm and compared our techniques against it.

All the algorithms are implemented using the exact experimental settings as described earlier in Section 3.5.1. Table 4.3 contains summary of the

datasets used in experimental evaluation. Gowalla dataset characteristics are already described in Section 3.5.1. In addition to this, we derived five synthetic datasets from Gowalla which consist of places ranging from $100,000$ to $500,000$ and their corresponding number of visitors, check-ins and friendship relations among the chosen visitors. The page size of each *Facility R-Tree* index is set to 4096 Bytes and 1024 Bytes for *Check-in R-Tree* and *FCR-Tree* indexes. For each experiment, we randomly select 100 users and treat them as query points. The cost in the experiments correspond to the average cost of 100 queries. The default value of range r is 100 km and the default value of k is set to 10 unless mentioned otherwise.

Parameters	**Values**
Number of Queries	50, **100**, 150, 200
Range (km)	50, **100**, 200, 400
Dataset Size (Places in thousands)	100, 200, 300, 400, 500, **1300**
Grid Size	2, **4**, 8, 16, 32, 64
Average Friends	200, 400, **600**, 800
k	5, **10**, 15, 20

Table 4.2: Parameters (Default shown in bold)

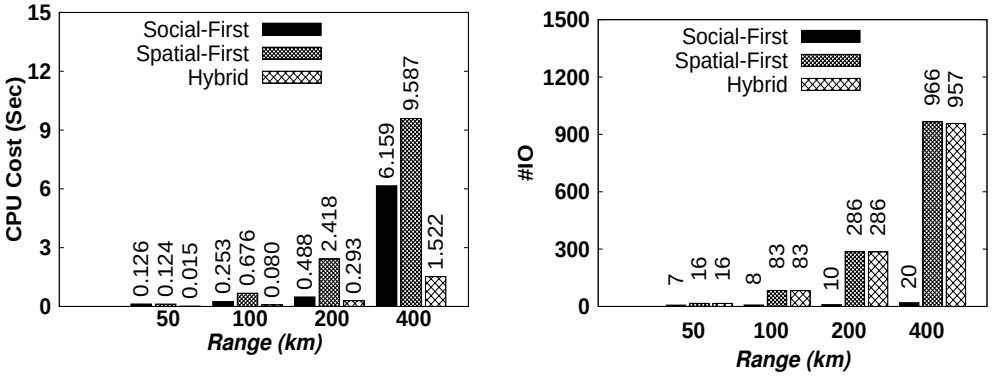
DataSet	**Places**	**Users**	**Friendships**	**Check-Ins**
Gowalla	1,280,956	196,591	950,327	6,442,890
Synthetic 01	100,000	17,162	77,254	503,000
Synthetic 02	200,000	39,223	152,957	1,100,698
Synthetic 03	300,000	61,394	241,369	1,830,235
Synthetic 04	400,000	86,687	301,887	2,536,897
Synthetic 05	500,000	103,856	395,745	3,258,659

Table 4.3: Datasets Characteristics

4.5.2 Performance Evaluation

Effect of Range: We analyse the performance of our algorithms for various range values ranging from $100 - 400$ kms. The size of the area formed by given

range determines the number of places it contains (ranging from $1500-94000$). Figure 4.8 shows that *Spatial-First* algorithm is most affected at bigger range values due to more number of places to be processed hence, it's performance deteriorates. Similarly, *Social-First* approach does not get affected much by the range because it only takes into account the visited places by query q's friends. Note that, the *Hybrid* algorithm performs better for bigger range since it is more likely to find *skyline* places by processing fewer blocks and by pruning more cells including their corresponding places which lie in them simultaneously. Figure 4.8(b) shows that I/O cost increases for bigger range values due to large number of places in given range and large number of visitors (in *spatial-first* approach) which results in higher index access rate.

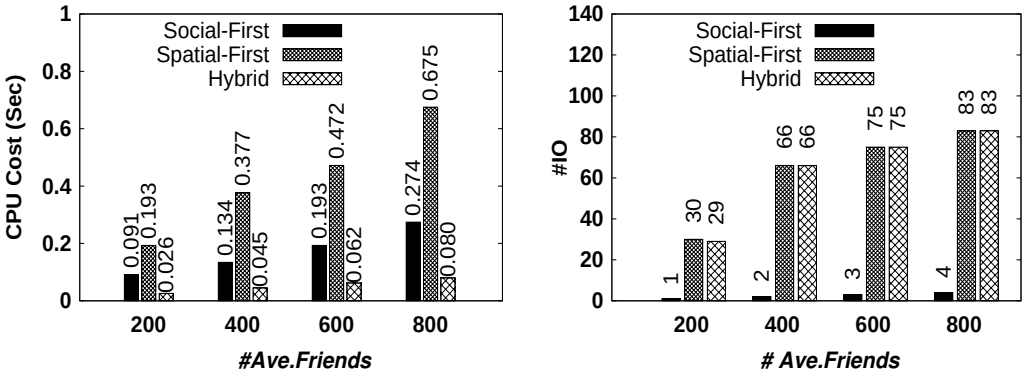


(a) CPU cost (b) I/O cost

Figure 4.8: Effect of varying range (number of places)

Effect of Average number of Friends: In Figure 4.9, we study the effect of the average number of friends of each query. Note that the size of *FCR-Tree* depends on the number of friends of each user in the dataset which essentially affects the Hybrid algorithm to some extent. Further, *Spatial-First* algorithm is greatly affected by it because the intersection of two large sets i.e., visitors set of each place in range and friends set of query q is more expensive. Similarly, *Social-First* algorithm is greatly affected by the number of friends

since it has to process more *Check-in* R-trees. Specifically, Figure 4.9(a), shows the CPU cost and Figure 4.9(b) shows the I/O cost of each method for varying average number of friends. The average number of places in given range r is 38319. Note that when average number of friends increases, the CPU and I/O cost of all three algorithms increases since each friend's check-in information is required to verify the candidate places.

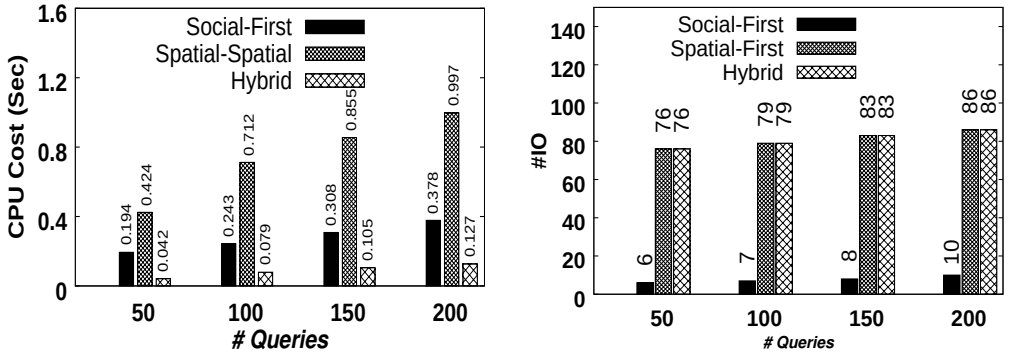


(a) CPU cost (b) I/O cost

Figure 4.9: Performance comparison on different number of friends

Effect of concurrent number of Queries: The number of concurrent queries ranging from 50 to 200 are analysed for all three algorithm. In addition, each experiment involves average number of friends ranging from $200 - 800$ and approximately 10,000 average number of places in given range r. All three algorithms need to traverse the *Facility-RTree* every time an $SSSQ$ is issued to retrieve candidate places. In-addition, *Social-First* algorithm also traverses the *Check-in R-Tree* that belongs to each friend and as we increase the number of queries, the number of friends to be processed, also increase. Therefore, *Social-First* algorithm exhibits more CPU cost for large number of queries. On the other hand, *Hybrid* algorithm leverages the *FCR-Tree* and both *Spatial-First and Hybrid* greatly rely on the size of visitors set of the places. In Figure 4.10, we report the CPU and I/O cost of each algorithm on

Gowalla data set for different number of queries. As expected, the I/O cost of *Social-First* algorithm is less than the other two due to low dependency on indexes. Note that *Hybrid* is up to eight times better than *Social-First* and *Spatial-First* algorithms.



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

Figure 4.10: Effect of number of Queries

Effect of Grid Size: In Figure 4.11, we study the effect of the size of grid partitioning ranges from $2-16$ on Hybrid algorithm. For region grid, the size of grid affects the the CPU cost since the size of a cell defines how many places will be processed/pruned simultaneously. Similarly, it also affects the termination condition on the algorithm. Note that the best CPU performance can be achieved by dividing the area into grid of size 4×4. In-addition, the Skyline workspace is also partitioned into 4×4 grid because algorithm achieves optimal performance at this granularity.

Effect of Dataset Size: In Figure 4.12(a) and 4.12(b), we study the effect of data set size on the performance of the three algorithms. Specifically, we conduct experiments on synthetic data sets of different sizes containing places ranging from 100k to 500k. In Figure 4.12(a), note that the *Spatial-First* algorithm is most effected by number of places. Similarly, in Figure 4.12(b), *Hybrid and Spatial-First* have higher I/O cost due to the intersection

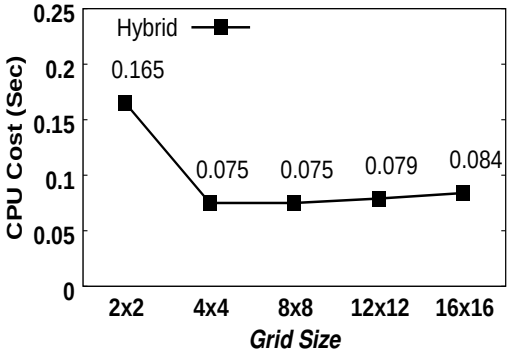
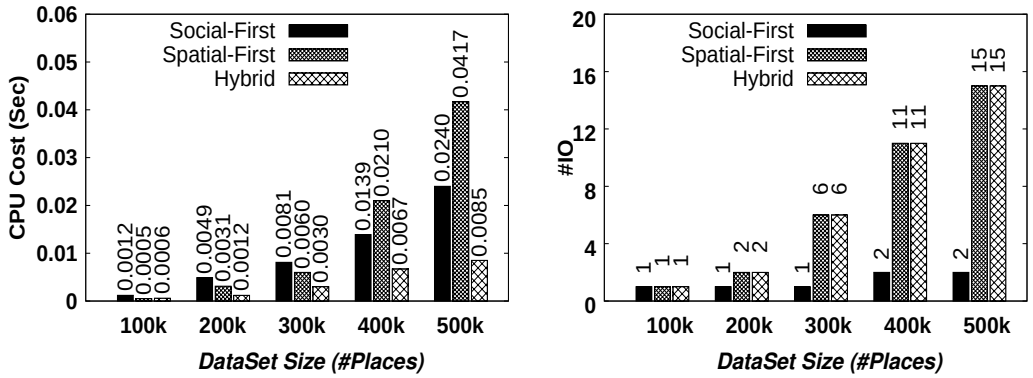


Figure 4.11: Effect of Grid Size

performed on visitors set of each place and friends set of query q.



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

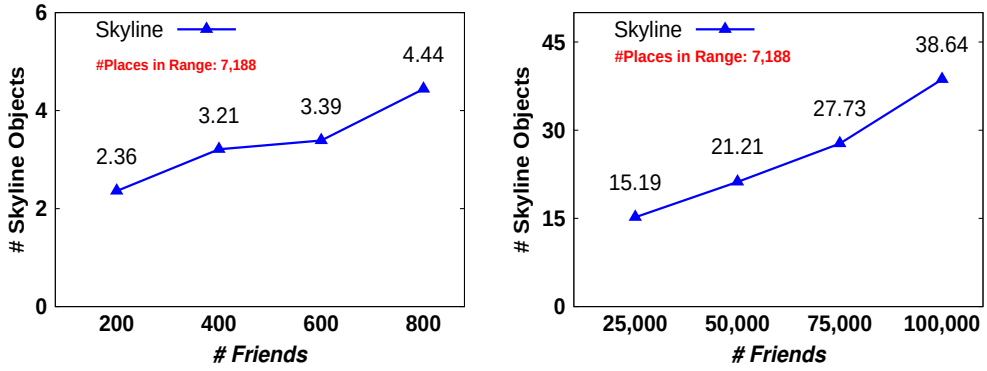
Figure 4.12: Effect of varying dataset sizes

4.5.3 Analysis of Results Quality

Top-k queries and skyline queries both have been extensively studied in the past. The advantage of a top-k query is that the number of objects to be returned is controlled by the user (by giving a value of k). However, the top-k query assumes that the user is able to define a suitable scoring function (e.g., a suitable value of α). This may be challenging because the user may not be able to choose a suitable scoring function mainly because of the incompatibility of the attributes involved in top-k queries and their distributions [26]. A skyline

query addresses this problem and does not require a scoring function to be defined. However, the user cannot control the number of objects returned by the query and, in the worst case, the number of skyline objects may be equal to the total number of objects in the data set. Therefore, top-k queries and skyline queries complement each other. In this section, we analyse the size of socio-spatial skyline queries and compare the results returned by top-k queries and skyline queries.

Size of Skyline: In Figure 4.13, we run 100 skyline queries for each setting and report the average size of skyline. Figure 4.13(a) shows that the average size of skyline is 2 to 5 as we vary the average number of friends for the query user. Note that, on average, the total number of places in the query range is more than $7,000$ and skyline shortlists up to 5 places, on average, that dominate all other places in terms of both spatial score and social score. One reason for such a small skyline size is that the data is sparse and there may not be many check-ins in the given range by all of the query users' friends and, as a result, the social score for most of the places may be zero.



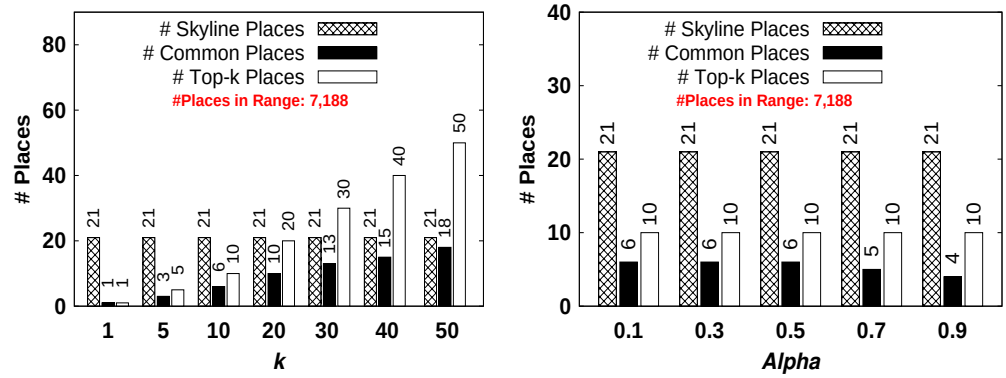
(a) Small number of average friends (b) Large number of average friends

Figure 4.13: Effect of number of friends

In Figure 4.13(b), we evaluate the size of skyline for a more challenging case where the average number of friends for the query user is varied from $25,000$

to 100,000. We remark that this is a realistic setting and many users may have such a large number of friends, e.g., query user is a page "Germany" and its friends represent the people who were born in Germany. Figure 4.13(b) shows that the size of skyline increases with the average number of friends but the size is still much smaller compared to the total number of places in the range. This shows that the skyline query studied in this paper is useful and returns only a small number of objects to the user. In the rest of the experiments, we choose 50,000 as default for the average number of friends of the query user.

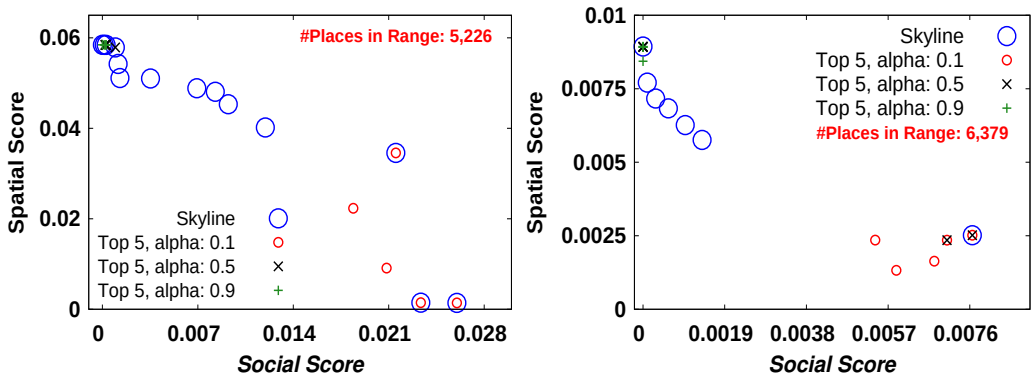
Results Returned by Top-k vs Skyline: In this section, we compare and analyse the results returned by skyline queries and top-k queries. In Figure 4.14, we run 100 queries for each setting and report the average number of result objects returned by top-k queries, skyline queries and the average number of objects that are returned by both of the queries (shown as "# Common Places"). Specifically, Figure 4.14(a) studies the effect of k and Figure 4.14(b) compares skyline and top-10 queries for varying α. Figure 4.14 demonstrates that the results returned by both top-k and skyline queries share many objects but, at the same time, each query reports several places that the other query fails to return. This shows that the two queries complement each other.



(a) Effect of k (b) Effect of α

Figure 4.14: # common places returned by both queries

In Figure 4.15, we further analyze the results returned by the two types of queries. Specifically, the result places are mapped to a two dimensional space where x-axis corresponds to their social scores and y-axis corresponds to their spatial scores. In Figure 4.15(a), the skyline query returns 15 places. The top-5 query with $\alpha = 0.1$ (high preference for social score) returns the places shown with small red circles. Three of these top-5 places are the skyline points and the other two places are not the skyline points because they are dominated by other places. For the top-5 queries with $\alpha = 0.5$ (equal preference for both social and spatial scores) and $\alpha = 0.9$ (high preference for spatial score), the top-5 places are the places on the top-left of the figure (having high spatial scores but low social scores). Figure 4.15(b) shows similar results except that some of the top-5 places for $\alpha = 0.5$ (equal preference) are the places in bottom-right of the figure and some are in the top-left of the figure.



(a) Skyline vs Top-k for user 1

(b) Skyline vs Top-k for user 2

Figure 4.15: Analysis of results

Figure 4.15 shows that the top-k queries may sometimes fail to capture the users' requirements, e.g., for example, by choosing $\alpha = 0.5$, a user may have wanted to obtain the places that have reasonably high values on both social and spatial scores but the results may contain places with either high social scores but very low spatial scores or high spatial scores but very low social

scores (as in Figure 4.15). The skyline query addresses this problem to some extent and gives a better coverage of the results. However, it fails to capture the requirements of users who have chosen α to be too high or too low. For example, in Figure 4.15(b), the skyline contains only one object that has a high social score, therefore, it would fail to capture the requirements of a user who prefers social score much more than the spatial score (e.g., $\alpha = 0.1$) and wants to obtain several places with high social scores. In contrast, the top-5 query with $\alpha = 0.1$ returns 5 objects each having a high social score. Also, as pointed out earlier, the number of skyline objects may be arbitrarily large and the user may not be able to control the number of objects returned.

4.6 Conclusions

In this chapter, we extend our work to complement our work in chapter one and propose another query called, *Socio-Spacial Skyline Query SSSQ*. We present three approaches to process the query called, 1) Social-First, 2) Spatial-First and 3) Hybrid. The first two approaches separately process the social and spatial components of the query and do not require a specialized index. The third approach (*Hybrid*) is capable of processing social and spatial components simultaneously by utilizing a hybrid index specifically designed to handle T_kFP and *SSSQ* queries. For efficient retrieval and filtering of objects, we partition our skyline work space into a grid and map the objects in it using their social and spatial score. To further improve the pruning phase, we map each grid cell to skyline workspace grid as a 2-dimensional point and filter dominated ones along with all the objects that lie inside them. We conduct an extensive evaluation of the proposed schemes using real and synthetic datasets and demonstrate the effectiveness of the proposed approaches and compare their

performance with [27]. Our results show that our algorithms are significantly better than the competitor.

Chapter 5

Spatial Group Top-k Queries in Social Networks

In this chapter, we study a problem of finding top-k places considering their distance from the group of query users Q and popularity of the place among each query user $q_i \in Q$'s social connections (e.g., the number of check-ins at the place by each q's friends). We formalize this problem as a *Geo-Social Group preference Top-k* ($SG\text{-}Top_k$) query and propose efficient query processing techniques. The outline of the chapter is as follows: Section 5.1 introduces the research problem and contains the contributions we have made, Section 5.2 explains the problem definition, Section 5.3 describes the proposed techniques, Section 5.4 consists of experimental evaluation and Section 5.5 concludes the chapter.

5.1 Introduction

In many applications, a group of users may want to plan an activity to find a point of interest (POI) for example, some conference attendees would like

to go out for dinner together. For this purpose, we consider their respective locations and social circles to recommend required POIs. In this chapter, we study a problem of finding top-k places considering their distance from the group of query users Q and popularity of the place among each query user $q_i \in Q$'s social connections (e.g., the number of check-ins at the place by each q's friends).

Consider an example of a group of tourists visiting Melbourne. The group consists of tourists from various countries e.g., conference attendees from Italy, Germany and Denmark. They may want to find a nearby pub which is popular (e.g., frequently visited) among people from their respective countries. This involves utilizing spatial information (i.e., near by pub, check-ins) as well as social information (i.e., people who were `born-in` Italy, Germany and Denmark).

Although several types of queries have been investigated on LBSNs [124, 31, 158], to the best of our knowledge, none of the existing methods can be applied to answer the queries like the above that aim at finding near by places that are popular in social circles of the query users satisfying social and spatial constraints. Motivated by this, in this chapter, we formalize this problem as a *Geo-Social Group preference Top-k* ($SG\text{-}Top_k$) query and propose efficient query processing techniques. Specifically, a $SG\text{-}Top_k$ query retrieves top-k places (points of interest) ranked according to their spatial and social relevance to the group of query users where the spatial relevance is based on how close the place is to the location of each group member and the social relevance is based on how frequently it is visited by the one-hop neighbors of each query user $q_i \in Q$. definition is provided in Section 5.2.1.

Firstly, we present *Branch-and-Bound* approach to solve our problem. Then, we propose some optimization techniques to further improve its performance.

Our experimental study shows that our optimized algorithm outperforms the other one.

5.1.1 Contributions

The contributions made in this chapter are listed below.

1. To the best of our knowledge, we are the first to study the $SG\text{-}Top_k$ query that retrieves near by places popular among a particular group of users w.r.t. each query user $q_i \in Q$ in the social network.

2. We present *Branch-and-Bound* algorithm followed by some optimization techniques to further improve the algorithm. The *Branch-and-Bound* approach uses R-tree to process places in ascending order of their aggregate distance from each query user $q_i \in Q$ and then computes social score of each place p to finally compute aggregate score i.e., $aggScore(p)$. Since computation of $aggScore(p)$ of each $p \in P$ is computationally expensive, our *optimized* approach leverages the *FCR-Tree* to offer efficient pruning techniques to prune irrelevant place. Moreover, the optimized algorithm computes a region based on current found $top - k_{th}$ aggregate score to quickly check the entries that can be pruned.

3. We conduct an exhaustive evaluation of the proposed schemes using real dataset and test them on various aggregate functions e.g., *min, max, avg.* Our results demonstrate the effectiveness of the proposed schemes.

5.2 Preliminaries

5.2.1 Problem Definition

The score of a place is defined in a similar way as in Chapter 3. We briefly describe it here again.

Score of a place p [24, 28]: Given a query user q_i, the score of a place $p \in P$ is the weighted sum of its spatial score (denoted as $spatial(p, q_i)$) and its social score (denoted as $social(p, q_i)$).

$$Score(p, q_i) = \alpha \times spatial(p, q_i) + (1 - \alpha) \times social(p, q_i) \tag{5.1}$$

Let V_p denotes the set of all users that visited (i.e., checked-in at) the place. The social score $social(p, q_i)$ of place p is computed as follows:

$$social(p, q_i) \ = 1 - \frac{|F_{qi} \cap V_p|}{|F_{qi}|} \tag{5.2}$$

The spatial score $spatial(p, q_i)$ is based on how close the place is to the query user $q_i \in Q$. Formally, $spatial(p, q_i) = ||p, q_i||$, where $||p, q_i||$ indicates Euclidean distance between the query user and p. Note that $social(p, q_i)$ is always between 0 to 1 and the smaller social score is considered better. In addition, we also normalize $spatial(p, q_i)$ such that it is also between 0 to 1, e.g., the data space is normalized such that $||p, q_i|| \leq 1$.

Aggregate Score of a place p: Given a set of query users $Q = \{q_1, q_2, ...q_n\}$, the aggregate score of a place p (denoted as $aggScore(p)$) is computed using a monotonic scoring function f which takes as input each $Score(p, q_i)$ for every $q_i \in Q$.

$$aggScore(p) = f(Score(p, q_i), ..., Score(p, q_n)) \tag{5.3}$$

P	$\|\|p, q_i\|\|$	$Social(p, q_i)$	$Score(p, q_i)$	$aggScore(p)$
p_1	$q_1 = 0.10$ $q_2 = 0.14$ $q_3 = 0.12$	$q_1 = 0.6$ $q_2 = 0.4$ $q_3 = 0.8$	$q_1 = 0.35$ $q_2 = 0.27$ $q_3 = 0.46$	$Avg = 0.36$ $min = 0.27$ $max = 0.46$
p_2	$q_1 = 0.09$ $q_2 = 0.05$ $q_3 = 0.13$	$q_1 = 0.8$ $q_2 = 0.6$ $q_3 = 0.4$	$q_1 = 0.445$ $q_2 = 0.325$ $q_3 = 0.265$	$Avg = 0.345$ $min = 0.265$ $max = 0.445$
p_3	$q_1 = 0.22$ $q_2 = 0.20$ $q_3 = 0.15$	$q_1 = 1.0$ $q_2 = 0.6$ $q_3 = 0.8$	$q_1 = 0.61$ $q_2 = 0.40$ $q_3 = 0.475$	$Avg = 0.495$ $min = 0.40$ $max = 0.61$
p_4	$q_1 = 0.20$ $q_2 = 0.17$ $q_3 = 0.10$	$q_1 = 0.2$ $q_2 = 0.4$ $q_3 = 0.4$	$q_1 = 0.20$ $q_2 = 0.285$ $q_3 = 0.25$	$Avg = 0.245$ $min = 0.20$ $max = 0.285$
p_5	$q_1 = 0.17$ $q_2 = 0.12$ $q_3 = 0.09$	$q_1 = 0.6$ $q_2 = 0.8$ $q_3 = 0.4$	$q_1 = 0.385$ $q_2 = 0.46$ $q_3 = 0.245$	$Avg = 0.363$ $min = 0.245$ $max = 0.46$

Table 5.1: Sample Dataset and Aggregate Scores

For example, if f is *average*, the aggregate score corresponds to $\sum_{i=1}^{i=n} Score(p, q_i)/n$. Similarly, if f is *min*, the aggregate score corresponds to minimum of $Score(p, q_i)$ for $q_i \in Q$.

Geo-Social Group preference Top-k ($SG\text{-}Top_k$) Query: Given a set of places $P = \{p_1, p_2, ...p_n\}$ in a LBSN, a $SG\text{-}Top_k$ query Q returns k places with smallest aggregate score $aggScore(p)$. The $aggScore(p)$ of each place $p \in P$ is computed as described above depending on the function f used. Some examples of the function f are *min, max and avg.* For instance, if f corresponds to *min*, the aggregate score will be $aggScore(p) = min(Score(p, q_i), ..., Score(p, q_n))$ that is, $aggScore(p) = \underset{i:1\ to\ n}{\arg\min}\ Score(p, q_i)\ \forall\ q_i \in Q$. As an example, consider a dataset containing a set of places i.e., $P = \{p_1, p_2, p_3, p_4, p_5\}$ and Q is a set of query users i.e., $Q = \{q_1, q_2, q_3\}$. Table 5.1 illustrates spatial score, social score, and score for each place p for each query user q_i and it also shows the aggregate score.

If f corresponds to avg, $\alpha = 0.5$ and $k = 2$, the corresponding $SG\text{-}Top_2$ query reports places (p_4 and p_2) that minimizes the average aggregate score

$aggScore(p)$. Similarly, if f corresponds to max, the $SG\text{-}Top_2$ query reports places (p_4 and p_2) that minimizes the maximum aggregate score ($aggScore(p)$). On the other hand, if f corresponds to min, the $SG\text{-}Top_2$ query reports places (p_4 and p_5) that minimizes the minimum aggregate score ($aggScore(p)$).

5.3 Techniques Overview

Before presenting our approaches to solve $SG\text{-}Top_k$ query in detail, first we provide a brief overview of the approaches and to the best of our knowledge, there does not exist any technique in literature that can be adopted to answer the proposed query. First we present *Branch-and-Bound* approach and then we present optimization techniques to further improve its performance. The *Branch-and-Bound* approach uses R-tree to process places in ascending order of their aggregate distance from each query user $q_i \in Q$ and then computes social score of each place p to finally compute aggregate score i.e., $aggScore(p)$. Since computation of $aggScore(p)$ of each $p \in P$ is computationally expensive, our Optimized approach leverages the *FCR-Tree* (as defined in section 3.4.3.1) to offer efficient pruning techniques to prune such candidate places that cannot be a part of top-k places.

5.3.1 Branch-and-Bound (B&B) Algorithm

Before presenting our *Optimized* approach, we first discuss *B&B* approach to process $SG\text{-}Top_k$ query. The B&B approach is to traverse *Facility R-Tree* in *best-first* manner. For this purpose, we use a *min-heap* where the key for each entry E is $minAggDist(Q, E)$. To compute $minAggDist(Q, E)$, the algorithm computes minimum distance of E from each query user $q_i \in Q$ and then according to the aggregate function f provided, the algorithm finalises the value of $minAggDist(Q, E)$. For example, if $f = min$, the $minAggDist(Q, E)$

is the smallest minimum distance between E and any of the q_i. Then, we initialize *min-heap* with the root of *Facility R-Tree*.

Further, the algorithm starts de-heaping entries in ascending $minAggDist(Q, E)$ order. If a de-heaped entry E is a node, it computes its lower-bound aggregate score (denoted as $LBaggScore(E)$) based on its $minAggDist(Q, E)$ only, assuming that its social score i.e., $social(E)$ is minimum. Further, if the de-heaped entry E is an object (a place p), it computes its exact aggregate score $aggScore(p)$ and updates *top-k* places. Finally, at any point, if an entry E having lower-bound aggregate score worse than current k_{th} best place score (denoted as $aggScore_k$) is de-heaped, the algorithm terminates. The reason is, every subsequent entry E in *min-heap* will have worse aggregate score than the current $aggScore_k$.

5.3.2 Optimized Algorithm

This section focuses on our *Optimized* approach to process $SG\text{-}Top_k$ queries and before presenting the technique in detail, first we describe a specialized index specifically designed for the technique.

5.3.2.1 Computation Module:

In B&B algorithm, since lower-bound of *Facility R-tree* nodes is computed based only on $minAggDist(Q, E)$, it is loose which results in high computation cost. To develop an efficient approach to computing solution for $SG\text{-}Top_k$ query, we propose few improvements in the algorithm. First, we present highlights of the improvements we made.

1. First we compute tighter lower-bound on aggregate score of a node entry

E i.e., $LBaggScore(E)$ using better estimate of its social score. For this purpose, the algorithm exploits *FCR-Tree* of each query user $q_i \in Q$.

2. Then, the algorithm computes a region (denoted as region bounded rectangle, RBR) based on current $aggScore_k$ to quickly check the entries that can be pruned.

3. Finally, while en-heaping an entry E, we make an observation to avoid computing its lower-bound on social score by associating its parent node's lower-bound on social score with it as an initial estimate.

The details of the above mentioned improvements are provided next.

1. Computing Lower bound on Aggregate score:

Recall that in B&B approach, to estimate best possible aggregate score $LBaggScore(E)$ of an entry E, we assume that its social score i.e., $social(E)$ is maximum and therefore, its lower-bound is loose. To overcome this limitation, the *Optimized* algorithm leverages *Friends Check-ins R-Tree* (FCR-Tree) to estimate social score of the E (denoted as $LBSocial(E)$) which in return, tightens the lower-bound on aggregate score $LBaggScore(E)$.

Specifically, to compute $LBSocial(E)$ of an entry E of *Facility R-tree*, the algorithm traverses specialized index i.e., *FCR-Tree* of a query user q_i to compute number of its objects (root MBRs of Check-In R-trees of friends) intersecting with E. Let's consider an example in Figure 5.1 where we have a *Facility R-tree* entry E and some *FCR-Tree* objects belonging to q_i's friends ranging from u_1 *to* u_5. Since only u_1, u_4 *and* u_5 overlap with E, they might have checked-in at any place p in E. Therefore, the maximum number of friends who might have visited a place in E is 3, which can be used to obtain the

lower-bound on social score. Let's denote the number of overlapping objects as $numOverlap$, the lower-bound on social score is computes as $1 - \frac{numOverlap}{|F_{qi}|}$.

In addition, once the lower-bound on social score against a query user q_i is computed, the lower-bound on score (denoted as $LBScore(E, q_i)$) is computed against the query user q_i using equation 5.4. The pseudocode of computing $LBScore(E, q_i)$ is given in Algorithm 8.

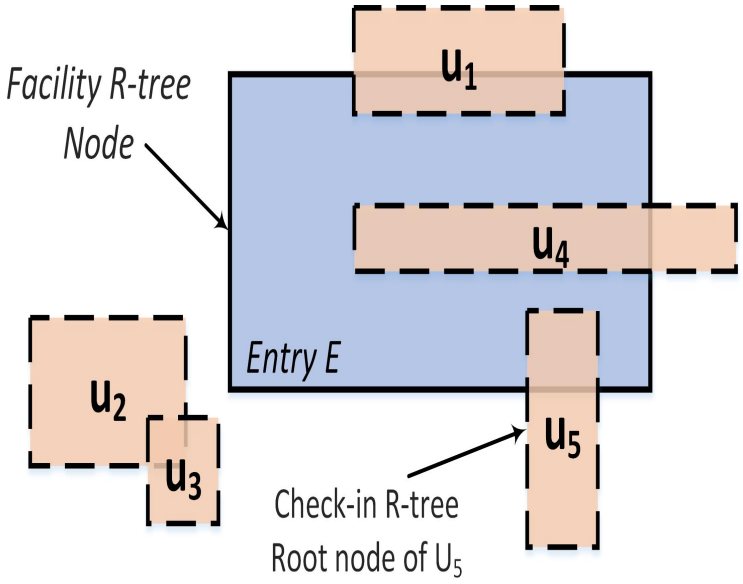


Figure 5.1: MBR Social Score Bound

$$LBScore(E, q_i) = \alpha \times minDist(E, q_i) + (1 - \alpha) \times \left(1 - \frac{numOverlap}{|F_{qi}|}\right) \quad (5.4)$$

2. Optimal MBR based Search Regions:

Recall that in B&B, we need to compute $minDist(E, q_i)$ n times to compute $minDist(E, Q)$. For this purpose, we require to compute minimum distance

Algorithm 8: Get-$LBaggScore(mbr, Q)$

```
1 numOverlap = ∅;
2 foreach user q_i ∈ Q do
3     Issue a range query on FCR-Tree;
4     numOverlap = Compute number of objects overlapping with the
      mbr;
5     LBSocial(mbr, q_i) = 1 − numOverlap / |F_qi|;
6     LBScore(mbr, q_i) =
      α × minDist(mbr, q_i) + (1 − α) × LBSocial(mbr, q_i) ;
7 end
8 LBaggScore(mbr) = f(LBScore(mbr, q_i), ..., LBScore(mbr, q_n));
9 return LBaggScore(mbr)
```

from the entry E to each q_i and if this distance is greater than $\frac{aggScore_k}{\alpha}$, we can ignore E. However, this requires computing $minDist(E, q_i)$ n times. To overcome this problem, we create a *Region Bounding Rectangle* (denoted as *RBR*) such that if an entry E does not overlap with it, it is pruned.

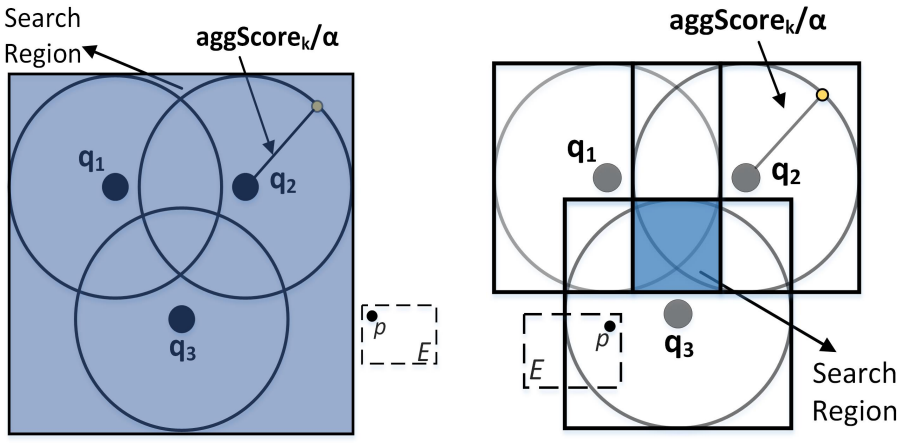
In particular, *RBR* is defined by corresponding aggregate function f and its size depends on current $aggScore_k$ and α i.e., $\frac{aggScore_k}{\alpha}$. The algorithm only accesses those entries which intersect with it. Figure 5.2 demonstrates two *RBR*s for *min, max and average* aggregate functions.

- ***Min***: Consider $SG\text{-}Top_k$ query with $Q = \{q_1, q_2, q_3\}$ in Figure 5.2(a). The shaded area corresponds to the RBR for *min* (where $n = 3$) which is a minimum bounding rectangle of union of three circles (centred at q_1, q_2, q_3) each with radius $\frac{aggScore_k}{\alpha}$. Entry E for example, should be not visited since it is not intersecting with the RBR and cannot contain a place p whose smallest score w.r.t. any of the $q_i \in Q$ is smaller than current $aggScore_k$ i.e., for any place p in E, $aggScore(p) > aggScore_k$.

- ***Max***: The RBR corresponds to the intersection of three minimum bounding rectangles (shaded area) for each circle (centred at correspond-

ing q_i) with radius $\frac{aggScore_k}{\alpha}$ as illustrated in Figure 5.2(b). Let's take an example of an entry E intersecting with the circle of q_3. This entry E should not be visited because for any place p in E, which is outside this intersected area, $dist(q_i, p) > aggScore_k$ for at least one q_i. Therefore, $aggScore(p) > aggScore_k$.

- ***Average***: For *f=Average*, the RBR is same as *f=min* because a place p for which $aggScore(p)$ regarding *f=min* is greater than $aggScore_k$, its $aggScore(p)$ regarding *f=Average* is also greater than $aggScore_k$. Note that a place p that is outside RBR, has $aggScore(p) > aggScore_k$ because its average distance is greater than $\frac{aggScore_k}{\alpha}$. For example, an entry E should not be visited since it is not intersecting with the RBR as shown in Figure 5.2(a). Therefore, it cannot contain a place p whose average score (aggScore(p)) w.r.t. all $q_i \in Q$ is smaller than current $aggScore_k$.



(a) Search Region (*f=avg, min*) (b) Search Region (*f=max*)

Figure 5.2: MBR Based Search Space

3. Observation on Social Score:

Recall that in step 1, to compute lower-bound aggregate score of an entry E, it requires traversing *FCR-Tree* another time while inserting it in priority queue for the first time. To avoid this overhead, we consider its parent's social score lower-bound as an initial estimate of $E's$ social score lower-bound which is a valid lower-bound on a child's social score.

We next describe the algorithm in detail with pseudocode given in Algorithm 9.

5.3.2.2 Algorithm Overview:

Algorithm 9 starts traversing *Facility R-tree* in best-first approach. For this purpose, a *min-heap* is initialized with the root node with $LBaggScore(root)$ as a sorting key (at line 3). To compute $LBaggScore$, it first invokes *Get-LBaggScore(mbr,Q)* (algorithm 8). Then in first loop (at line 4), algorithm starts de-heaping entries iteratively and examines whether or not it intersects with *RBR*, if it does not, it is immediately pruned along with all the entries lie inside (at line 6). Further, if the entry E overlaps and is a place p (an object), the algorithm computes its $aggScore(P)$ (at line 8) and update current k_{th} best place score $aggScore_k$ and RBR (at line 9).

Otherwise, the algorithm invokes *Get-LBaggScore(mbr,Q)* (algorithm 8) to compute $LBaggScore(E)$ (at line 11). Subsequently, if $LBaggScore(E)$ is better than current $aggScore_k$, in second loop, it starts en-heaping its child nodes provided that they overlap with RBR (at line 14). Moreover, if a child node c qualifies, the algorithm computes its $LBaggScore(c)$ by inheriting its social score from its parent. Consequently, if the estimated $LBaggScore(c)$ of c

Algorithm 9: Optimized Algorithm

```
1  min-heap = ∅, aggScore_k = ∞;
2  LBaggScore = Get-LBaggScore(root, Q);
3  Initialize min-heap with root of Facility R-tree with LBaggScore as a
   key ;
4  while min-heap ≠ ∅ do
5  |  De-heap entry E;
6  |  if E overlaps with RBR then
7  |  |  if E is an object then
8  |  |  |  Compute aggScore(E);
9  |  |  |  Update aggScore_k and RBR;
10 |  |  else
11 |  |  |  LBaggScore(E) = Get-LBaggScore(E, Q) // Algorithm 8
12 |  |  |  if LBaggScore(E) < aggScore_k then
13 |  |  |  |  foreach child node c of E do
14 |  |  |  |  |  if c overlaps with RBR then
15 |  |  |  |  |  |  LBaggScore(c) =
   |  |  |  |  |  |  α × minAggDist(c, Q) + (1 − α) × LBsocial(E);
16 |  |  |  |  |  |  if LBaggScore(c) < aggScore_k then
17 |  |  |  |  |  |  |  insert c in min-heap;
18 |  |  |  |  end
19 end
20 return Return Top-k Places
```

is less than the current $aggScore_k$, it is finally en-heaped for further processing (at line 17). Once *min-heap* is emptied, the algorithm terminates and reports top-k places (at line 20).

5.4 Experiments

5.4.1 Experimental Setup

To the best of our knowledge, this problem has not been studied before and no previous algorithm can be trivially extended to answer *SG-Top*$_k$ queries therefore, we evaluate the proposed algorithms on their performance by comparing them with each other.

Parameters	Values
Group Size (n)	2, **4**, 6, 8
f	*min*, *max*, *avg*
Query MBR Size (km)	50, **100**, 200, 400
Average Friends	200, 400, **600**, 800
k	5, **10**, 15, 20

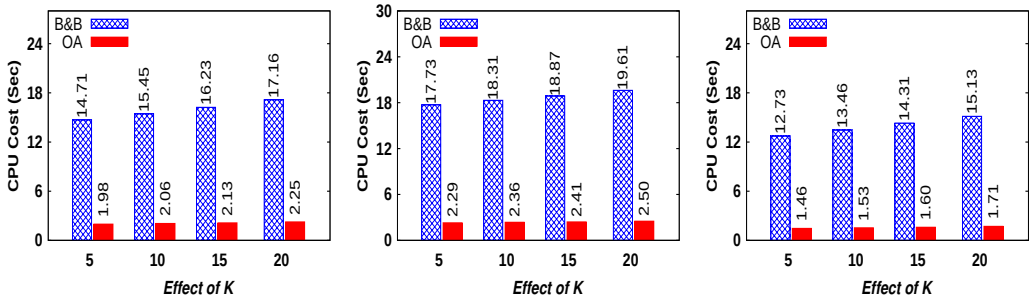
Table 5.2: Parameters (Default shown in bold)

Each method is implemented by employing the exact experimental settings and datasets described earlier in Section 3.5.1. Various parameters used in our experiments are shown in Table 5.2 where *Query MBR Size* represents the size of the region in which query users are spread. For each experiment, we randomly choose 10 groups of query users and consider them as query groups Q.

5.4.2 Performance Evaluation

Effect of k: In this evaluation, we test our proposed algorithms for various values of k for *min, max and avg* function. Note that for $f = min$ in Figure 5.3(a), *Optimized Algorithm* (OA) is upto 10 times faster and the performance is not significantly affected by the value of k. The reason is that, the main cost depends on traversing *Facility R-tree* and then computing lower bounds of the nodes and this dominant cost is not affected by k. Similarly, in Figure 5.3(b) for $f = max$, *Branch-and-Bound Algorithm* (B&B) takes little bit longer to process the query due to more number of places being qualified as candidates. However, for $f = avg$ both the algorithms performs better as compared to other functions where OA outperforms $B\&B$ by atleast 8 time as shown in Figure 5.3(c).

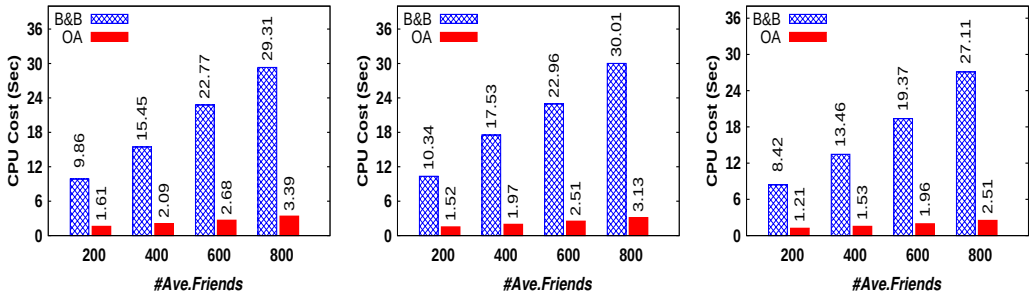
Effect of Average number of Friends: In this experiment, we study the effect of number of friends on $B\&B$ and OA algorithms in Figure 5.4. Note that the size of *FCR-Tree* relies on number of friends of a query user $q_i \in Q$.



(a) f=min (b) f=max (c) f=avg

Figure 5.3: Effect of varying number of requested places (k)

Also, the distribution of each friend's check-ins in search space determines the size of root node of *Check-in R-Tree*. This in return, affects the lower-bound on social score of *Facility R-tree* entries in *OA* Algorithm. In *B&B* algorithm, CPU cost mainly depends on computing the social score of the places $p \in P$ and as we increase the number of friends, the CPU cost increases. On the other hand, *OA* algorithm is less affected due to the optimization techniques. Note that, if f=avg, both the algorithms perform better than *min and max* functions due to lower $aggScore_k$ which aids in pruning more places.

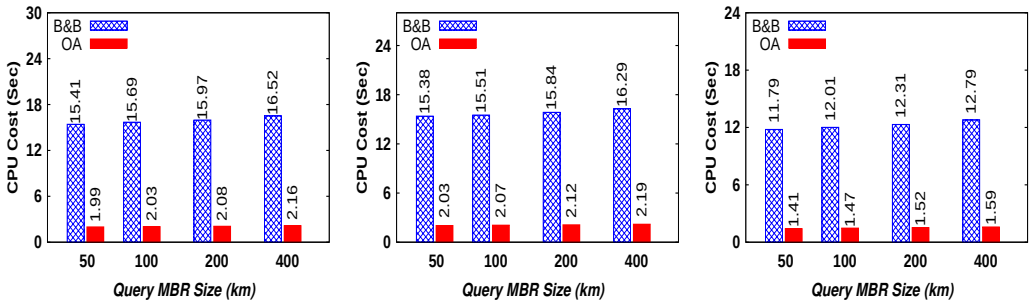


(a) f=min (b) f=max (c) f=avg

Figure 5.4: Effect of varying number of Friends

Effect of Query MBR Size: Next in Figure 5.5, we evaluate the performance of our algorithms on query MBR size. For this purpose, we randomly spread the query users in the region of size between 50 to 400 kilometres. Note that, as we increase the size, it does not affects the query processing to great extent since the main cost involves traversing *Facility R-tree*, computing lower-bounds

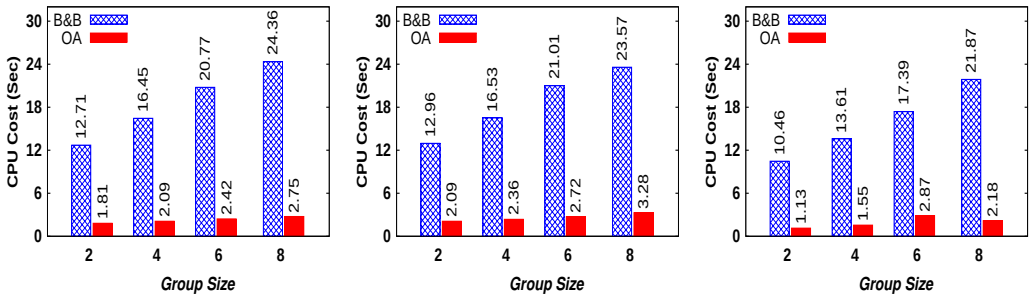
and social score of the query. Note that for $f = min$ and $f = max$ in Figure 5.5(a) and 5.5(b) respectively, *Optimized Algorithm* (OA) is upto 10 times faster than $B\&B$ algorithm. However, if $f = average$, the algorithms perform relatively better as illustrated in Figure 5.5(c).



(a) f=min (b) f=max (c) f=avg

Figure 5.5: Effect of varying Query MBR Size

Effect of Group Size: In this evaluation, we test our proposed algorithms for different group sizes ranging from 2 to 8 for *min, max and avg* aggregate functions in Figure 5.6. As we increase the group size, it greatly affects the performance of the two algorithms because the algorithms have to process spatial and social information for more query users. However, for larger groups, *OA* performs better than the other one. In addition, if $f = avg$, both the algorithms perform relatively better than *min and max* as shown in Figure 5.6(c).



(a) f=min (b) f=max (c) f=avg

Figure 5.6: Effect of varying Group Size (n)

5.5 Conclusions

In this chapter, we study Spatial Group Top-k queries in Social Networks and formalize this problem as a *Geo-Social Group preference Top-k* ($SG\text{-}Top_k$) query that retrieves near by places popular among a particular group of users w.r.t. each query user $q_i \in Q$ in the social network. We present *Branch-and-Bound* algorithm followed by some optimization techniques to further improve the algorithm. The *Branch-and-Bound* approach uses R-tree to process places in ascending order of their aggregate distance from each query user $q_i \in Q$ and then computes social score of each place p to finally compute aggregate score i.e., $aggScore(p)$. Since computation of $aggScore(p)$ of each $p \in P$ is computationally expensive, our Optimized approach leverages the *FCR-Tree* to offer efficient pruning techniques to prune irrelevant place. Moreover, the optimized algorithm computes a region based on current found $top - k_{th}$ aggregate score to quickly check the entries that can be pruned. In addition to this, we conduct an exhaustive evaluation of the proposed schemes using real dataset and test them on various aggregate functions e.g., *min, max, avg.* Our results demonstrate the effectiveness of the proposed schemes.

Chapter 6

Spatial Temporal Top-k Queries in Social Networks

In this chapter, we investigate Spatial Temporal Top-k queries in Social Networks and propose a new type of query called *Geo-Social Temporal Top − k (GSTTk)* query by adding a third dimension (time) to our first work presented in chapter three which retrieves top-k places considering their distance from the query user q and popularity of the place among q's social connections during a certain period of time. The outline of the chapter is as follows: Section 6.1 introduces the research problem and the contributions we have made, Section 6.2 explains the problem definition, Section 6.3 describes the proposed techniques, Section 6.4 consists of experimental evaluation and Section 6.5 concludes the chapter.

6.1 Introduction

Temporal queries retrieve query results based on given temporal properties. It is noteworthy that time dimension has a strong influence in many domains, for

example, Topic Detection and Tracking, Spatial queries, Information retrieval, Top-k queries, Geo-Textual queries [131, 132]. Therefore, in this chapter, we added a third dimension (time) to our first work presented in chapter three and study a problem of finding top-k places considering their distance from the query user q and popularity of the place among q's social connections during a certain period of time. Consider an example of a visitor from Switzerland visiting Melbourne. She maybe keen in finding a nearby *café* which serves *Rösti* (a traditional Swedish hot cake) with coffee and has become popular (e.g., frequently visited) among people from Switzerland in last year. This involves utilizing spatial information (i.e., nearby *café*, check-ins), social information (i.e., people who were `born-in` Switzerland) as well as temporal information (i.e., *cafés* that are visited during last year).

The applications of such queries are not only limited to traditional location-based social services. These can also be used in disaster management, public health, security, tourism, marketing etc. For example, in disease monitoring, we may want to find frequently visited places (top-k) in last 6 months by people infected by Ebola virus. Consider a health-based social network where each health risk (e.g., Ebola) is an entity and people affected by it are connected to it via an edge. One can issue a query to find the top-k frequently visited places in the last 6 months by the one-hop neighbors of the Ebola entity. Similarly, for instance in public safety and crime prevention, law enforcement agencies may be keen on finding frequently visited places in last one month by users who have tweeted about *Drugs* and have also joined some pages/groups containing/sharing drugs related information on social networks. The users are socially connected through an edge (entity) e.g., a tweet, a page or a group in social network and then agencies can exploit one-hop neighbours of the entities to find frequently visited places to raid and prevent drugs dissemination.

Further, such kind of queries can also be exploited by various businesses. Consider a chain of Chinese grocery stores that is interested in opening a new store in an area/shopping centre which is frequently visited by Chinese people in recent months. The Chinese people are socially connected through a Facebook page for *China* having `born-in` relationship. By analyzing their visits information in recent months, the company can discover a suitable venue for this purpose.

As we have remarked before, several types of queries have been investigated on LBSNs [31, 149], to the best of our knowledge, none of the existing methods can be applied to answer the queries like mentioned above that aim at finding nearby places that are popular among a particular group of users satisfying social and temporal constraint. Motivated by this, in this chapter, we formalize this problem as a *Geo-Social Temporal Top-k* ($GSTT_k$) query and propose efficient query processing techniques. Specifically, a $GSTT_k$ query retrieves top-k places (points of interest) ranked according to their spatial, social and temporal relevance to the query user where the spatial relevance is based on how close the place is to a given location and, the social and temporal relevance is based on how frequently it is visited by the one-hop neighbors of the query user in the social graph during given time interval. A formal definition is provided in Section 6.2.

We make the following contributions in this chapter.

- We believe that we are the first to study the $GSTT_k$ query that retrieves nearby places popular among a particular group of users in the social network during specified temporal interval.

- At first, we present two different approaches i.e., *Social-First* and *Spatial-First* to solve our problem and then we propose our main algorithm called

Hybrid. In addition, we propose three dimensional index structures like *3DFCR-Tree and 3D Check-In R-Tree* which facilitate in efficient pruning of objects based on temporal constraints.

- We conduct an exhaustive evaluation of the proposed schemes using real dataset and demonstrate the effectiveness of the proposed approaches. Our experiments show that our main algorithm outperforms the other two.

6.2 Problem Definition

The score of a place is defined in a similar way as in Chapter 3. We briefly describe it here again.

Score of a place *p*: Given a query user q, a range r and a temporal interval $I[st, et]$ (where st denotes start time and et denotes end time), the score of a place $p \in P$ is 0 if $||q, p|| \geq r$ where $||q, p||$ is the Euclidean distance between query location and p. If $||q, p|| \leq r$, the score of p is a weighted sum of its spatial score (denoted as $p_{spatial}$) and its social score (based on number of check-ins of query q's friends in the given temporal interval I, which is denoted as p_{social}).

$$p.score = \alpha \times p_{spatial} + (1 - \alpha) \times p_{social} \tag{6.1}$$

Let $p.visitors$ denotes the set of all users that visited (i.e., checked-in at) the place p during given temporal interval $I(st, et)$. The social score p_{social} of place p is computed as follows:

$$p_{social} = \frac{|F_q \cap p.visitors|}{|F_q|} \tag{6.2}$$

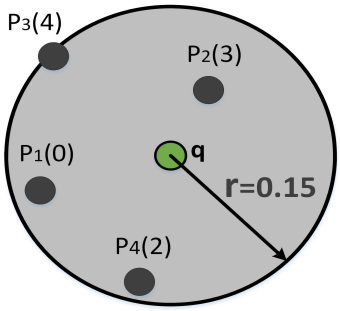
Intuitively, p_{social} is the proportion of friends of q who have visited a place p during given temporal interval I.

The spatial score $p_{spatial}$ is based on how close the place is to the query location. Formally, given a range r, $p_{spatial} = 0$ if the place does not lie in the range r. Otherwise, $p_{spatial} = (r - ||q, p||)$ where $||q, p||$ indicates Euclidean distance between query location and p. Note that p_{social} is always between 0 to 1 and we normalize $p_{spatial}$ such that it is also within the range 0 to 1, e.g., the data space is normalized such that $||q, p|| \leq 1$ and $r \leq 1$.

Geo-Social Temporal Top-k ($GSTT_k$) Query: Given an LBSN, a $GSTT_k$ query q returns k places with the highest scores where the score $p.score$ of each place p is computed as described above.

Example 2.1: Figure 6.1(a) illustrates the locations of a set of places $P = \{p_1, p_2, p_3, p_4\}$ and a query q. Let's assume that the query q is with $k = 2$, range $r = 0.15$, temporal interval I is *"during last year"* and has a set of friends $F_q = \{u_1, u_2....u_9, u_{10}\}$. The number in bracket next to each place is the number of friends of q who visited the place during last year. Figure 6.1(b) shows the Euclidean distances and visitors (from the friends of q i.e., F_q) of each place during all times (column 3) as well as during last year (column 4). Let's assume $\alpha = 0.5$, the spatial score of p_2 is $p_{spatial} = 0.07$, the social score of p_2 is $p_{social} = 0.30$ and by applying Equation 6.1, we get the score of p_2 i.e., $p_2.score = 0.5 \times 0.07 + (1 - 0.5) \times 0.30 = 0.185$.

Similarly, for p_1, the spatial score is $p_{spatial} = 0.05$, the social score of p_1 is $p_{social} = 0.0$ and by applying the same equation, we get the score of p_1 i.e., $p_1.score = 0.5 \times 0.05 + (1 - 0.5) \times 0.0 = 0.025$. For p_3 and p_4, their scores will be $Score(p_3) = 0.205$ and $Score(p_4) = 0.115$ respectively. The result of the query q is (p_2, p_3) according to scoring function in equation 6.1. The important notations used in this paper are listed in Table 6.1.



Range = shaded area

(a) A Query q

P	$\|\|q, p_i\|\|$	$P_{i.visitors}$ From F_q	$P_{i.visitors}$ From F_q (during I)	*Final Score*
P_1	0.10	u_3, u_7	-	*0.025*
P_2	0.08	u_1, u_3, u_9, u_7, u_8	u_1, u_3, u_9	*0.185*
P_3	0.14	$u_3, u_5, u_6, u_2, u_7, u_8$	u_5, u_6, u_2, u_7	*0.205*
P_4	0.12	u_5, u_7, u_{10}	u_7, u_{10}	*0.115*

$F_q = \{u_1, u_2 \ldots\ldots u_9, u_{10}\}$, $\alpha = 0.5$, $k = 2$, $r = 0.15$
I = "During Last One Year"

(b) Sample Dataset

Figure 6.1: Temporal Top-k Query Example

6.2.1 Framework Overview

The proposed framework consists of three techniques to answer $GSTT_k$ query: I) Social-First, II) Spatial-First and III) Hybrid. The *Social-First* approach first processes the social component (e.g., friendship relations and their check-ins) for given temporal interval I and then processes spatial component (e.g., places in given range). The *Spatial-First* approach initially processes the spatial component followed by processing the social component. In contrast, *Hybrid approach* is capable of processing both social and spatial components simultaneously to answer the query. More specifically, it utilizes two types of pre-processed information associated with each user $u \in U$.

1. Her check-in information (check-ins) and summary of her friends' check-

Notation	Definition
q	Query User i.e., $q \in U$
u	User i.e., $u \in U$
p	A place i.e., $p \in P$
I	Temporal Interval
st	Start Time
et	End Time
F_q	Set of Friends of query user q
$p.visitors$	Set of visitors of a place p
r	Given query range
k	Number of requested places
c_{ij}	Range Grid's cell
P_c	Set of places lie inside a cell
V_{cell}	Set of visitors of a cell

Table 6.1: Notations

ins information.

2. The information which summarizes the visitors' check-in for each place p.

Precisely, we index places, users' check-in information and visitor's check-in information by exploiting R-tree [118]. Before presenting our techniques, we briefly describe the indexes used in the techniques.

3D Check-In R-Tree: For each user u, we create a *3D Check-In R-Tree* which indexes all the check-ins of the u. This is a three-dimensional R-tree, where two dimensions belong to location coordinates of a check-in and the third dimension corresponds to timestamp of the check-in.

6.3 Proposed Techniques

6.3.1 Social-First based Approach

In this approach, scores of the places in given range r are computed by considering the check-ins of each friend $u \in F_q$. Specifically, for each friend $u \in F_q$, its *3D Check-In R-Tree* is traversed to obtain the places in the range where u has checked-in during given temporal interval I. The social score of each checked-in place by any friend is updated. When every user $u \in F_q$ is processed, we have the final social score of each place in the range. Next, the algorithm considers each place in the range and computes its final score. Finally, the top-k places are returned.

Let's assume, the score of current k_{th} place p is $Score_k$, it was shown in [24] that if the $||q, p|| \geq (r - \frac{Score_k}{\alpha})$, we can prune that place p. We next describe the technique in detail with pseudocode given in Algorithm 10.

Initially, in the first loop of the algorithm (at line 1), it exploits *3D Check-in R-Tree* of each friend $u \in F_q$ of query q to get all places in range r provided that, its time dimension intersects with the given time interval I. Then, the algorithm computes social score and score of each candidate place p in r (at line 4) while maintaining the score of current k_{th} place p (at line 7). Further, in the third loop (at line 11), the *Facility R-Tree* is exploited to compute the score of those places $p \in P$ in range r which are not visited by q's friends (at line 13). Hence, their respective scores only comprises of spatial score and their $(P_{social}) = 0$. Finally, the *top-k* result set is retrieved using $Score_k$ (at line 16).

Algorithm 10: Social-First

Input : Query q, range r, weight-age constant α, integer k, Time Interval $I(st, et)$
Output: Result set R

```
1  foreach friend u in F_q do
2      if time interval I overlaps with 3D Check-In R-tree's time
       dimension then
3          Issue a range query on 3D Check-In R-tree;
4          foreach place p in r do
5              if st ≤ time_check−in(p) ≤ et then
6                  update social score and score of p;
7                  update Score_k;
8          end
9  end
10 Issue a range query on Facility R-Tree;
11 foreach place p in range r do
12     if p.dist ≤ (r − (Score_k/α)) then
13         compute Score(p);
14         update Score_k;
15 end
16 return R
```

6.3.2 Spatial-First based Approach

Initially, this approach retrieves all places in given range r and computes spatial score of each place p in ascending order of distance from q. For each accessed place, its social score is computed by exploring the visitors of the place and the friends of the query user q. For each unaccessed place p, an upper bound is computed using its distance from q and assuming its social score to be 1 (the maximum possible). The algorithm terminates if the upper bound score of the next place is smaller than the score of k_{th} place found so far.

Let's assume, the score of current k_{th} place p is $Score_k$, it was shown in [24] that if the $||q, p|| \geq (r - \frac{(Score_k - (1-\alpha))}{\alpha})$, the process stops since every subsequent place p in the priority queue is further than the current place p from q. Next, we elaborate the technique in detail with pseudocode given in Algorithm 11.

The algorithm starts with the issuance of a range query on *Facility R-Tree* to retrieve all places in range r (at line 2). Then in first loop (at line 3), for each place $p \in P$ in range r, we compute the *Score* of p in ascending order of the distance of the place p from q. To achieve this, a *min-heap* is initialized with the root entry of *Facility R-Tree* with $||q, e||$ as a key to process spatial component first. Further, it computes social score of a place p by retrieving the number of friends $u \in F_q$ who visited the place in given temporal interval I by exploring the visitors' information of the place (at line 8). Finally, the final score of place p is computed using equation 6.1 (at line 12). If the distance of the retrieved place p from q i.e., $||q, p|| \geq (r - \frac{(Score_k - (1-\alpha))}{\alpha})$, the process stops (at line 5) since every subsequent place p entry in *min-heap* is further than the retrieved place p entry from q.

Algorithm 11: Spatial-First

Input : Query q, range r, weight-age constant α, integer k, Time Interval $I(st, et)$
Output: Result set R

```
1  V_p = ∅;
2  Issue a range query on Facility R-Tree;
3  foreach place p in range r do
4  |  if ||q,p|| ≥ (r − ((Score_k − (1 − α))/α)) then
5  |  |  return Result set R
6  |  V_p = Get p.visitors;
7  |  sort V_p on time;
8  |  foreach visitor v ∈ V_p do
9  |  |  if st ≤ V_checkInTime ≤ et then
10 |  |  |  update P_social;
11 |  end
12 |  Compute Score of p ;
13 |  update Score_k ;
14 end
15 return Result set R
```

6.3.3 Hybrid Approach

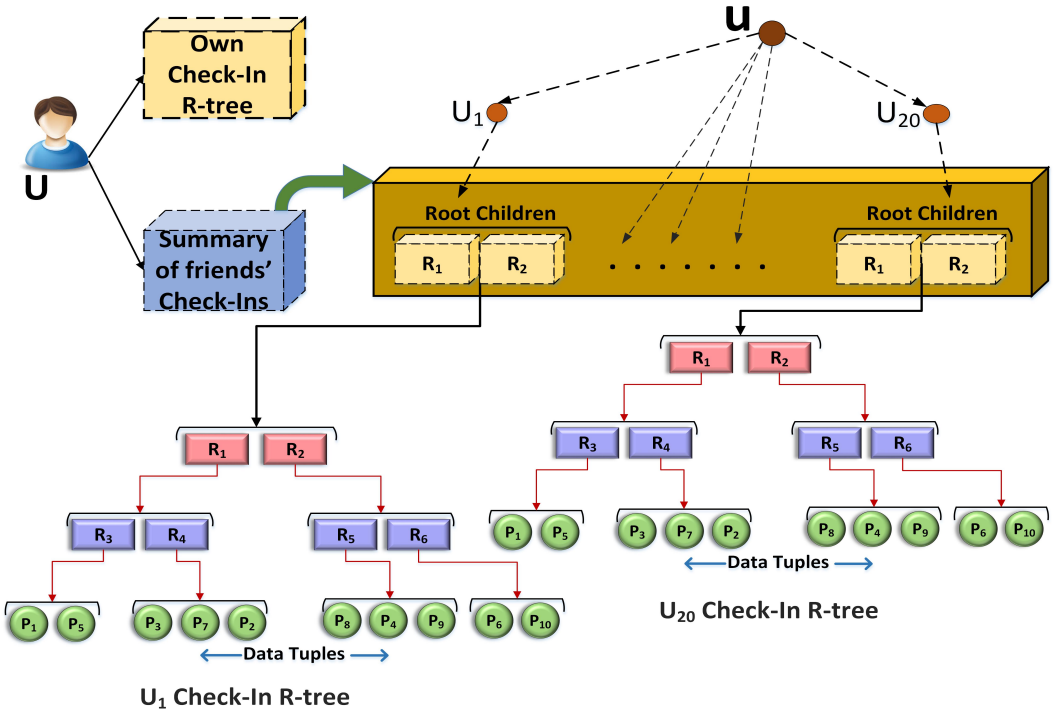
This section focuses on our third approach (i.e., Hybrid) to process $GSTT_k$ queries which is capable of processing social, spatial and temporal components simultaneously. Before presenting the technique in detail, we describe our index and space partitioning techniques.

3D Friends Check-Ins R-Tree: In addition to the previous indexes, for each user u, we introduce another index called *3D Friends Check-Ins R-tree (3DFCR-Tree)* which maintains the summary of check-ins of the user u's friends. Specifically, *3DFCR-Tree* stores check-in information of each friend of u by indexing a few MBRs for each friend. Thus, it represents the summary of all friends check-ins.

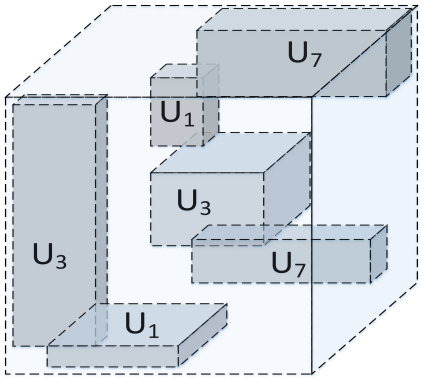
One approach is to use the root MBR of each of u's friends *3D Check-In R-Tree* and index them in *3DFCR-Tree*. The problem with indexing root MBR is that, many root nodes may be too big (e.g., consider a user who has checked-in in every continent) and this would result in huge overlap among the MBRs affecting the effectiveness of the R-tree. To overcome the shortcoming, we propose to index the children of the root nodes instead of the root nodes.

Let's assume a query $q \in U$ where the friends of q are $F_q = \{u_1, u_2, u_3 \ldots u_{19}, u_{20}\}$. Figure 6.2(a) illustrates the idea of the *3DFCR-Tree* of q. Similarly, Figure 6.2(b) shows one of the leaf nodes of a *3DFCR-Tree* which indexes child entries of root MBR of few friends' (e.g., u_1, u_3, u_7) *3D Check-In R-Trees*.

Visitors Check-Ins R-Tree (*VCRTree*): As described earlier, for each user, we maintain her friends' summary to prune irrelevant friends when a query arrives. Similarly, each place p has visitors ($p.visitors$ containing their IDs) and their check-ins information during different times. To maintain this information, we create an R-tree (denoted as *VCRTree*) and each indexed



(a) Summary of Friends' check-Ins



(b) Leaf Node

Figure 6.2: 3D Friends Check-Ins R-Tree

point is a two-dimensional point where one dimension is visitor ID and other dimension is check-in time.

6.3.3.1 Algorithm Overview:

Initially, when a query arrives, we create a two dimensional grid (which covers given range r) on the fly. For each cell of the grid, we compute an upper bound on score for each place that may lie in the cell (to be explained later) and based on the upper bound, we access places in the order to prune unnecessary places and friends which are not relevant. Secondly, by employing *VCRTree* for remaining candidate places, we further prune the irrelevant ones based on social and temporal criteria. Below, we explain the pruning criteria in detail with pseudocode given in Algorithm 12.

Using Range Grid: In this step, we construct the on the fly 2D-Grid to prune two kind of objects based on *3DFCR-Tree* as follows.

1. Pruning Friends: If an MBR of *3DFCR-Tree* does not overlap with the grid or with given temporal interval I, we can prune it which in return, prunes that particular friend. The pruned friends are the friends of the query who have not checked-in in given range r during given temporal interval I.

Specifically, to compute the upper bound on social score of a cell c_{ij} of range grid, the algorithm traverses *3DFCR-Tree* of a query user q to compute number of objects (child nodes of root of friends' Check-In R-trees) intersecting with the cell. Let's consider an example in Figure 6.3 where we have a range grid cell c_{ij} and some *3DFCR-Tree* objects belonging to q's friends ranging from u_1 *to* u_5. Since only u_1, u_4 *and* u_5 overlap with c_{ij}, they might have checked-in at any place p in the cell. Therefore, the maximum number of friends who might have visited a place in the cell is 3, which can be used to obtain the upper bound on social score.

2. Pruning Places: Each cell c_{ij} in the grid contains a list of places

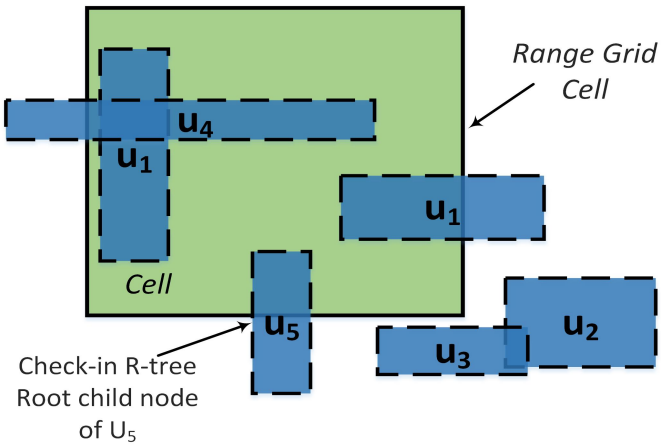


Figure 6.3: A cell's Social Score Upper Bound

P_c that lie inside it and a list of overlapping MBRs of *3DFCR-Tree* (denoted as V_{cell}) based on criteria described above. Once the list V_{cell} for each cell is fetched, an upper bound on the score (denoted as $Score_{cell}$) of each cell is computed using equation 6.3.

$$Score_{cell} = \alpha(r - mindist(cell, q)) + (1 - \alpha)\left(\frac{|V_{cell}|}{|F_q|}\right) \tag{6.3}$$

Since, $|V_{cell}|$ denotes the number of query friends who might have visited a place in the cell within the query temporal interval, the upper bound on social score of a cell is computed as $\frac{|V_{cell}|}{|F_q|}$. Similarly, $(r - mindist(cell, q))$ gives an upper bound on the spatial score of any place in the cell where $mindist(cell, q)$ denotes the distance between the query q and the nearest place p to the q in the cell.

In first loop (at line 1), for each cell in descending order of the $Score_{cell}$, the algorithm accesses each place p in the cell (at line 4) to compute score of the place while maintaining the current k_{th} place score (at line 7). If the current k_{th} place score is greater than the next cell's $Score_{cell}$, the algorithm stops since all the subsequent cells can not contain a place with higher score

than current k_{th} place's score (at line 3). Below, we describe how to compute the score of a candidate place efficiently.

Algorithm 12: Hybrid Algorithm

```
1 foreach Range Grid cell c_ij in descending order of upper bound scores
  do
2     if Score_cell ≤ Score_k then
3         return top-k results;
4     foreach place p in cell c_ij do
5         ComputeScore(p) ;                          // Algorithm 13
6     end
7     Update top-k results and Score_k;
8 end
```

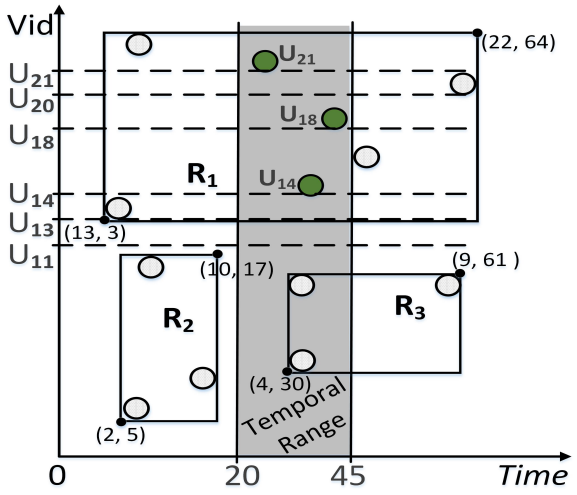
6.3.3.2 Computing Score of a Candidate Place:

Intuition: A naïve approach: Let's consider a query q with a list of friends and their check-ins, and a place p with a list of visitors. To compute score of the place, we have to traverse through whole friends' list and visitors' list to see if a friend has visited the place during given temporal interval I.

In general, the above approach is not efficient since in many applications, the size of F_q may be huge e.g., people born-in Germany. To speed-up the query processing, we employ an R-Tree (*VCR-Tree*) to index visitor IDs and their check-in times (as described earlier).

Let's consider an example where we have three MBRs of *VCR-Tree* (R_1, R_2 *and* R_3) shown in Figure 6.4 along with given temporal interval $I(20, 45)$ (shaded area) and a list of friends F_q. Clearly, MBR R_2 neither intersects with any user in F_q (see the broken lines) nor with given temporal interval I. Therefore, we can prune the MBR. Note that, MBR R_3 does overlap with given temporal interval I. However, it does not intersect with any user in F_q

and can also be pruned. Similarly, we can prune user U_{11} since none of the three MBRs intersect with it. Since MBR R_1 intersects with both the F_q and given temporal interval I, it might contain check-ins of the friends. In Figure 6.4, the relevant users of the query q are U_{14}, U_{18} and U_{21} (shown in solid) who visited the place during given temporal interval I. Next, we explain the technique in Algorithm 13 with pseudocode given below.



F_q	U_{11}, U_{13}, U_{14}, U_{18}, U_{20}, U_{21}
I(st, et)	(20, 45)

Figure 6.4: Social Score Computation Example

To compute the score of a candidate place p, the algorithm starts finding the friends of q who visited the place p by traversing the F_q in ascending order of friend IDs (at line 1). For this purpose, it accesses MBRs of *VCRTree* of the place p based on their minimum visitor ID by first initializing a *min-heap* with root MBR of *VCRTree* with *minimum visitor ID* as sorting key (at line 2). Then, in first loop (at line 3), the algorithm starts de-heaping entries iteratively and examines whether or not it intersects with the remaining number of friends in F_q (to be verified as visitors and denoted as $F_{q.remaining}$) and given temporal

interval I (at line 7). If the entry E intersects with either of the two and is also an object (i.e., check-in belongs to a friend in the F_q), the algorithm either updates the social score of the place p or prunes it based on an upper bound on the score of the place p (denoted as $P_{maxScore}$) using Equation 6.4.

$$P_{maxScore} = \alpha \times p_{spatial} + (1 - \alpha) \times P_{maxSocial} \tag{6.4}$$

To choose from one of the two options, the algorithm first computes $P_{maxScore}$ of the place p using maximum possible social score of the place p (denoted as $P_{maxSocial}$) which is computed as described in Equation 6.5. Let $F_{q.traversed}$ denotes a subset of the F_q which has been traversed so far to find friends in the F_q who visited the place p and assuming all the remaining friends in the $F_{q.remaining}$ have visited the place p.

$$P_{maxSocial} = \frac{|F_{q.traversed} \cap p.visitors| + |F_{q.remaining}|}{|F_q|} \tag{6.5}$$

Intuitively, $P_{maxSocial}$ is the maximum possible social score of a place p at any point during the computation of final score of the place p. Consequently, if $P_{maxScore}$ of the place p is less than the current k_{th} place score, the algorithm terminates without computing the final score and prunes the place p (at line 10) since it cannot be in top-k places. Otherwise, it updates the social score of the place p (at line 12). Similarly, if the entry E does not overlap either with the $F_{q.remaining}$ or given temporal interval I, it is instantly pruned and consequently, its child entries are not en-heaped.

6.3.3.3 Handling Updates:

Algorithm 13: ComputeScore(p)

```
1  F ← first friend in F_q;
2  Initialize min-heap with Root of VCR-Tree;
3  while min-heap is not empty do
4  |  De-heap entry E;
5  |  if F < Minimum Visitor ID of E then
6  |  |  F ← binary search to find first F in F_q with ID >= minimum
   |  |  visitor ID of E;
7  |  |  if (E overlaps with given temporal interval I or with the friend
   |  |  F then
8  |  |  |  if E is an object then
9  |  |  |  |  if P_maxScore < Score_k then
10 |  |  |  |  |  Prune the place p;
11 |  |  |  |  else
12 |  |  |  |  |  Update Social score ;
13 |  |  |  else
14 |  |  |  |  Insert child entries of E into min-heap with minimum
   |  |  |  |  visitor ID as a key;
15 end
16 Return Score(p);
```

Now, we provide a very high level idea of how to update the indexes. For this purpose, we index last month data using a separate data structure in addition to the data structure that maintains all previous months data and during query processing, we use both the data structures. Similarly, to update the data structures, a periodic bulk update is performed.

6.4 Experiments

6.4.1 Experimental Setup

To the best of our knowledge, this problem has not been studied before and no previous algorithm can be trivially extended to answer $GSTT_k$ queries. There-

Parameters	Values
Number of Queries	50, **100**, 150, 200
Range (km)	50, **100**, 200, 400
Temporal Interval (Months)	1, 3, **6**, 12
Grid Size	8, **16**, 24, 32
Average Friends	200, 400, **600**, 800
k	5, **10**, 15, 20

Table 6.2: Parameters (Default shown in bold)

fore, we evaluate the proposed algorithms on their performance by comparing them with each other.

Each method is implemented by employing the exact experimental settings and datasets described earlier in Section 3.5.1. Various parameters used in our experiments are shown in Table 6.2 where *Query MBR Size* represents the size of the region in which query users are spread. For each experiment, we randomly choose 100 users and consider them as query users.

6.4.2 Performance Evaluation

Index Size: Figure 6.5 compares the index sizes of five different subsets of the real dataset. To obtain these datasets, we randomly selected $100,000$ to $500,000$ places and we extracted their corresponding social networks based on visitors of the places. The input data contains *places, check-in information, friends and their relationship information* in few simple text files (without indexing). The value on top of each Bar denotes how many times bigger the respective index size is compared to the input data. For example, for $100,000$ places, the size of indexes utilized by the *Social-First* algorithm is 2.29 times bigger than the input data. Note that the size of all our indexes is linear to the input data for all datasets (e.g., *Hybrid* is 3-4 times bigger than the input data). As expected, *Hybrid* index is the largest index.

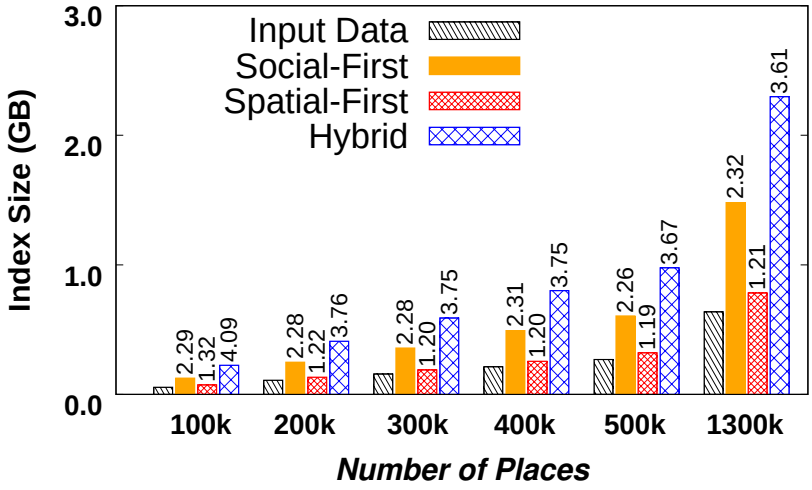


Figure 6.5: Index Size

Effect of Grid Size: In Figure 6.6, we study the effect of different number of cells in which grid is partitioned in *Hybrid* technique. The CPU cost also depends on the number of cells because it affects grid partitioning and grid cells' upper bound computation which plays a vital role in pruning phase. In our study, we found that the best CPU performance can be achieved by splitting the region covering given range r into grid of 16×16 cells for the default parameters.

Effect of k: In this evaluation, we test our proposed techniques for various values of k. As shown in Figure 6.7(a), *Hybrid* is up to 15 times faster and the performance is not significantly affected by the value of k. The reason is that, the main cost depends on creating the grid and then computing upper bounds of the cells and this dominant cost is not affected by k. Similarly, I/O cost remains unaffected to much extent as illustrated in Figure 6.7(b) for all the three algorithms since the higher value of k does not incur more disk access. Note that, *Hybrid* performs better than the other two even though it processes more indexes. However, due to efficient pruning techniques, it incurs less I/O

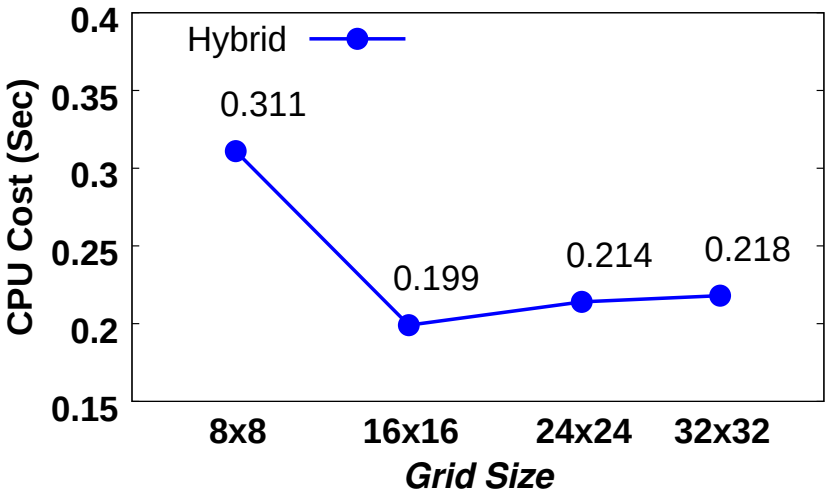
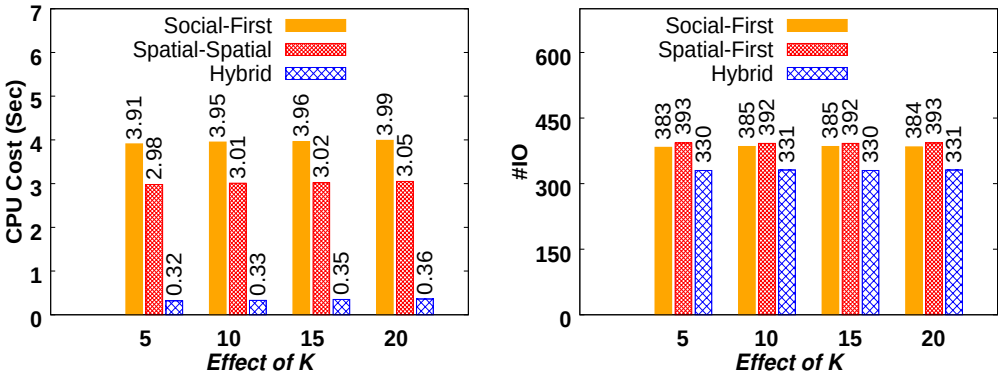


Figure 6.6: Effect of Grid Size

cost.

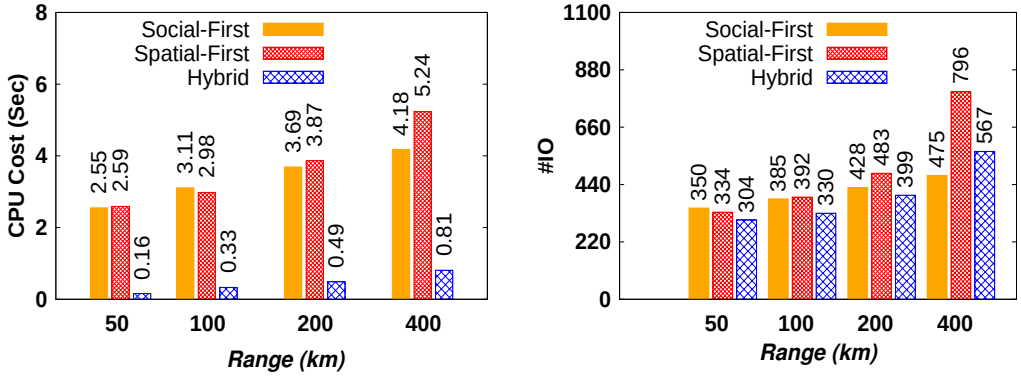


(a) CPU cost (b) I/O cost

Figure 6.7: Effect of varying number of requested places (k)

Effect of Range: Next, we evaluate the performance of our techniques for range between 50 to 400 kilometers in Figure 6.8. The region in which we want to find top-k places is defined by the given range r containing average number of places between $5,000$ *to* $100,000$. Note that, *Spatial-First* is linearly affected as we increase the range r due to linear growth in number of places as shown in Figure 6.8(a). Similarly, *Social-First* shows a steady growth in CPU cost with the increase in range r. Although it accesses the *3D Check-In R-Tree*

for each friend but the cost is increased because the cost of range query on these R-trees is affected with the range r. Note that, *Hybrid* performs several times better than the other two. Further, in terms of I/O cost, as we increase the range r, *Social-First* again shows a steady growth since the number of *3D Check-In R-Trees* which need to be processed, is independent of the range r as illustrated in Figure 6.8(b). Note that, *Spatial-First* is most affected since as we increase the range r, it has to process more places which in return, incurs more disk access.

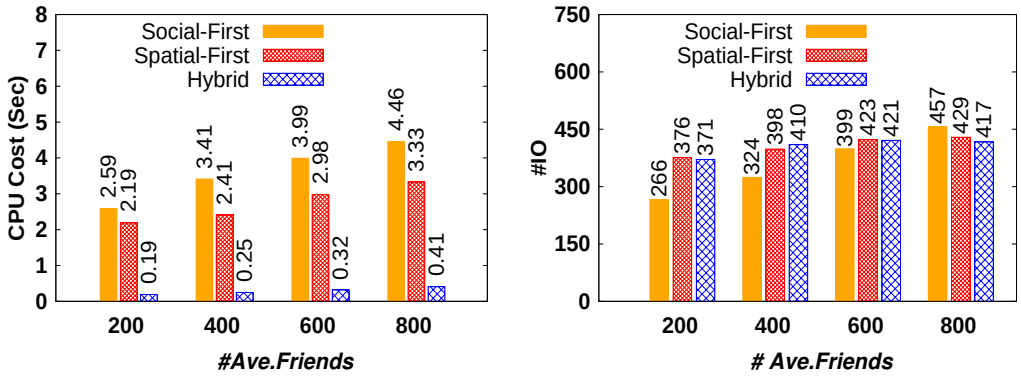


(a) CPU cost (b) I/O cost

Figure 6.8: Effect of varying Range

Effect of Average number of Friends: In this experiment, we study the effect of number of friends on the three techniques in Figure 6.9. Note that the size of *3DFCR-Tree* relies on number of query q's friends and the distribution of each friend's check-ins in search space which determines the number of objects to be indexed in *3DFCR-Tree*. This in return affects the upper bound of grid cells in *Hybrid* technique. In *Spatial-First* technique, CPU cost is mainly dependent on the cost of range query on *Facility R-Tree* and to some extent on number of query q's friends which affect the social score computation module as depicted in Figure 6.9(a). Similarly, as we increase the number of friends, *Social-First* has to process more *3D Check-In R-Trees* which affects its CPU

cost. Note that, when we increase the average number of friends, we found that *Social-First* is linearly affected since it requires to access *3D Check-In R-Tree* for each friend as shown in Figure 6.9(b).

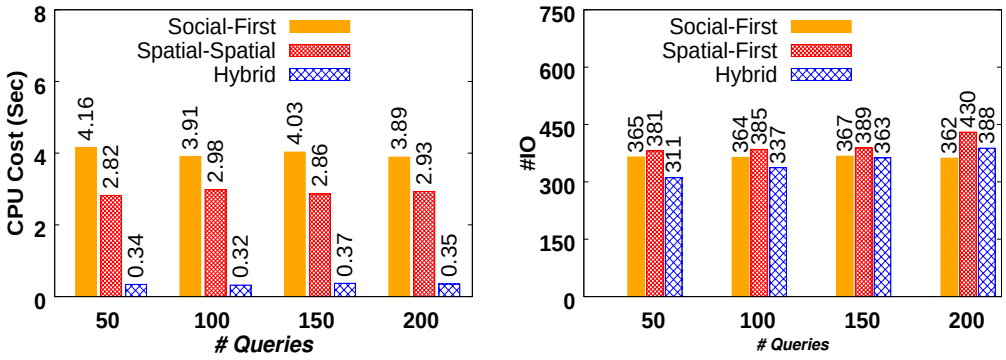


(a) CPU cost (b) I/O cost

Figure 6.9: Effect of varying number of Friends

Effect of concurrent number of Queries: Next, we compare the performance of our techniques for various number of queries. Figure 6.10(a) shows average CPU cost of the techniques which slightly varies depending on the number of query q's friends and query location. *Social-First* technique has higher CPU cost for any number of queries than the other two because it accesses *3D Check-In R-Tree* for each friend and issues a range query as illustrated in the figure. Similarly, *Spatial-First* has slightly different CPU cost because of different query and number of places in range r. Note that, *Hybrid* algorithm performs several times better and the cost for different number of queries is slightly affected by the query location, number of friends and number of places in range r. Similarly, for all the three algorithms, the average I/O cost is mainly independent of the number of queries and is slightly affected by the query location, number of friends and number of places in range r as depicted in Figure 6.10(b).

Effect of Temporal Interval: In Figure 6.11, we evaluate the effect of size of

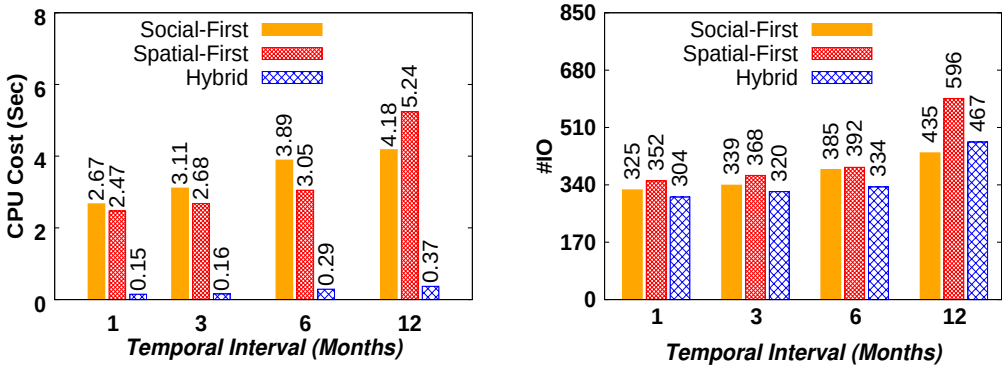


(a) CPU cost

(b) I/O cost

Figure 6.10: Effect of varying number of Queries

temporal interval I to test the performance of the three methods. *Social-First* algorithm has higher CPU cost even for smaller temporal intervals because most of *3D Check-In R-Trees* overlap with the temporal interval as shown in Figure 6.11(a). Similarly, the CPU cost of *Spatial-First* method remains high specifically for bigger temporal intervals due to more number of places to be processed. On the other hand, in *Hybrid* technique, temporal interval affects cells' upper bound computation and consequently, the pruning phase gets affected. Note that, *Hybrid* performs several times better than the others. Further, in terms of I/O cost, as we increase the temporal interval I, *Social-First* shows a steady growth since the number of *3D Check-In R-Trees* which need to be processed, slightly depends on the temporal interval I as illustrated in Figure 6.11(b). Note that, *Spatial-First* is most affected since as we increase the temporal interval I, it has to process more places and is the main cause of high disk access.



(a) CPU cost

(b) I/O cost

Figure 6.11: Effect of varying Temporal Interval

6.5 Conclusions

In this chapter, we study Spatial Temporal Top-k queries on Social Networks and we believe that we are the first to formalize this problem as *Geo-Social Temporal Top-k* ($GSTT_k$) query where the query retrieves top-k places (points of interest) ranked according to their spatial, social and temporal relevance to the query user. Further, we propose efficient indexing and query processing techniques and present two different approaches i.e., *Social-First* and *Spatial-First* to solve our problem and then, we propose our main algorithm called *Hybrid*. *Hybrid* technique is capable of processing social, spatial and temporal components simultaneously by utilizing a hybrid index specifically designed to handle $GSTT_k$ queries. In addition, we propose three dimensional index structures like *3DFCR-Tree and 3D Check-In R-Tree* which facilitate in efficient pruning of objects based on temporal constraints. Results of empirical studies demonstrate the effectiveness of our main algorithm (i.e., *Hybrid*).

Chapter 7

Concluding Remarks

7.1 Conclusion

In this thesis, we study various location-based queries in social networks and propose efficient indexing and query processing techniques. The summary of the contributions made to this thesis is listed below.

In chapter 3, we are the first to study T_kFP queries that retrieve nearby places popular among a particular set of users in a social network by taking into account their social and spatial relevance. We propose efficient indexing and pruning techniques and based on these, we design three algorithms to process the query including the once which is capable of processing both social and spatial components simultaneously by leveraging our specialised indexing structures. Our extensive experimental study conducted on real and synthetic data sets demonstrates the effectiveness of proposed techniques.

In-addition to this, we develop a demonstration to show the real application of *Location-Based Top-k Queries* in social networks. This demonstration enables participants to view actual output of T_kFP query containing *top-k* places

checked-in by friends in a given region.

A *top-k* query uses a scoring function that combines social and spatial scores to rank the objects. However, users must have adequate domain knowledge to be able to decide upon a good value of α. In particular, it is not easy to define a scoring function (e.g., due to incompatible attributes, different distributions of attributes, the inability of users to choose a good scoring function) [26]. Therefore, to complement our *Top-k famous places* query, we extend our work to study skyline queries which return those objects that are within given range r (i.e., $||q, p|| < r$) and are not dominated by any other object. In order to answer such queries, we compute social and spatial scores of each place and map it to a space where x-coordinate refers to spatial score and y-coordinate refers to social score to retrieve such places which are not dominated by any other place . The intuition behind using a range r is that sometimes users are not interested in places that are too far. We propose three techniques to process the query and propose efficient algorithms and indexing techniques. Furthermore, We conduct an extensive experimental study on real and synthetic data sets and compare their performance with [27] and find that our algorithms are significantly better than the competitor.

In many applications, a group of users may want to plan an activity and to find a point of interest (POI) for example, some conference attendees would like to go out for dinner together. For this purpose, we may consider their respective locations and social circles to recommend required POIs. Therefore, in our third work, we study a problem of finding top-k places considering their distance from the group of query users Q and popularity of the place among each query user $q_i \in Q$'s social connections (e.g., the number of check-ins at the place by each q's friends). To the best of our knowledge, we are the first to study the *SG-Top*$_k$ query that retrieves nearby places popular among a

particular group of users w.r.t. each query user $q_i \in Q$ in the social network. We present *Branch-and-Bound* algorithm to solve the problem followed by some proposed optimization techniques to further improve its performance. We conduct an exhaustive evaluation of the proposed schemes using real dataset and demonstrate the effectiveness of the schemes.

In this study, we add a third dimension (time) to our first work and study a problem of finding top-k places considering their distance from the query user q and popularity of the place among q's social connections during a certain period of time. Based on this, we formalize the problem as a *Geo-Social Temporal Top-k ($GSTT_k$)* query and propose three techniques to answer the query: I) Social-First, II) Spatial-First and III) Hybrid. To improve the query performance, we propose few index structures (e.g., FCR-Tree, 3D Check-Ins RTree) which enable flexible data management and algorithmic design. We conduct extensive experiments using real and synthetic data sets to demonstrate the effectiveness of the proposed algorithms.

7.2 Future Work

There are several interesting directions for future work which we briefly describe as follows:

- In this work, we have studied some *location-based* queries in social networks. In RQ1, we studied a problem of finding top-k places based on its spatial and social relevance. However, a user might be interested not only in finding top-k places but also in finding top-k *Regions of Interest*. There are many real world applications that can leverage such queries like in urban planning, marketing and many more.

  Intuitively, the goal of such query is to find the location of nearest group

of points of interest, which is different from returning the single point in the usual NN query. Thus, the primitive data type is extended from a point of interest to a region of interest. It can be represented as a particular-sized region of interest (ROI) containing multiple POIs not necessarily of the same type. Of course, the number of points constituting a region cannot be too small. Thus, we seek each ROI to be contained at least a particular number of POIs.

Score of a Region of Interest *RoI*: Given a query user q_i, a radius r, minimum number of required places in a circle $MinPOIs$, the score of a RoI is the weighted sum of its spatial score (denoted as $spatial(RoI)$) and its social score (denoted as $social(RoI)$).

Top-k Regions of Interests (T_kRI) Query: Given a set of places $P = \{p_1, p_2, ...p_n\}$ in an LBSN, a query user q, a positive number k, minimum number of places $MinPOIs$ required in each region of interest *(RoI)* and a radius r, a T_kRI query finds k locations $o \in O$ that maximizes the score ($Score(RoI)$) of each RoI using given radius r centred at o and contains at least $MinPOIs$.

- In RQ4, we considered time window as a third dimension. There can be another interesting possible direction to handle time window where the freshness rate in scoring function can also be considered. For example, we can give more priority to the recent check-ins and less to those carrying older timestamps. For example, a tourist visiting Melbourne may be interested in finding recently ongoing activities which are popular among German people. Such queries can be exploited in many real world applications such as in crime prevention, disaster management and many more.

- Often, a direct relationship between two entities in LBSNs does not suffice the requirement of getting desired results based on social criteria. For example, if a German visitor is looking for any German food restaurant popular among German people in Melbourne, and could not find any desired results, she might be interested in extending the search criteria to the friends of German people who are Australian (this time we have more than one edge. i.e., a path from user u to v in an LBSN graph). In such scenario, the social component constitutes multiple relationships. Based on this, we can further extend the social circle to extract social relevance. Such queries can be utilized any many applications, for example in health, marketing, law enforcement any many more.

  There are two types of distances in LBSNs: social distance and spatial distance. The integration of the two requires careful balance of different metrics.

  Social network is an undirected weighted graph (denoted as *G(V,E)*) and $w(u, v)$ denotes the weight of an edge between two entities which defines relationship type.

  Definition: Social Distance. Social distance between 2 entities u and v in a given graph G is a sum of weights of the shortest path between them and is denoted by $dist_{social}(u, v)$. Similarly, $dist_{social}(u, v) = w(u, v)$ if there is a direct edge between them.

Bibliography

[1] Zahy Bnaya, Rami Puzis, Roni Stern, and Ariel Felner. Social network search as a volatile multi-armed bandit problem. *HUMAN*, 2(2):pp–84, 2013.

[2] https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.

[3] https://www.yellow.com.au/wp-content/uploads/2018/06/Yellow-Social-Media-Report-2018-Consumer.pdf.

[4] Theresa Marie Bernardo, Andrijana Rajic, Ian Young, Katie Robiadek, Mai T Pham, and Julie A Funk. Scoping review on search queries and social media for disease surveillance: a chronology of innovation. *Journal of medical Internet research*, 15(7), 2013.

[5] Janaína Gomide, Adriano Veloso, Wagner Meira Jr, Virgílio Almeida, Fabrício Benevenuto, Fernanda Ferraz, and Mauro Teixeira. Dengue surveillance based on a computational model of spatio-temporal locality of twitter. In *Proceedings of the 3rd international web science conference*, page 3. ACM, 2011.

[6] https://www.business.gov.au/info/plan-and-start/develop-your-business-plans.

[7] https://edition.cnn.com/2013/09/30/us/nsa-social-networks/.

[8] Jennifer A Johnson and John David Reitzel. Social network analysis in an operational environment: Defining the utility of a network approach for crime analysis using the richmond city police department as a case study. In *International Police Executive Symposium*, 2011.

[9] https://atelier.bnpparibas/sites/default/files/google_reseaux_sociaux.pdf.

[10] Philip E Ross. Top 11 technologies of the decade. *IEEE Spectrum*, 1(48):27–63, 2011.

[11] Hans H Bauer, Tina Reichardt, Stuart J Barnes, and Marcus M Neumann. Driving consumer acceptance of mobile marketing: A theoretical framework and empirical study. *Journal of electronic commerce research*, 6(3):181, 2005.

[12] Stuart J Barnes and Eusebio Scornavacca. Mobile marketing: the role of permission and acceptance. *International Journal of Mobile Communications*, 2(2):128–139, 2004.

[13] Michael T Gastner and Mark EJ Newman. The spatial structure of networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 49(2):247–252, 2006.

[14] Michael F Goodchild and J Alan Glennon. Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, 3(3):231–241, 2010.

[15] Huiji Gao, Xufei Wang, Geoffrey Barbier, and Huan Liu. Promoting coordination for disaster relief–from crowdsourcing to coordination. In

Social computing, behavioral-cultural modeling and prediction, pages 197–204. Springer, 2011.

[16] Huiji Gao, Geoffrey Barbier, Rebecca Goolsby, and Daniel Zeng. Harnessing the crowdsourcing power of social media for disaster relief. Technical report, DTIC Document, 2011.

[17] Hussein Dia. An object-oriented neural network approach to short-term traffic forecasting. *European Journal of Operational Research*, 131(2):253–261, 2001.

[18] Moshe Ben-Akiva, Michel Bierlaire, Haris Koutsopoulos, and Rabi Mishalani. Dynamit: a simulation-based system for traffic prediction. In *DACCORS short term forecasting workshop, The Netherlands*. Citeseer, 1998.

[19] Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin, Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, and Ning Zhang. Unicorn: A system for searching the social graph. *PVLDB*, 6(11):1150–1161, 2013.

[20] `https://www.attorneygeneral.gov.au/Mediareleases/Pages/2015/FirstQuarter/20-February-2015-Combating`.

[21] Stefan Raue, Leif Azzopardi, and Chris W Johnson. # trapped!: social media search system requirements for emergency management professionals. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 1073–1076. ACM, 2013.

[22] https://www.emarketer.com/Article/Social-Network-Ad-Spending-Hit-2368-Billion-Worldwide-2015/1012357.

[23] https://www.marketsandmarkets.com/PressReleases/social-media-analytics.asp.

[24] Ammar Sohail, Ghulam Murtaza, and David Taniar. Retrieving top-k famous places in location-based social networks. In *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016, Sydney, NSW, September 28-29, 2016, Proceedings*, pages 17–30, 2016.

[25] Ammar Sohail, David Taniar, Andreas Züfle, and Jeong-Ho Park. Query processing in location-based social networks. In *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017*, pages 1379–1381, 2017.

[26] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 301–312. ACM, New York, NY, USA, 2003.

[27] Dingming Wu, Yafei Li, Byron Choi, and Jianliang Xu. Social-aware top-k spatial keyword search. In *IEEE MDM, 2014, Brisbane, Australia.*

[28] Ammar Sohail, Muhammad Aamir Cheema, and David Taniar. Social-aware spatial top-k and skyline queries. *The Computer Journal*, 62, 2018.

[29] Ammar Sohail, Arif Hidayat, Muhammad Aamir Cheema, and David Taniar. Location-aware group preference queries in social-networks. In

Databases Theory and Applications - 29th Australasian Database Conference, ADC 2018, Gold Coast, QLD, Australia, May 24-27, 2018, Proceedings, pages 53–67, 2018.

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

[31] Nikos Armenatzoglou, Ritesh Ahuja, and Dimitris Papadias. Geo-social ranking: functions and query processing. *VLDB J.*, 24(6):783–799, 2015.

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

[33] Gregory Ference, Mao Ye, and Wang-Chien Lee. Location recommendation for out-of-town users in location-based social networks. In *22nd ACM, CIKM 2013, San Francisco, CA, USA*.

[34] Kyriakos Mouratidis, Jing Li, Yu Tang, and Nikos Mamoulis. Joint search by social and spatial proximity. *IEEE Trans. Knowl. Data Eng.*, 27(3):781–793, 2015.

[35] Yerach Doytsher, Ben Galon, and Yaron Kanza. Querying geo-social data by bridging spatial networks and social networks. In *LBSN 2010, San Jose, CA, USA,*.

[36] Qian Huang and Yu Liu. On geo-social network services. In *Geoinformatics, 2009 17th International Conference on*, pages 1–6. Ieee, 2009.

[37] Weimo Liu, Weiwei Sun, Chunan Chen, Yan Huang, Yinan Jing, and Kunjie Chen. Circle of friend query in geo-social networks. In *DASFAA, Busan, South Korea, April*, 2012.

[38] De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. On socio-spatial group query for location-based social networks. In *KDD 2012, Beijing, China*.

[39] Mao Ye, Peifeng Yin, and Wang-Chien Lee. Location recommendation for location-based social networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 458–461. ACM, 2010.

[40] Mohamed Sarwat, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel. Lars*: An efficient and scalable location-aware recommender system. *Knowledge and Data Engineering, IEEE Transactions on*, 26(6):1384–1399, 2014.

[41] Huiji Gao and Huan Liu. Data analysis on location-based social networks. In *Mobile social networking*, pages 165–194. Springer, 2014.

[42] Jiwei Li and Claire Cardie. Timeline generation: tracking individuals on twitter. In *23rd International World Wide Web Conference, WWW 2014, Seoul, Korea*.

[43] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-Lee Tan, and Wen-Syan Li. Efficient location-aware influence maximization. In *SIGMOD 2014, Snowbird, UT, USA*.

[44] Nikos Armenatzoglou, Stavros Papadopoulos, and Dimitris Papadias. A general framework for geo-social query processing. *Proceedings of the VLDB Endowment*, 2013.

[45] Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD*. ACM, 2011.

[46] Yerach Doytsher, Ben Galon, and Yaron Kanza. Managing socio-spatial data as large graphs, In WWW, 2012.

[47] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 201–212, 2000.

[48] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.*, pages 71–79, 1995.

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

[50] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 154–165, 1998.

[51] Christian S. Jensen, Jan Kolárvr, Torben Bach Pedersen, and Igor Timko. Nearest neighbor queries in road networks. In *GIS*, pages 1–8, 2003.

[52] M. Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[53] Mohammad R. Kolahdouzan and Cyrus Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *Spatio-Temporal Database Management, 2nd International Workshop STDBM'04, Toronto, Canada, August 30, 2004*, pages 33–40, 2004.

[54] Cyrus Shahabi, Mohammad R. Kolahdouzan, and Mehdi Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. In *ACM-GIS*, pages 94–10, 2002.

[55] Hyung-Ju Cho and Chin-Wan Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, pages 865–876, 2005.

[56] Haibo Hu, Dik Lun Lee, and Jianliang Xu. Fast nearest neighbor search on road networks. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, pages 186–203, 2006.

[57] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, pages 43–54, 2006.

[58] Hua Lu, Xin Cao, and Christian S. Jensen. A foundation for efficient indoor distance-aware query processing. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 438–449, 2012.

[59] Xike Xie, Hua Lu, and Torben Bach Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In Jensen et al. [159], pages 434–445.

[60] Jiao Yu, Wei-Shinn Ku, Min-Te Sun, and Hua Lu. An RFID and particle filter-based indoor spatial query evaluation system. In *Joint 2013*

EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013, pages 263–274, 2013.

[61] Joon-Seok Kim and Ki-Joune Li. Location k-anonymity in indoor spaces. *GeoInformatica*, 20(3):415–451, 2016.

[62] Hans-Peter Kriegel, Peter Kunath, and Matthias Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, pages 337–348, 2007.

[63] Yuan-Ko Huang, Shi-Jei Liao, and Chiang Lee. Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction. In *SSDBM*, pages 549–557, 2008.

[64] Yuan-Ko Huang, Shi-Jei Liao, and Chiang Lee. Evaluating continuous k-nearest neighbor query on moving objects with uncertainty. *Inf. Syst.*, 34(4-5):415–437, 2009.

[65] Goce Trajcevski, Roberto Tamassia, Hui Ding, Peter Scheuermann, and Isabel F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 874–885, 2009.

[66] Cyrus Shahabi, Lu An Tang, and Songhua Xing. Indexing land surface for efficient knn query. *PVLDB*, 1(1):1020–1031, 2008.

[67] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Kai Xu, and Xuemin Lin. Surface k-nn query processing. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 78, 2006.

[68] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Qing Liu, Kai Xu, and Xuemin Lin. A multi-resolution surface distance model for k-nn query processing. *VLDB J.*, 17(5):1101–1119, 2008.

[69] Songhua Xing, Cyrus Shahabi, and Bei Pan. Continuous monitoring of nearest neighbors on land surface. *PVLDB*, 2(1):1114–1125, 2009.

[70] Andreas Henrich. A distance scan algorithm for spatial access structures. In *ACM-GIS*, pages 136–143, 1994.

[71] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot L. Siegel, and Zenon Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 215–226, 1996.

[72] Apostolos Papadopoulos and Yannis Manolopoulos. Performance of nearest neighbor queries in r-trees. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, pages 394–408, 1997.

[73] Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.*, pages 369–380, 1997.

[74] Hanan Samet, Jagan Sankaranarayanan, and Houman Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 43–54, 2008.

[75] Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 397–410, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[76] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 155–162, 2005.

[77] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In Jensen et al. [159], pages 901–912.

[78] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases - 12th International Symposium, SSTD 2011, Minneapolis, MN, USA, August 24-26, 2011, Proceedings*, pages 205–222, 2011.

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

[80] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Joining ranked inputs in practice. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 950–961, 2002.

[81] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked in-

puts. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 281–290, 2001.

[82] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3):207–221, 2004.

[83] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB J.*, 13(1):49–70, 2004.

[84] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting ad-hoc ranking aggregates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 61–72, 2006.

[85] Zhitao Shen, Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Haixun Wang. A generic framework for top-k pairs and top-k objects queries over sliding windows. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1349–1366, 2012.

[86] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[87] Muhammad Aamir Cheema, Zhitao Shen, Xuemin Lin, and Wenjie Zhang. A unified framework for efficiently processing ranking related queries. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pages 427–438. OpenProceedings, Konstanz, Germany, 2014.

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

[89] Surya Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, pages 22–29, 1999.

[90] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 419–428. ACM, New York, NY, USA, 2000.

[91] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 648–659, 2004.

[92] Giuseppe Amato, Fausto Rabitti, Pasquale Savino, and Pavel Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Trans. Inf. Syst.*, 21(2):192–227, 2003.

[93] Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 886–895, 2007.

[94] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The*

Marmara Hotel, Istanbul, Turkey, April 15-20, 2007, pages 896–905, 2007.

[95] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[96] Zhen Zhang, Seung-won Hwang, Kevin Chen-Chuan Chang, Min Wang, Christian A. Lang, and Yuan-Chi Chang. Boolean + ranking: querying a database by k-constrained optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 359–370, 2006.

[97] Jinling Jiang, Hua Lu, Bin Yang, and Bin Cui. Finding top-k local users in geo-tagged social media data. In *31st IEEE ICDE 2015, Seoul, South Korea*.

[98] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[99] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 1029–1038, 2010.

[100] Ritesh Ahuja, Nikos Armenatzoglou, Dimitris Papadias, and George J. Fakas. Geo-social keyword search. In *SSTD 2015, Hong Kong, China* [160].

[101] Mehdi Kargar and Aijun An. Discovering top-k teams of experts with/without a leader in social networks. In *Proceedings of the 20th*

ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011, pages 985–994, 2011.

[102] Mehdi Kargar and Aijun An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[103] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. Collective spatial keyword querying. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 373–384, 2011.

[104] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 155–162, 2005.

[105] Wenjie Zhang, Xuemin Lin, Ying Zhang, Muhammad Aamir Cheema, and Qing Zhang. Stochastic skylines. *ACM Trans. Database Syst.*, 37(2):14:1–14:34, 2012.

[106] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. A safe zone based approach for monitoring moving skyline queries. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 275–286. OpenProceedings, Konstanz, Germany, 2013.

[107] Michael D. Morse, Jignesh M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna,*

Austria, September 23-27, 2007, pages 267–278. ACM, New York, NY, USA, 2007.

[108] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Maximal vector computation in large data sets. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 229–240. ACM, New York, NY, USA, 2005.

[109] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings*, pages 256–273. Springer, Berlin, Germany, 2004.

[110] Chee Yong Chan, Pin-Kwang Eng, and Kian-Lee Tan. Stratified computation of skylines with partially-ordered domains. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 203–214. ACM, New York, NY, USA, 2005.

[111] Yufei Tao and Dimitris Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.

[112] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[113] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 275–286, 2002.

[114] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD Conference*, pages 467–478, 2003.

[115] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 301–310, 2001.

[116] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430. Springer, Berlin Heidelberg, 2001.

[117] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 301–310. ACM, New York, NY, USA, 2001.

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

[119] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 275–286. ACM, New York, NY, USA, 2002.

[120] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San*

Diego, California, USA, June 9-12, 2003, pages 467–478. ACM, New York, NY, USA, 2003.

[121] Tobias Emrich, Maximilian Franzke, Nikos Mamoulis, Matthias Renz, and Andreas Züfle. Geo-social skyline queries. In *Database Systems for Advanced Applications - 19th International Conference, DASFAA 2014, Bali, Indonesia, April 21-24, 2014. Proceedings, Part II*, pages 77–91. Springer, Berlin Heidelberg, 2014.

[122] Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. Group nearest neighbor queries. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 301–312. IEEE, 2004.

[123] Man Lung Yiu, Xiangyuan Dai, Nikos Mamoulis, and Michail Vaitis. Top-k spatial preference queries. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 1076–1085, 2007.

[124] Yuan Tian, Peiquan Jin, Shouhong Wan, and Lihua Yue. Group preference queries for location-based social networks. In *Web and Big Data - First International Joint Conference, APWeb-WAIM 2017, Beijing, China, July 7-9, 2017, Proceedings, Part I*, pages 556–564, 2017.

[125] Muhammad Attique, Hyung-Ju Cho, Rize Jin, and Tae-Sun Chung. Top-k spatial preference queries in directed road networks. *ISPRS Int. J. Geo-Information*, 5(10):170, 2016.

[126] João B. Rocha-Junior, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2):93–104, 2010.

[127] George Tsatsanifos and Akrivi Vlachou. On processing top-k spatio-textual preference queries. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 433–444, 2015.

[128] Muhammad Attique, Rizwan Qamar, Hyung-Ju Cho, and Tae-Sun Chung. A new approach to process top-k spatial preference queries in a directed road network. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS 2014, Dallas/Fort Worth, TX, USA, November 4, 2014*, pages 34–42, 2014.

[129] Hyung-Ju Cho, Se Jin Kwon, and Tae-Sun Chung. ALPS: an efficient algorithm for top-k spatial preference search in road networks. *Knowl. Inf. Syst.*, 42(3):599–631, 2015.

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

[131] Feifei Li, Ke Yi, and Wangchao Le. Top-k queries on temporal data. *VLDB J.*, 19(5):715–733, 2010.

[132] Tuan-Anh Hoang-Vu, Huy T. Vo, and Juliana Freire. A unified index for spatio-temporal keyword queries. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 135–144, 2016.

[133] Arif Hidayat, Muhammad Aamir Cheema, and David Taniar. Relaxed reverse nearest neighbors queries. In *Advances in Spatial and Temporal*

Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings [160], pages 61–79.

[134] Yehuda Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.

[135] Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 485–492, 2005.

[136] James McInerney, Jiangchuan Zheng, Alex Rogers, and Nicholas R. Jennings. Modelling heterogeneous location habits in human populations for location prediction under data sparsity. In *The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '13, Zurich, Switzerland, September 8-12, 2013*, pages 469–478, 2013.

[137] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat-Thalmann. Time-aware point-of-interest recommendation. In *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*, pages 363–372, 2013.

[138] Mao Ye, Peifeng Yin, and Wang-Chien Lee. Location recommendation for location-based social networks. In *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, November 3-5, 2010, San Jose, CA, USA, Proceedings*, pages 458–461, 2010.

[139] Zijun Yao, Yanjie Fu, Bin Liu, Yanchi Liu, and Hui Xiong. POI recommendation: A temporal matching between POI popularity and user reg-

ularity. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 549–558, 2016.

[140] Huiji Gao, Jiliang Tang, Xia Hu, and Huan Liu. Exploring temporal effects for location recommendation on location-based social networks. In *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*, pages 93–100, 2013.

[141] I Kamel, M Khalil, and V Kouramajian. Bulk insertion in dynamic r-trees. In *Proceedings of the International Symposium on Spatial Data Handling*, volume 4, pages 31–42, 1996.

[142] Li Chen, Rupesh Choubey, and Elke A. Rundensteiner. Bulk-insertions info r-trees using the small-tree-large-tree approach. In *ACM-GIS '98, Proceedings of the 6th international symposium on Advances in Geographic Information Systems, November 6-7, 1998, Washington, DC, USA*, pages 161–162, 1998.

[143] Rupesh Choubey, Li Chen, and Elke A. Rundensteiner. GBI: A generalized r-tree bulk-insertion strategy. In *Advances in Spatial Databases, 6th International Symposium, SSD'99, Hong Kong, China, July 20-23, 1999, Proceedings*, pages 91–108, 1999.

[144] Lars Arge, Klaus H. Hinrichs, Jan Vahrenhold, and Jeffrey Scott Vitter. Efficient bulk operations on dynamic r-trees. *Algorithmica*, 33(1):104–128, 2002.

[145] Lars Arge. The buffer tree: A new technique for optimal i/o-algorithms (extended abstract). In *Algorithms and Data Structures, 4th International Workshop, WADS '95, Kingston, Ontario, Canada, August 16-18, 1995, Proceedings*, pages 334–345, 1995.

[146] Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik Lun Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 325–334, 2011.

[147] Bin Liu, Yanjie Fu, Zijun Yao, and Hui Xiong. Learning geographical preferences for point-of-interest recommendation. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 1043–1051, 2013.

[148] Lisi Chen, Gao Cong, Xin Cao, and Kian-Lee Tan. Temporal spatial-keyword top-k publish/subscribe. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 255–266, 2015.

[149] Tobias Emrich, Maximilian Franzke, Nikos Mamoulis, Matthias Renz, and Andreas Züfle. Geo-social skyline queries. In *DASFAA*, 2014.

[150] Memcached. `http://memcached.org/`.

[151] Twitter: Real-time Geo. `http://slideshare.net/raffikrikorian/rtgeo-where-20-2011`.

[152] GeoSpatial indexes in MongoDB. `http://docs.mongodb.org/manual/core/geospatial-indexes/`.

[153] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD'84, 1984*, pages 47–57, 1984.

[154] Dingqi Yang, Daqing Zhang, and Bingqing Qu. Participatory cultural mapping based on collective behavior data in location-based social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(3):30, 2016.

[155] Ke Deng, Xiaofang Zhou, and Heng Tao Shen. Multi-source skyline query processing in road networks. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 796–805. IEEE, New York, NY, USA, 2007.

[156] Dimitris Sacharidis, Panagiotis Bouros, and Timos K. Sellis. Caching dynamic skyline queries. In *Scientific and Statistical Database Management, 20th International Conference, SSDBM 2008, Hong Kong, China, July 9-11, 2008, Proceedings*, pages 455–472. IEEE, New York, NY, USA, 2008.

[157] Mehdi Sharifzadeh and Cyrus Shahabi. The spatial skyline queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 751–762. ACM, New York, NY, USA, 2006.

[158] Yuqiu Qian, Ziyu Lu, Nikos Mamoulis, and David W. Cheung. P-LAG: location-aware group recommendation for passive users. In *Advances in Spatial and Temporal Databases - 15th International Symposium, SSTD 2017, Arlington, VA, USA, August 21-23, 2017, Proceedings*, pages 242–259, 2017.

[159] Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors. *29th IEEE International Conference on Data Engineering, ICDE*

2013, Brisbane, Australia, April 8-12, 2013. IEEE Computer Society, 2013.

[160] Ritesh Ahuja, Nikos Armenatzoglou, Dimitris Papadias, and George J. Fakas. Geo-social keyword search. In *Advances in Spatial and Temporal Databases, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings* [160], pages 431–450.