THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LOCATION RECOMMENDATION FOR MOBILE USERS IN LOCATION-BASED SOCIAL NETWORKS

GREGORY FERENCE
SPRING 2013

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Science and in Mathematics
with honors in Computer Science

Reviewed and approved* by the following:

Wang-Chien Lee
Associate Professor of Computer Science and Engineering
Thesis Supervisor

John Hannan
Associate Professor of Computer Science and Engineering
Honors Adviser

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Schreyer Honors College.

Abstract

Location-based services have become popular in the twenty-first century due to technological advances, such as mobile and online social networking. One of its key features is location recommendation, which encourages users to explore new locations. In this thesis, we propose methods for recommending locations that are interesting for the user in terms of closeness in proximity and spatial diversity in relation to the user's current location.

First, most previous research on location recommendation services in location-based social networks (LBSNs) makes recommendations without considering where the targeted user is currently located. Such services, as a result, may recommend to a user traveling out of town a place close to her hometown. In this thesis, we study the issues in making location recommendations for out-of-town users by taking into account user preference, social influence and geographical proximity. Accordingly, we propose a collaborative recommendation framework, called *User Preference, Proximity and Social-Based Collaborative Filtering* (UPS-CF), to make location recommendation for mobile users in LBSNs. We validate our ideas by comprehensive experiments using real datasets collected from Foursquare and Gowalla. By comparing two baseline algorithms (i.e., popularity-based and distance-based approaches) and conventional collaborative filtering approach (and its variants), we show that UPS-CF exhibits the best performance. Additionally, we find that when users are in town, preference derived from similar users is important while social influence becomes more important as a user is out of town.

Second, studies have shown that users prefer diversity in their recommendation results, but few works have considered spatial diversity in terms of recommending locations. In this thesis, we investigate the k-nearest diverse neighbor problem, which chooses locations that are spatially diverse as well as close in proximity to the user's current location. By fixing deficiencies in a state-of-the-art framework, we first propose the *Modified Index-Based Diverse Browsing* (Mod-IBDB) framework. In addition, we propose the *Distance-Based Diverse Browsing* (DBDB) framework that improves upon an initial solution to provide spatially diverse and nearby locations. We develop two versions based upon selecting the initial framework: *Distance-First DBDB* and *Diversity-First DBDB*. Using Foursquare and Gowalla datasets as well as synthetic datasets, we show that Mod-IBDB improves upon the performance of its predecessor. In addition, we show that DBDB methods outperform the mentioned state-of-the-art algorithms (as well as the k-nearest neighbor baseline) in terms of proximity and spatial diversity while maintaining high efficiency.

Table of Contents

List of Figures

List of Tables

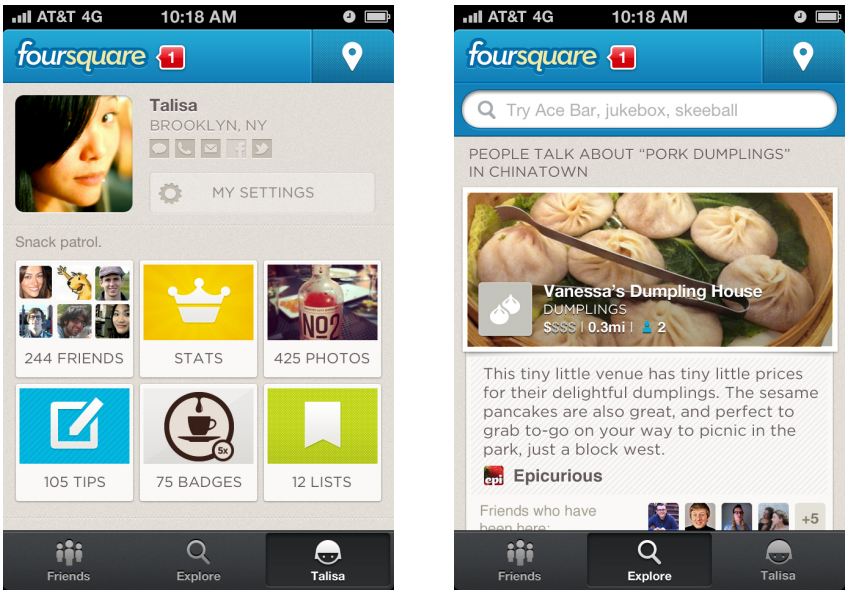
Acknowledgments

I would like to thank many people for their support throughout my college career. First, I would like to thank Dr. Wang-Chien Lee, who has served as my research advisor for the past several years. His guidance and wisdom has helped me tremendously throughout my research career. Through having him as my research advisor, I have learned a great deal that helped me complete my degrees and will serve me well in the future. In addition, I would like to acknowledge Dr. Lee's Pervasive Data Access (PDA) group, which has given insightful input to help guide my research. In particular, Dr. Mao Ye helped me tremendously while at Penn State, from the very beginning of my research career to completing research projects. Also at Penn State, Dr. John Hannan has served as an excellent thesis committee member and academic advisor throughout my academic career, answering all the questions that I had. Furthermore, I would like to thank my fellow colleagues at Penn State as well as my friends for their help and support throughout my life at Penn State. Lastly, I would like to thank my family, who has supported and loved me through my whole life and college career.

Chapter 1

Introduction

Due to the rapid development of Web 2.0, social networking and mobile technologies, *location-based social networks (LBSNs)*, such as Foursquare, Facebook Places and Gowalla, have emerged in recent years. They allow users to connect with friends, explore places (e.g., restaurants), share their locations, and upload comments, photos and videos. Different from conventional social networking services that connect people merely in the cyber world, LBSNs bring people together via cyber connections and "physical" interactions with places, e.g., a user may "check-in" to a location indicating she has visited the location.

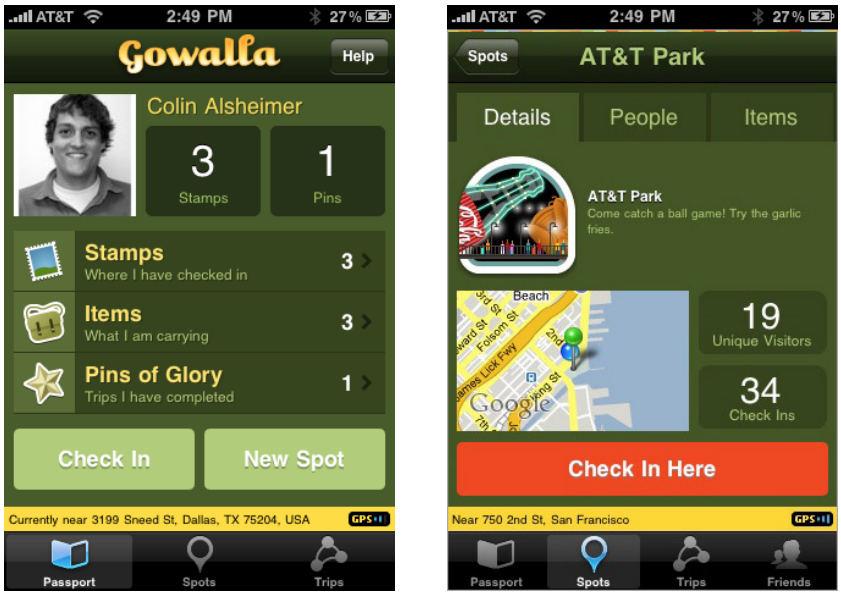


(a) User Profile[1] (b) Location Details[1]

Figure 1.1: User profile and location detail page for Foursquare mobile application

[1]Screenshots from `https://foursquare.com/about/photos`.

In addition, LBSNs include many features that keep users interested in checking into locations as well as using their services. For example, Foursquare, as shown in Figure 1.1, allows users to obtain badges for different achievements, such as visiting five different airports or over ten check-ins within twelve hours. For users who visit the same location(s) frequently, they can obtain mayor status for being the user who has checked into a location the most times in the past 60 days. Some locations, such as a restaurant, give discounts to users who are the mayor of the restaurant, which can further motivate users to visit locations and check in. Lastly, users can leave tips for different locations, such as their favorite meal at a restaurant, which will allow other users to view them when accessing that location.



(a) User Profile[2] (b) Location Details[3]

Figure 1.2: User profile and location detail page for Gowalla mobile application

In another example, Gowalla[4] (see Figure 1.2) allows users to check-in on their smartphones to show they have visited a spot (i.e., location). In addition, users are allowed to link together spots to create a trip, which others can view and complete by visiting all of the spots. To motivate users to visit spots, a user receives a stamp (i.e., icon) in their profile and may receive an item, which can be picked up or dropped at other spots for other users to pick up. Similar to badges in Foursquare, Gowalla users receive pins for certain achievements, such as visiting ten coffeeshops or creating ten spots.

To encourage mobile users to explore new locations, the *location recommendation service* is an essential function to LBSNs (just like the item recommenders to many e-commerce services,

[2]Screenshot from `http://www.leveltendesign.com/blog/colin/11-location-based-applications-your-iphone`.

[3]Screenshot from `http://smokinapps.com/iphone-discover-interesting-facts-about-an-area-and-earn-rewards/`.

[4]Gowalla was purchased by Facebook in December 2011 and shut down in March 2012

such as netflix.com and amazon.com). The goal is to recommend a list of new locations that a targeted user may be interested in visiting. The question that arises is: how do we recommended locations for users to visit? Similarly, what factors are important to consider when making location recommendations? Three factors that are important for location recommendation are:

1. **Effectiveness:** The recommended locations should be interesting for the user. Since users with similar visiting histories tend to visit similar locations, a location that a similar user has visited has a high probability of being interesting for the user. In addition, since users tend to visit places that friends go, a location that friends have visited will more likely be interesting for the user.

2. **Close in Proximity:** The recommended locations should be nearby the current coordinate (query point) of the user. Since users tend to visit locations that are closer due to the time and expenses of traveling long distances, the recommended locations should be closer to the query point of the user.

3. **Spatially Diverse:** The recommended locations should be spatially diverse in relation to the user's query point (e.g., the locations should not be close to one another). Users would prefer spatial diversity because they want more choices to make their decision for which location to visit. If the locations are located in one area since the results are not spatially diverse and the user does not want to travel there (e.g., it is a bad area of town, user wants to explore different, new areas), the recommendation will not be satisfactory for the user.

This thesis considers these factors in two research works. In the first work, we emphasize increasing the effectiveness of location recommendation results. In the second work, we consider providing location recommendation results that are close in proximity to the user's current location as well as spatially diverse.

For increasing the effectiveness of location recommendation as well as recommending nearby locations, some state-of-the-art research proposes to incorporate social and geographical influence into collaborative filtering techniques when making location recommendations [66, 67]. However, they do not consider the current location of a mobile user. Thus, regardless of where the user is located, these systems will recommend the same locations, which could possibly be far away from the user's current standing location. For example, as illustrated in Fig. 1.3, consider a target user that lives in State College, PA, USA. The user travels out of town to Los Angeles, CA (at the other side of USA) for a vacation. Based on the core idea of collaborative filtering, similar users of the target user (i.e., those who exhibit similar location visiting behaviors to the target user) are chosen to provide clues for making recommendation. Due to the geographical locality of human mobility, most of these similar users are likely to be living in the State College area than other places because they may have visited many locations in the area where the target user has also visited and thus exhibiting "similar" location visiting behaviors. As a recommendation is made by considering locations visited by the similar users (and many of them may have never visited Los Angeles), the recommended locations could be very far away and not reasonable.

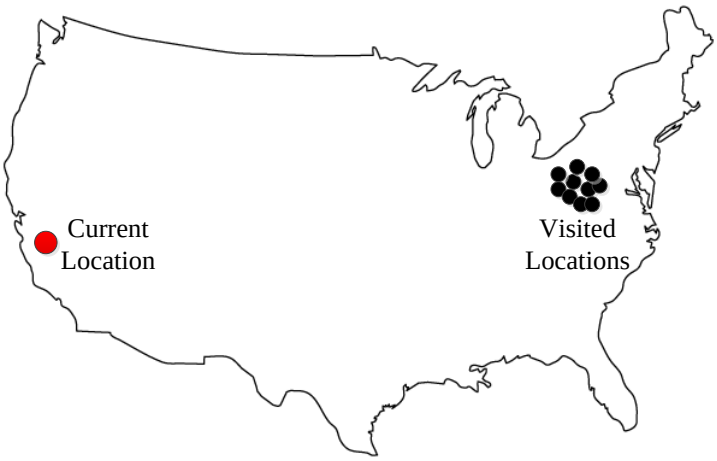


Figure 1.3: Illustration of recommendation for a user traveling a long distance

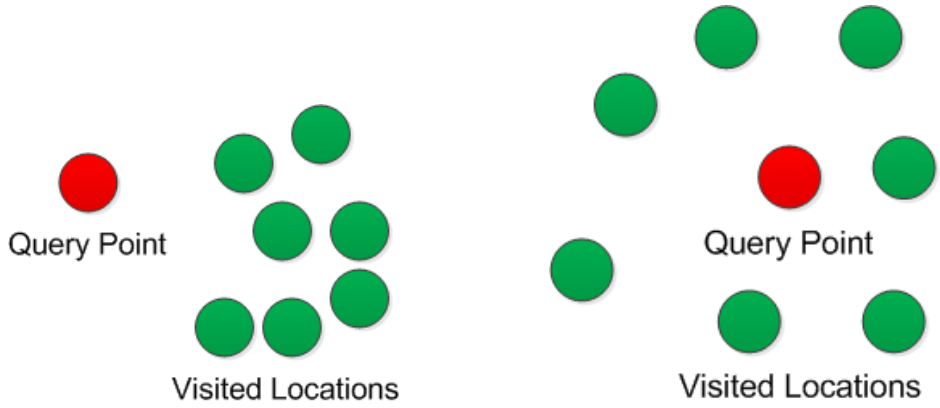
Additionally, for the recommendation algorithms that do not consider geographical proximity, they may recommend the same locations, no matter where the user is currently located. While some systems do consider the current location of the user by either filtering far away locations or adding a travel penalty for farther away locations [16, 38], they do not consider the social aspects that influence a user in an LBSN. In this thesis, we aim to study the issues in making location recommendations for out-of-town users by taking into account user preference, social connections and geographical proximity.

To solve this problem, we propose a collaborative recommendation framework, called *User Preference, Proximity and Social-Based Collaborative Filtering* (UPS-CF), to facilitate location recommendation for mobile users in LBSNs. This framework, built upon collaborative filtering, has two simple but important features: i) it filters locations that are too far away based on the current proximity of the user; and ii) it integrates similar users (who have visited many common places) and social connections (i.e., friends in LBSNs) into the collaborative filtering algorithm. This framework also allows us to investigate the different roles of similar users and social connections in location recommendation when the user is in town or out of town.

The contributions of the aforementioned research are summarized as follows.

- This work investigates the issues in employing collaborative filtering to make location recommendation for mobile users in LBSNs, with differentiation of in-town and out-of-town users.
- We propose a new recommendation framework, namely, *UPS-CF*, that considers user preference, social connections and geographical proximity.
- Through extensive experimentation with real datasets, we validate our ideas and show that UPS-CF outperforms other baseline algorithms and collaborative filtering variants.

- We show that friends play a more important role than similar users when the target user is farther away from home.

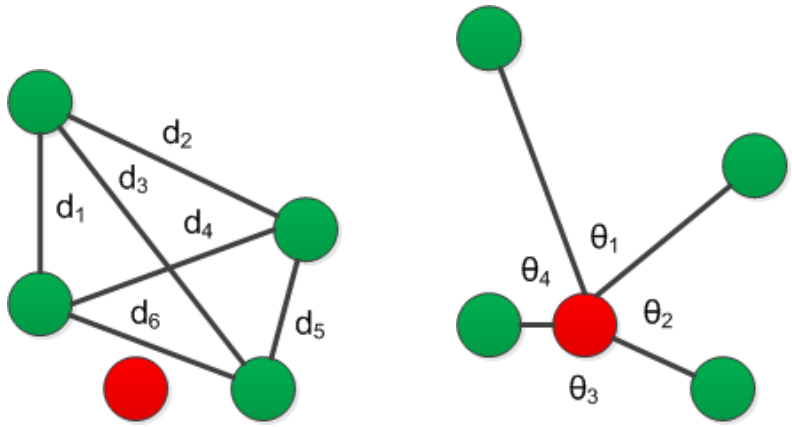


(a) Not Diverse (b) Diverse

Figure 1.4: Example of spatially non-diverse and diverse recommended locations

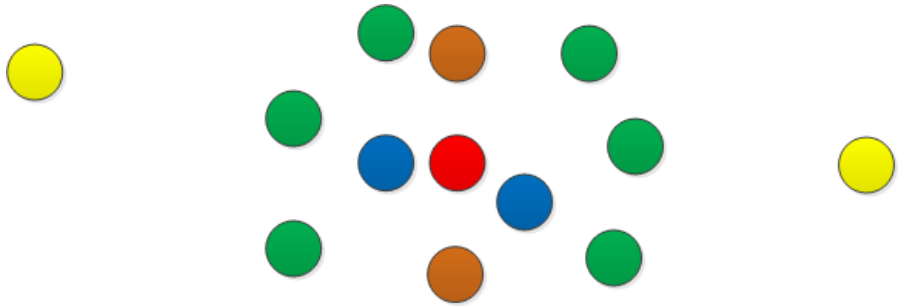
For choosing locations that are close in proximity to the user's query point as well as spatially diverse, we look into solving the k-nearest diverse neighbor problem, which involves choosing locations that are nearby the query point (can be considered relevancy for location recommendation) such that the recommended locations are diverse. Consider the example in Figure 1.4. The locations in the left example are not diverse since the locations are nearby and in the same direction in relation to the query point while the locations on the right example show spatially diverse locations since pairwise locations are relatively far apart and the locations are in different directions in relation to the query point. This problem of non-diverse results can occur in collaborative filtering (and other recommendation algorithms) since it tends to cluster recommended locations nearby other visited locations [67]. With this, there are many motivating reasons to consider spatial diversity in results. Having locations in one area makes the assumption that the user wants to visit that specific area. This could be incorrect for several reasons, such as if the user believes it is a bad area of town or has a pre-conceived notion for not visiting the area. For a user that wants to tour the city/surrounding area, she will want recommendations in different parts of the town instead of clustered in one area. Moreover, users may prefer to have a diverse set of choices before they decide where to go. Lastly, recent research has shown that users prefer diverse recommendation results, even if the effectiveness of the results is lower. Specifically for spatial diversity, Tang et al. shows that for image search results with images having geographical information, users would rather be recommended spatially diverse images [57].

For the k-nearest diverse neighbor problem, two different types of diversity can be considered: *distance diversity* and *angular diversity*. Distance diversity, as shown in Figure 1.5a, compares the distance between pairwise locations, so the farther apart the locations are, the larger the diversity. Angular diversity, as shown in Figure 1.5b, compares the direction (e.g. north, south)



(a) Distance Diversity (b) Angular Diversity

Figure 1.5: Comparison of distance diversity and angular diversity



(a) Low Angular Diversity/High Distance Diversity

Figure 1.6: Example of choosing closest two locations (blue), two locations with highest distance diversity (brown), and two locations with highest angular diversity (yellow)

of the locations in relation to the query point, with locations in varying directions having larger diversity. Even though distance diversity is the most popular diversity metric, it does not work well with the k-nearest diverse neighbor problem. Since distance diversity chooses locations that are farther apart while choosing locations close in proximity requires choosing locations that are close together around the query point, a contradiction occurs, which is visualized in Figure 1.6. To our best knowledge, two frameworks exist to solve the k-nearest diverse neighbor problem. The first framework uses distance diversity [24, 28], which suffers from the above mentioned problem with choosing locations. The other state-of-the-art system is the Index-Based Diverse Browsing (IBDB) which considers the k-nearest diverse neighbor problem with angular diversity [34, 35], but their algorithm has several deficiencies that negatively impact the result of their algorithm. In the second part of this thesis, we aim to develop alternative algorithms that provide better results in terms of proximity to the query point and spatially diversity while still being efficient.

For solving the k-nearest diverse neighbor problem, we first propose the Modified Index-Based

Diverse Browsing (Mod-IBDB) algorithm to fix deficiencies in the algorithm. In addition, we develop the Distance-Based Diverse Browsing (DBDB) framework to solve the k-nearest diverse neighbor problem. The idea of the DBDB framework is based upon search algorithms, which start with an initial solution and then attempt to swap candidate locations into the solution to improve the proximity to the query point and/or spatial diversity. We provide two methods - Distance-First Distance-Based Diverse Browsing (Dist-DBDB) and Diversity-First Distance-Based Diverse Browsing (Div-DBDB) - for choosing the initial set of locations. Dist-DBDB chooses the closest k locations while Div-DBDB uses a heuristic to choose a diverse selection of k locations.

The contributions related to the k-nearest diverse neighbor problem are below.

- We make modifications for performance improvements to the IBDB framework to create the Mod-IBDB framework.
- We propose a new framework for k-nearest diverse neighbors, *DBDB*, that chooses an initial solution and considers locations in increasing order from the query point for replacement operations to improve the solution.
- We create the Partition Diversity Metric for angular spatial diversity, which calculates the variance from "perfect" diversity (i.e., evenly spread around the query point).
- Confirming our intuition, the modifications for Mod-IBDB allow for it to outperform IBDB, especially when proximity to the query point is more important than spatial diversity.
- Through extensive experimentation with Foursquare and Gowalla datasets as well as synthetic datasets, we validate our ideas and the DBDB framework outperforms other k-nearest diverse neighbor algorithms and baseline algorithm while still being efficient.

1.1 Research Questions and Approach

For this thesis, the main goal is to shed some light in answering the following main research question:

How do we choose locations for mobile users in location-based social networks such that the locations are interesting for the user, close in proximity to the query point, and spatially diverse, no matter where the user is currently located?

To study this main question, we can break it into smaller sub-questions. Below are the sub-questions along with the details of how/where the questions are answered.

1. *What are the current state-of-the-art techniques that are used for recommendation?*

Section 2.2 includes a literature review of the different state-of-the-art recommendation techniques, including algorithms that are specific for recommending locations. In addition, we detail an in-depth review of user-based collaborative filtering in Section 3.3.2.

2. *How does the spatial nature of locations affect state-of-the-art recommendation techniques, such as user-based collaborative filtering?*

   Experiments are included in Section 4.2 to show how the performance of user-based collaborative filtering degrades as the user moves away from her home region. In addition, Section 4.3 discusses the mobility of users.

3. *What factors can be considered to increase the effectiveness of a location recommendation algorithm?*

   To show these factors, we perform a data analysis on Foursquare and Gowalla data in Chapter 4. We show how the mobility of users as well as similar users and friends play roles into which locations users visit.

4. *How can we combine these factors into a new state-of-the-art location recommendation algorithm?*

   In Chapter 5, we introduce the User Preference, Proximity and Social-Based Collaborative Filtering framework. We show in detail how the algorithm employs similar users and friends to find the top-k users for a target user as well as the proximity constraint to recommend nearby locations. In addition, an example calculation is shown to further detail how the algorithm performs.

5. *How do we measure the effectiveness of previous state-of-the-art recommendation algorithms versus our newly formulated location recommendation algorithm?*

   In Section 7.1.1, we explain the evaluation process for the experiments. Then, we use precision in Sections 7.1.3 and 7.1.4 to compare the effectiveness of previous state-of-the-art recommendation algorithms (and several baseline algorithms) against the User Preference, Proximity and Social-Based Collaborative Filtering framework.

6. *How do we recommend spatially diverse locations while still recommending locations that are close in proximity to the user?*

   Chapter 6 details several additions for solving the k-nearest diverse neighbor problem. Due to deficiencies in the Index-Based Diverse Browsing algorithm, we modify the algorithm to increase its performance. In addition, we develop the Distance-Based Diverse Browsing framework, which first chooses the k locations and performs replacement operations with locations to increase an objective function based on diversity and proximity to the query point. As a result of how we choose the initial selection of locations, we introduce the Distance-First Distance-Based Diverse Browsing and Diversity-First Distance-Based Diverse Browsing algorithms.

7. *How do we measure spatial diversity of recommended locations as well as compare the combination of spatial diversity and proximity to the query point?*

   We introduce the metrics used in our evaluation, including a new spatial diversity metric for angular diversity that solves deficiencies of other metrics used in other papers, in Section 7.2.2. Applying the evaluation process for experiments detailed in Section 7.2.1, we compare various algorithms that solve the k-nearest diverse neighbor problem, including the algorithms introduced in Chapter 6, in Section 7.2.4.

1.2 Thesis Structure

The remainder of this thesis is organized as follows. In Chapter 2, we perform a literature review of relevant works. In Chapter 3, we introduce preliminary topics that are important for the remainder of the paper. In Chapter 4, we show how social friends, similar users and geographical proximity are important factors for location recommendation in LBSNs. In Chapter 5, we present our location recommendation algorithm and in Chapter 6, we present our techniques for solving the k-nearest diverse neighbor problem. In Chapter 7, we report our findings in an experimental study to validate our algorithm designs for effectiveness as well as diversity using synthetic datasets as well as datasets collected from Foursquare and Gowalla. Lastly, in Chapter 8, we conclude this study and point out future directions.

Chapter 2

Literature Review

Here we review relevant research on social networks/LBSNs in three categories: i) data analysis on social networks, ii) recommendation techniques, and iii) diversifying results.

2.1 Data Analysis

There exists many studies surrounding the analysis of data in social networks. Java et al. study geographical and topological properties to answer the question of why people use microblogging services, such as Twitter, which can allow microblogging services to tailor new features to retain users [29]. In their findings, they show that people typically have one of a few intentions, such as sharing information and conversation. Another study explores the social networking sites YouTube, Orkut, Flickr, and LiveJournal in order to analyze the structure of the social network [43]. The authors show interesting characteristics, such as the in-degree and out-degree of a node (person) being equal, as well as confirm several properties of online social networks (i.e. power-law and small-world). Kossinets et al. explore the evolution of social networks using a dynamic social network based off e-mails sent at a large university [33]. They discover that the organizational structure of the university as well as network topology affect network evolution. In addition, the average properties of the social network tend to reach an equilibrium state even while individual properties fluctuate. Also, the use of the co-authorship networks of scientists is studied to discover the evolution and topology of the social network created [9]. Even though these papers perform a data analysis on social networks, these works differ from our work since they do not deal with the connections to locations that exist in an LBSN.

Recently, many research works on analyzing geographical properties of social networks have been reported. Eagle et al. analyze call logs and geographic information of mobile phone users to compare self-reported data versus data collected from the phones [19]. They perform experiments that test whether recency (recent events are easier to remember) and salience (prominent events are easier to remember) affect a user's ability to report average behavior. Ludford et al.

investigate how different location types affect users who are sharing location information [41]. Among the experiments completed, they analyze what types of locations people tend to share, what location-related information do people share, and how useful is the information for others.

For LBSNs, it becomes obvious that location is an essential part of social networks, especially for mobile users. Cho et al. discuss how social relationships and periodic behavior shape user movements by using a dataset collected from Gowalla [15]. Their experiments show that short-ranged travel is more influenced by spatial and temporal properties while long-ranged travel is more influenced by friends. Scellato et al. analyze the socio-spatial properties across three popular LBSNs: Brightkite, Foursquare, and Gowalla [53]. The three LBSNs exhibit similar spatial features and heterogeneity in socio-spatial behavior of users exists when considering the distance between social ties and triangles. Li et al. provide a large-scale quantitative analysis of user data in Brightkite [39]. The authors are able to classify users by mobility patterns (e.g. home users and home-vacation users) and behavior groups (e.g. active and inactive). Lastly, Scellato et al. analyze how users connect with friends and checked-in locations. The work shows that distribution between friends and check-ins differ, which is likely because of the physical constraint that exists on check-ins but not friends [52]. Our work is unique from all of the above papers because our analysis aims to explore how factors may help in making location recommendations.

2.2 Recommendation Techniques

In this section, we review the main types of recommender systems as well as explain related work in location recommender systems.

2.2.1 Content-Based vs. Collaborative Filtering

There are typically two types of recommendation systems (or recommender systems): content-based and collaborative filtering [5]. Content-based recommendation systems recommend items (e.g., movies, products, locations) that are similar to items that the user has preferred in the past [5]. Similar items are typically determined based on the textual information of the items. For example, a book recommender system will contain information about books such as title, author, genre, and brief summary. If a user has previously read books that have the genre non-fiction or the author Stephen King, the recommender system will recommend other books that have genre non-fiction or author Stephen King. This method allows users to receive recommendation for any item (no matter if the item has been preferred by a previous user), but suffers from several problems, such as its inability to make a recommendation for a new user (cold-start problem) and requirement of items having textual information.

Collaborative filtering recommendation systems make recommendations based upon similar users' preference [5, 8]. This uses the belief that similar users will tend to prefer similar products. For our book recommender system, the history is saved for which books each user prefers. If we are making a recommendation for user u and another user v has many commonly read books, a

book that v has read and u has not read will have a higher probability of being recommended. In addition, collaborative filtering methods can be further broken down into two categories: memory-based (use heuristics based on histories of preferences of users and items) and model-based (use ratings to learn a model, which is then used for recommendation) [5]. Collaborative filtering can generate recommendations without items needing textual information, but suffers from the cold-start problem, data sparsity, and the fact that new items without any user ratings cannot be recommended. Lastly, in addition to considering similar users, memory-based collaborative filtering methods for social network have been introduced that also consider social friendship in addition to similar users to make a recommendation [5, 8, 31, 51, 65]. However, the works listed do not deal with location recommendation, (i.e., they do not consider geospatial features of locations).

2.2.2 Location Recommendation

Recently, due to the growing popularity of mobile devices, the research momentum on location recommendation has increased [11, 16, 26, 38, 56, 66, 67, 73, 74]. Horozov et al. study the use of collaborative filtering to recommend restaurants for mobile users [26]. The authors modify the collaborative filtering algorithm to have a scalable system that recommends a nearby restaurant that is interesting for the user with low latency. In another paper, the modified collaborative filtering algorithm recommends shopping sites based off visiting history, current GPS information, and user's typical shopping routes [56]. Zheng et al. use other users' data and applies it to collaborative filtering methods to provide a recommendation of locations to visit, even when little information is known about the user [73]. In addition, another paper models users' location and activity history from GPS data to find location features and activity correlations that are used as additional attributes in collaborative filtering [74]. However, the majority focuses on GPS data or mobile environments [26, 56, 73, 74], without considering aspects of LBSNs, such as social influences.

Only recently, a few research works have started to investigate location recommender systems for LBSNs [11, 16, 38, 66, 67]. To our best knowledge, Ye et al. are the first ones to study location-based recommendation algorithms for LBSNs [66]. Instead of considering all users for picking the top-m users, the authors introduce friend-based collaborative filtering algorithms that only consider friends or friends that satisfy certain geographic characteristics. Unlike our paper, its emphasis is mainly on the efficiency of the algorithm. In addition, the study of the geographical characteristics of users assumes that the user's current coordinate is her home location. Also, users are constantly moving and may not be near their home location when a recommendation is made, which is not considered in this paper. An additional study proposes a collaborative filtering algorithm for location recommendation in LBSNs [11] using matrix factorization [32], which is a latent factor model. The authors perform experiments to show the feasibility of using the method for location recommendation for Gowalla. However, the paper does not consider the social aspects that are inherent in a LBSN or the current coordinate of the user when the user

makes a recommendation.

Chow et al. provide several location-based social networking services, including location recommendation [16]. The paper creates a SQL query that filters locations that are farther than a certain threshold from the query point of the user. For ranking the locations within the distance threshold, the paper only states that a technique, such as collaborative filtering, can be used, which does not use the social aspects in LBSNs. Levandoski et al. consider spatial and non-spatial aspects of users, ratings, and items when making a recommendation [38]. Their recommendation algorithm uses a partitioning technique of user's locations for system scalability and adds a travel penalty for spatial items (such as locations) for farther away locations. Again, social aspects are not considered.

Ye et al. delve into location recommendations by analyzing the geographical influences among locations and proposes a unified framework that combines user preference, social influence, and geographical influence [67]. The geographical influence uses the phenomenon of geographical clustering of a user's visited locations to provide a better recommendation. However, nothing proposed in the paper factors the current coordinate of the user into account. Thus, no matter where the target user is located, the algorithm will recommend the same locations to the user. As stated in the Introduction section, our research work for increasing the effectiveness for recommendation algorithms aims to study the issues in making location recommendations for *out-of-town users* and investigate the roles of friends and similar users for in-town and out-of-town scenarios.

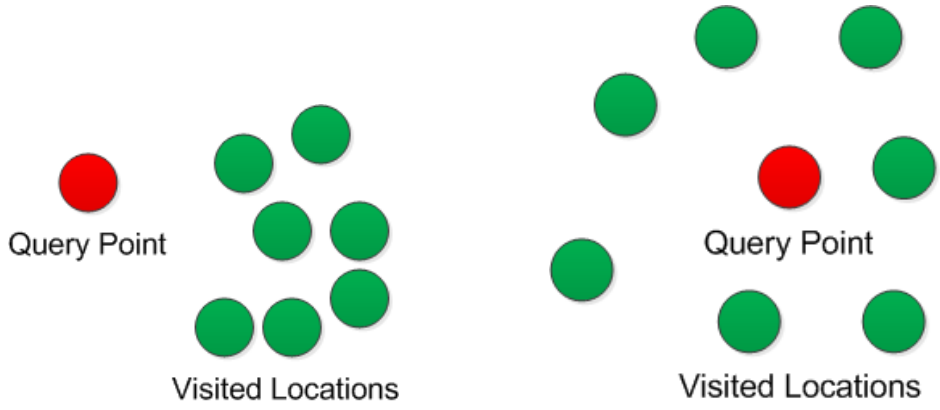
2.3 Diversity

In this section, we first introduce the different types of diversity followed by related work in search, queries and recommendation. Lastly, we review works that solve the k-nearest diverse neighbor problem.

2.3.1 Distance Diversity vs. Angular Diversity

According to the Merriam-Webster dictionary [1], diversity is the "condition of having or being composed of differing elements." For example, a restaurant and a museum could be considered diverse since these two types of locations vary while two restaurants could be considered non-diverse since they are the same type of establishment. Looking more specifically into restaurants, a Greek and an Italian restaurant could be considered diverse while two Italian restaurants could be considered non-diverse. The diversity in these examples is based upon the description of the locations. On the other hand, spatial diversity deals with diversity with the geographic details of locations (i.e., geographical coordinate).

Figure 2.1 shows an example of visited locations of a user that are non-diverse and diverse in relation to the query point. There are multiple interpretations as to why this is true. First, a location in Figure 2.1a is typically close to other locations while a location in Figure 2.1b is



(a) Not Diverse (b) Diverse

Figure 2.1: Visualization of spatially non-diverse and diverse recommended locations

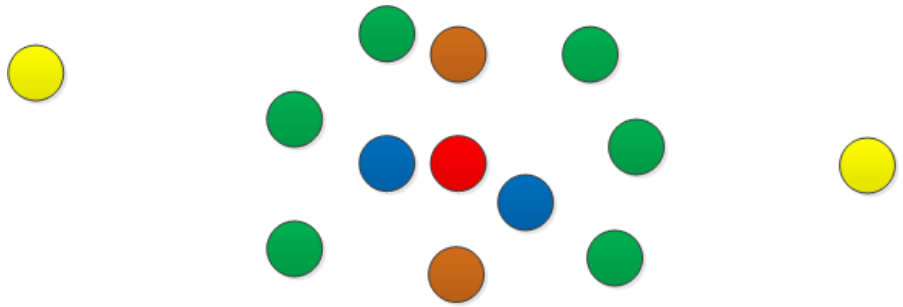
typically far away from other locations. This logic uses distance diversity, which tries to maximize the physical distance between locations. Also, locations in Figure 2.1a are located in the same direction (i.e. east or southeast) compared to the query point while locations in Figure 2.1b are located in varying angles around the query point. This refers to angular diversity, which tries to spread locations at different angles around the query point.

(a) Low Angular Diversity/High Distance Diversity (b) High Angular Diversity/Low Distance Diversity

Figure 2.2: Example with varying diversity values

Overall, distance and angular diversities are just two ways for quantifying diversity. In Figure 2.1, both diversity techniques would give 2.1a a low diversity score and 2.1b a high diversity score. However, the two techniques can provide drastically different results. In Figure 2.2a, the angular diversity would be low since the two locations (green) are at the same angle in relation to the query point (red) while the distance diversity can be large if the distance between the two locations is far apart. However in Figure 2.2b, the angular diversity is large since the locations are on opposite sides of the query point while the distance diversity can be small because the two locations are close together.

There are advantages and disadvantages to using each type of diversity. Angular diversity requires the use of a query point, which makes it harder to use for indexing to make the calculations of diversity quicker. Since distance diversity is just based upon pairwise distance between chosen locations, the query point is not necessary. Distance diversity prefers locations that are as far away as possible because it maximizes the distances between pairwise locations. However,



(a) Low Angular Diversity/High Distance Diversity

Figure 2.3: Example of choosing closest two locations (blue), two locations with highest distance diversity (brown), and two locations with highest angular diversity (yellow)

since the k-nearest diverse neighbor problem chooses locations that are closer to the user since users do not want to travel far, this provides a contradiction. This phenomenon is shown in Figure 2.3. With this last problem in mind, angular diversity will be used in the paper and all future references to diversity refer to angular diversity, unless otherwise stated.

2.3.2 Search and Query Diversity

When a person uses a search engine, recommender system, etc. to obtain results, it may be ambiguous for exactly what the user wants. For example, when the user searches for "pirates" using a search engine, it is ambiguous whether the user wants information about people who pillage ships, people who steal software, or the American baseball team. Instead of returning results for one of the above definitions, the search engine can return results about all three definitions of pirates, so the user will receive the necessary information, no matter which version of "pirates" she is searching. This topic has widely been studied for search engines and queries for non-spatial items [6, 7, 13, 14, 17, 18, 40, 46, 47, 50, 60, 61, 62, 63, 64, 72]. However, these papers explore distance diversity and do not consider spatial aspects, such as choosing locations that are diverse and close to a query point, as in the k-nearest diverse neighbor problem.

In addition, there have been a few works that have worked with diversifying results of spatial objects [45, 57, 58, 59]. Van Kreveld et al. study the problem of ranking methods to provide diversity to textually and spatially similar documents in geographic information retrieval [58, 59]. They explore various distance and angular diversity metrics in the paper. However, all of their algorithms have to visit all n points in the dataset (the most efficient algorithm is $O(n * \log(n))$), which is not practical for online calculations. Paramita et al. propose different spatial diversity algorithms that are used in image search result diversification [45]. The algorithms achieve high diversity while not significantly affecting precision, but the algorithms focus on distance diversity and do not consider the proximity to the query point. Lastly, Tang et al. add a user preference study with Amazon Mechanical Turk to study spatial diversity in image search results [57]. The results show that images with higher spatial diversity have higher user preferences. This motivates us to investigate increasing spatial diversity in our results. However, the paper does

not consider other spatial features, such as proximity to the query point, in its results.

2.3.3 Recommendation Diversity

For a recommender system, it may also be advantageous to recommend items that are diverse. This is shown in the example of choosing different types of restaurants. In addition, we can consider other characteristics when calculating diversity, such as price (non-spatial) and location (spatial). The non-spatial form of diversity has been widely studied in recommendation frameworks [2, 3, 4, 12, 21, 22, 27, 44, 55, 68, 70, 71, 75], but this is not applicable to us since we are considering spatial diversity. Recently, a paper considers the spatial diversity for recommending locations [20]. Their goal is to recommend locations that have high relevance (i.e., score from another algorithm, such as collaborative filtering) and spatially diverse. Since the paper uses distance diversity, most diverse locations are farther away from the query point, which contradicts the notion that people will not travel far to visit recommended locations.

2.3.4 K-Nearest Diverse Neighbor Query

Lastly, we look into the k-nearest diverse neighbor query, which attempts to find k locations that are both close to a query point and "diverse." The combination of proximity to the user and angularly diverse in respect to the query point have been studied [24, 28, 34, 35, 36]. Lee et al. introduce the problem of finding the nearest spatial objects that completely surround the query point [36]. With this problem, they propose two methods to efficiently solve the problem. However, the problem considers rectangles and other two-dimensional objects instead of points and the algorithm may return more or less than k spatial objects, which conflicts with the k-nearest diverse neighbor definition.

To our knowledge, Harista and Jain et al. are the first authors to consider the k-nearest diverse neighbor problem [24, 28]. For diversity, the authors of these papers create a diversity function that is similar to distance diversity, with the definition of distance between two locations being modified based off the Gower coefficient. The first algorithm they give, called Intermediate Greedy, accesses locations in increasing distance from the query point one at a time and adds the location to the chosen set if it is diverse with respect to each other location already chosen. In addition, the Buffered Greedy method executes the Intermediate Greedy method first, and then makes swaps based upon a certain criteria. The results show that the Buffered Greedy algorithm almost always chooses the optimal answer. However, since the diversity function uses a variation of distance diversity, it differs from our paper.

Kucuktunc et al. explore the k-nearest diverse neighbor problem using angular similarity [34, 35]. First, the authors introduce geometric diverse browsing methods based off Voronoi diagrams and Delaunay triangulations to choose locations. These methods involve preprocessing to create the necessary data structures, which does not work for locations in LBSNs since the database is dynamic. In addition, they introduce the Index-Based Diverse Browsing method, which works similar to distance browsing for k-nearest neighbor using an R-Tree (more detail of

the algorithm in Section 3.3.6). This method is more practical for a dynamic database since it just uses an R-Tree, but it contains inefficiencies and an incorrect calculation that we plan to investigate. Our research aims to improve upon the state-of-the-art by creating a better algorithm in terms of effectiveness (locations close to query point and diverse) and efficiency (disk accesses).

Chapter 3

Preliminaries

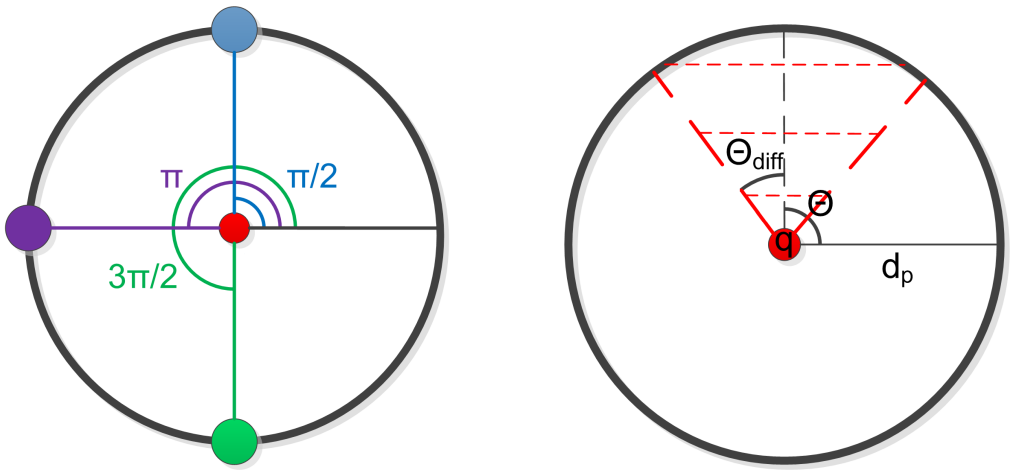
In this section, we first introduce terminology that will be used throughout the thesis and formulate the problems we are trying to solve. Then, we review user-based collaborative filtering, which is the basic algorithm behind the recommendation algorithm we develop. Lastly, we review angular spatial diversity metrics and techniques for adding spatial diversity into recommendation results.

3.1 Terminology

To help understand, below is a list of terms that are used throughout the paper.

- *Location* - A location (also known as a place or point-of-interest) is an object in an LBSN that can be visited. Locations are physically located somewhere over the map, i.e., they are always associated with a coordinate, such as (latitude, longitude).
- *Check-in* - A check-in is a record in an LBSN that a user has visited a location. This can also contain other information, such as a timestamp. A user is typically allowed to have multiple check-ins for the same location.
- *Location Recommendation System* - A location recommendation system or location recommender system is an algorithm that chooses locations for a user. It is advantageous for a location recommendation system to recommend locations that a user would likely want to visit in the future.
- *Prediction Value* - The prediction value for a user u and location l is a normalized score that estimates the willingness for u to visit l in the future. This term is used in this paper with the score that a collaborative filtering algorithm gives for location when making a recommendation for a location. The higher the prediction value, the more likely the collaborative filtering algorithm will recommend the location for the user.

- *Target User* - The target user is the user that a location recommendation system is providing locations for.
- *Candidate Location* - A candidate location is a location that is being considered by a location recommendation system.
- *Similar User* - A user v is considered similar to user u if u and v share some commonly visited locations.
- *Top-m Users* - For a user u, the top-m users of u are the m users that are most similar to u. This is utilized in the user-based collaborative filtering algorithm.
- *Home Region* - A home region for a user u is a physical region on a map that contains many locations that u has checked into.
- *Query Point* - A query point for a location recommendation system (often shown as a red circle in figures) or the current coordinate of the user is the user's current position on a map when she receives recommended locations.
- *Proximity Constraint* - The proximity constraint d_p in a location recommendation system is a number such that all locations that are farther than d_p distance from the query point are automatically pruned.



(a) Angle (b) Sector

Figure 3.1: Definition of angle and sector

- *Angle* - The angle (as shown in Figure 3.1a) between a query point q (red circle) and another point or location p (blue purple or green circle) is the angle formed from a line that extends from q in the positive x-axis direction to $\overline{qp}$. For example, the angle between query

point q and a point directly east of q is 0, the angle between q and a point directly north of q is $\frac{\pi}{2}$, the angle between q and a point directly west of q is π, and the angle between q and a point directly south of q is $\frac{3\pi}{2}$.

- *Sector* - A sector or circle sector (as shown in Figure 3.1b), defined as a tuple $(q, \theta, \theta_{diff}, d_p)$ is a portion of a circle (i.e., "piece of a pie") centered at point q with radius d_p that includes the area within the circle between the angles of $\theta - \theta_{diff}$ and $\theta + \theta_{diff}$.

3.2 Problem Formulation

Here, we introduce the location recommendation and k-nearest diverse neighbor problems, which will be studied in detail throughout the paper.

3.2.1 Location Recommendation

Let L be the set of all locations in a location-based social network. Let u be the target user and let L_u be the locations that u has previously visited. The goal of a location recommendation algorithm is to choose a set of locations R, such that $R \subseteq L \setminus L_u$ and $|R| = k$, where k is the number of locations the algorithm should recommend. To appease users, the locations in R should be interesting to u, and we also argue that the locations R should be spatially diverse.

3.2.2 K-Nearest Diverse Neighbors

Let L be the set of all points or locations in a dataset. Given a query point q, the goal is to choose (or recommend) k locations such that the locations are close in proximity to q and "diverse" with each other. More information on measuring the diversity of locations is located in Section 3.3.3 and the metric used to measure the diversity for our experiments is introduced in Section 7.2.2.2. Since it is a common belief that users like to have location recommendation results that are close in proximity to the current coordinate of the user (query point) and diverse, solving the k-nearest diverse neighbor problem can provide good recommendation results.

Formally, let λ be a control parameter that represents the ratio between distance and diversity. Our goal is to choose a set of locations R (with $|R| = k$) such that

$$\max_{R \subseteq L} \ \lambda * \mathrm{Div}(q, R) + (1 - \lambda) * \mathrm{Rel}(q, R). \tag{3.1}$$

The Div function represents the spatial diversity score, with the larger score meaning the chosen locations are more diverse. The Rel function represents the relevancy of the locations, which is a measure of distance such that a larger score means the locations are closer in proximity to the query point. The relevance and diversity metrics used in our evaluation are described in Section 7.2.2.

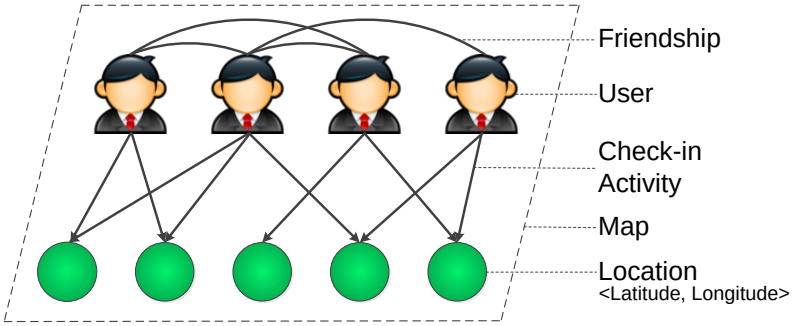


Figure 3.2: Graph representation of users and locations in a LBSN

3.3 Background

This section explores the necessary background information for the thesis, including data in LBSNs, user-based collaborative filtering, angular diversity metrics, R-Tree, k-nearest neighbor query, and Index-Based Diverse Browsing.

3.3.1 Location-Based Social Network Data

There are two essential entities in a LBSN: *users* and *locations*. The interactions among users and locations are illustrated in Figure 3.2, where users and locations are presented by nodes. The edges connecting users show social connections among users in the LBSN while the edges connecting users and locations present the check-in activities of users to the corresponding locations. In this section, we first describe the relationship between a user and a location and then detail the relationship between users.

3.3.1.1 User-Location Relationship

The main function of LBSNs is that users check-in to locations to signify that they have visited the location. This allows other users to see which location a particular user visited (as long as privacy settings allow this), which may give ideas for locations to visit. The relationship between users and locations can be defined by a bipartite graph $G = (U \bigcup L, E)$, where U is the set of all users and L is the set of all locations in a LBSN. An edge (u, l) for $u \in U$ and $l \in L$ is created if u has checked into l. Some definitions allow for a weight to be placed on the edge that is equal to the number of times u has visited l, but this is ignored in this thesis. Therefore, all the edges that connect to a location are all the users that have visited that location, and all the edges that connect to a user are all the locations a user has visited. It is interesting to investigate the two-hops of a user u, which are the users that share at least one visited location with u. Similarly, the two-hops of a location l are the locations that have been visited by the same users that visited l. This can help us explore similar users to a user or similar locations to a location.

3.3.1.2 User-User Relationship

(a) Foursquare[1] (b) Gowalla[2]

Figure 3.3: Friend details page for Foursquare and Gowalla mobile applications

In LBSNs, users are allowed to establish friendship relationships with other users that they discover through the LBSN. Figure 3.3 shows details of check-ins that friends have completed for Foursquare and Gowalla. This allows users to follow the actions and check-ins of their friends, which can influence which locations a user may visit in the future.

The friendships can be represented in a graph as follows. Letting U be the set all users in the LBSN, let F_u represent the friends (which are other users) of user $u \in U$. We can construct a graph $G = (V, E)$, where each $v_u \in V$ represents a $u \in U$. For each user u, we add an edge (v_u, v_w) into E for every user $v_w \in F_u$. If the LBSN requires friendships between users to be mutual, (u, v) can be an undirected edge; otherwise, (u, v) should be a directed edge. Therefore, all vertices that having an incoming edge from v_u represent the friends of u.

3.3.2 User-Based Collaborative Filtering

This section covers the user-based collaborative filtering algorithm as well as an example calculation to show how the algorithm executes.

[1]Screenshot from `https://foursquare.com/about/photos`.
[2]Screenshot from `http://smokinapps.com/iphone-discover-interesting-facts-about-an-area-and-earn-rewards/`.

3.3.2.1 Algorithm

User-based collaborative filtering is a well-received technique for item recommendations, which can be adopted for location recommendations by treating locations as items. As a result, it recommends locations for a target user in accordance with location visiting behaviors of "similar users" (i.e., other users with similar visiting histories in terms of commonly visited locations). Let U be the user set and L be the location set in an LBSN. The check-in activity for a user $u \in U$ and location $l \in L$ is denoted as $c_{u,l}$, where $c_{u,l} = 1$ represents that u has a check-in at l and $c_{u,l} = 0$ otherwise. Using the user check-in activities of locations, user-based collaborative filtering derives a user's implicit preference over a specific location as a score (between 0 and 1), denoted by $p_{u,l}$, which ranks how likely a user u would like to visit a location l. It is defined below

$$p_{u,l} = \frac{\sum_{v \in U'} c_{v,l} w_{u,v}}{\sum_{v \in U'} w_{u,v}} \tag{3.2}$$

where $w_{u,v}$ is the similarity weight between users u and v and $U' \subset U$ is the top-m users of the target user, i.e., the m users that have the highest similarity weight with the target user.

There are many ways of calculating the similarity weight, including cosine-based similarity, correlation-based similarity, and adjusted cosine similarity [51]. For simplicity, cosine-based similarity is used throughout this paper. The similarity weight between users u and v, denoted as $w_{u,v}$, is defined as follows.

$$w_{u,v} = \frac{\sum_{l \in L} c_{u,l} c_{v,l}}{\sqrt{\sum_{l \in L} c_{u,l}^2} \sqrt{\sum_{l \in L} c_{v,l}^2}} \tag{3.3}$$

Algorithm 1 shows the user-based collaborative filtering algorithm.

Algorithm 1 User-Based Collaborative Filtering Algorithm

```
1: //Input: user u', number of locations to recommend k
2: //Output: list of recommended locations R
3: for all u ∈ U and u ≠ u' do
4:    Calculate w_{u',u}
5: end for
6: Save top-m users with largest similarity weight into U'
7: for all l ∈ L do
8:    Calculate p_{u',l}
9: end for
10: return List of k locations with highest prediction value
```

3.3.2.2 Example Calculation

Table 3.1 shows a toy example of 5 users and 6 locations, with an 'x' for u_i and l_j representing that user i has visited location j or no 'x' representing that user i has not visited location j. Let

Table 3.1: Example table of user check-ins

	l_1	l_2	l_3	l_4	l_5	l_6
u_1	x	x				
u_2			x	x	x	
u_3		x		x		
u_4	x	x			x	x
u_5	x	x		x		

us recommend $k = 3$ new locations for u_1 to visit, with $m = 2$ top users. The first step is to find two other users such that $w_{u_1,u}$ is maximized. Below are the calculations.

$$w_{u_1,u_2} = \frac{0}{\sqrt{2}\sqrt{3}} = 0.000 \qquad w_{u_1,u_4} = \frac{2}{\sqrt{2}\sqrt{4}} = 0.707$$

$$w_{u_1,u_3} = \frac{1}{\sqrt{2}\sqrt{2}} = 0.500 \qquad w_{u_1,u_5} = \frac{2}{\sqrt{2}\sqrt{3}} = 0.816$$

Since w_{u_1,u_5} and w_{u_1,u_4} are the largest, u_4 and u_5 are the top-2 users. Next, we calculate the prediction value for u_1 and each location u_1 has not already visited.

$$p_{u_1,l_3} = \frac{0 * 0.707 + 0 * 0.816}{0.707 + 0.816} = 0.000 \qquad p_{u_1,l_5} = \frac{1 * 0.707 + 0 * 0.816}{0.707 + 0.816} = 0.464$$

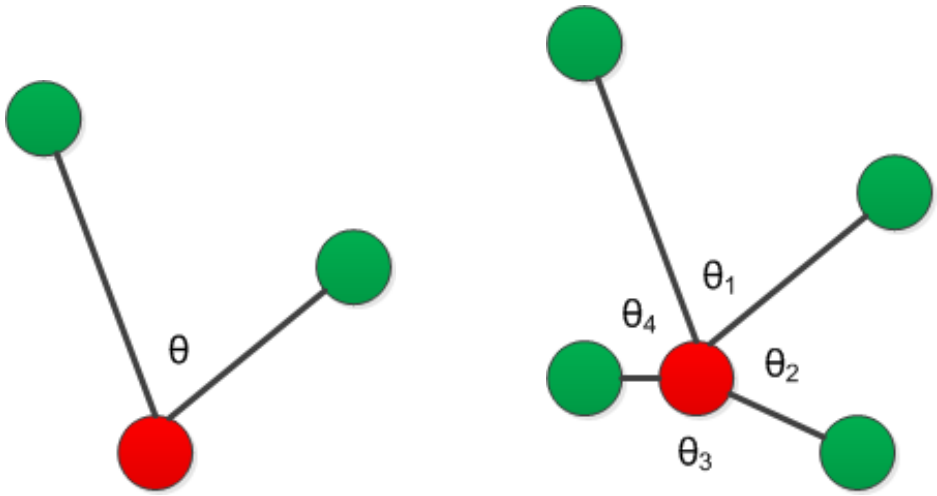
$$p_{u_1,l_4} = \frac{0 * 0.707 + 1 * 0.816}{0.707 + 0.816} = 0.536 \qquad p_{u_1,l_6} = \frac{1 * 0.707 + 0 * 0.816}{0.707 + 0.816} = 0.464$$

Since p_{u_1,l_4}, p_{u_1,l_5} and p_{u_1,l_6} are the largest prediction values, the user-based collaborative filtering would recommend l_4, l_5 and l_6 for u_1.

3.3.3 Angular Diversity Metrics

Given a set of locations L and query point q, angular diversity can be considered as the average pairwise angle between locations. If the angle is larger, the more diverse the locations are. Figure 3.4 shows angular diversity for two locations and three or more locations. Given a query point q and two locations l_1 and l_2, one simple technique for calculating angular diversity is to compute the angle and divide by a normalizing constant. Let $\phi(q, l_1, l_2)$ denote the angle in radians between $\overline{ql_1}$ and $\overline{ql_2}$ (with $\phi(q, l_1, l_2) \leq \pi$), a simple calculation is as follows.

$$\text{Div}_A(q, l_1, l_2) = \frac{\phi(q, l_1, l_2)}{\pi}. \tag{3.4}$$



(a) Two Locations (b) Three or More Locations

Figure 3.4: Angular diversity for two locations and three or more locations

It becomes difficult to extend for three or more locations, but there exists several techniques to calculate the angular diversity of an arbitrary number of locations (greater than 1). The first technique uses the belief that the diversity score depends only on the worst diversity for two locations, which is shown below.

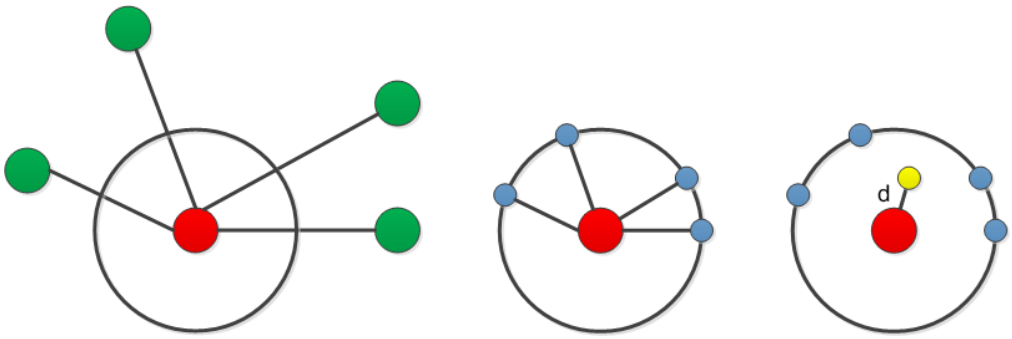
$$\text{Div}_{min}(q, L) = \frac{|L| * \min_{l_1, l_2 \in L | l_1 \neq l_2} \phi(q, l_1, l_2)}{\pi} \tag{3.5}$$

Since the larger the angle is between two locations, the more diverse the locations are, the next technique calculates the angle between every pair of locations and averages them. The normalized equation is shown below.

$$\text{Div}_{pairwise}(q, L) = \frac{2 * \sum_{l_1, l_2 \in L | l_1 \neq l_2} \phi(q, l_1, l_2)}{|L| * (|L| - 1) * \pi} \tag{3.6}$$

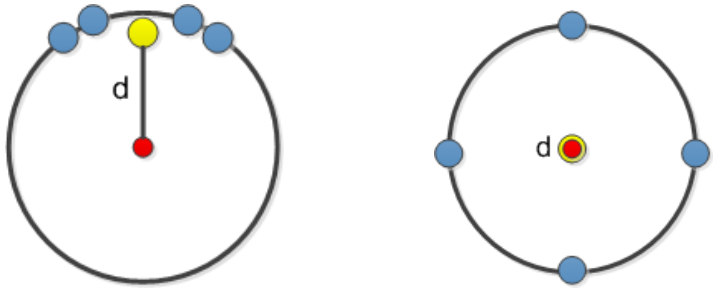
Lastly, a more complicated technique was developed to calculate the angular diversity for two or more locations [34, 35], which is shown in Figure 3.5. Let the green circles be the recommended location L and let the circle be query point q. First, a unit circle is drawn such that q is in the center (Figure 3.5a). Next, each recommended location $l_i \in L$ is normalized to $l'_i \in L'$ on the unit circle such that its distance to the query point is 1 and all pairwise angles are equivalent between L and L' (Figure 3.5b). Lastly, the mean location of L', denoted as l_m is calculated (Figure 3.5c). Thus, the final diversity score is $1 - \text{Dist}(q, l_m)$. The angular diversity equation using the above technique is below.

$$\text{Div}_{mean}(q, L) = 1 - \frac{\| \sum_{l \in L} \frac{\overline{ql}}{\|\overline{ql}\|} \|}{|L|} \tag{3.7}$$



(a) Locations with unit circle (b) Locations normalized on unit circle (c) Mean Location of Normalized Locations

Figure 3.5: Calculation of angular diversity for two or more locations



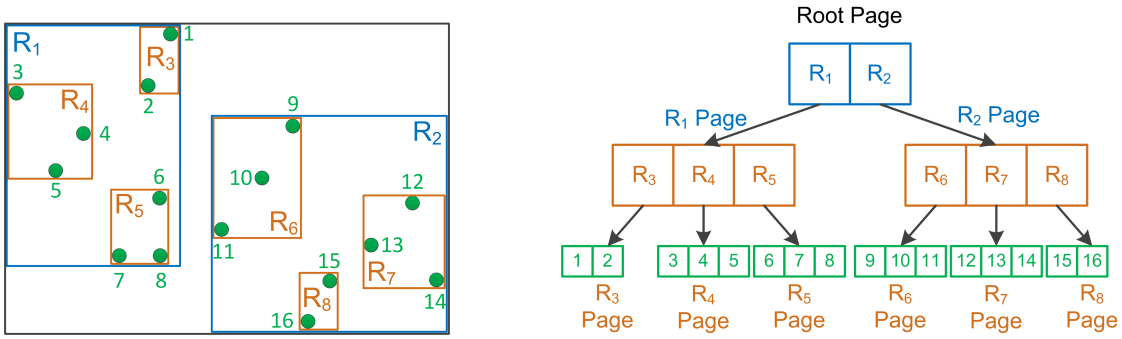
(a) Not Diverse (b) Diverse

Figure 3.6: Spatially non-diverse and diverse normalized locations for mean calculation of angular diversity

To show the difference between non-diverse and diverse for angular diversity, Figure 3.6 shows normalized location with the mean location calculated. In Figure 3.6a, the locations are all above the query point, which means that the corresponding mean location is far away from the query point. However in Figure 3.6b, the locations are in different directions, which means that the corresponding mean location is very close to the query point, with $\text{Dist}(q, l_m) = 0$ in this example.

3.3.4 R-Tree

The R-Tree [23], with variants such as the R+-Tree [54] and R*-Tree [10], is a dynamic index structure that stores spatial objects (e.g. points, rectangles, cubes). It can handle objects of any dimensions, but this thesis focuses on one-dimensional points (locations) in two-dimensional space. The main idea is that the children of a page in the tree are spatially nearby. An R-Tree consists of two types of pages: leaf pages and index pages. A leaf page contains a set of points and a minimum bounding rectangle, which is the smallest rectangle possible that contains all the points stored in the page. Similarly, an index page contains a set of children pages (index or leaf pages) and a minimum bounding rectangle that contains all of the minimum bounding rectangles of its children pages. An example of points and a respective R-Tree are shown in Figure 3.7. We



(a) Points in 2D space with minimum bounding rectangles

(b) R-Tree

Figure 3.7: R-Tree representation for 2D points

have a root node that points to two children, which are the pages of R_1 and R_2. R_1 and R_2 point to its children which are leaf pages, which means that they point to different points stored in the structure.

Each page has a minimum and maximum number of children (or points) that it can contain. During insertion and deletion, the R-Tree remains balanced, which means that the leaf pages are on the same level of the tree. If a page overflows or underflows, the R-Tree will split pages or merge pages to maintain the minimum and maximum number of children. For initially creating an R-Tree with a set of points, bulk loading methods, such as Nearest-X [49], Hilbert Curve [30] and Sort-Tile-Reverse [37], allow the R-Tree to be more efficiently loaded instead of inserting each point individually.

The R-Tree allows for efficient implementations of popular spatial search queries, such as point queries (e.g., find the points with a given coordinate), range queries (e.g., find all points that are within 100 km of given coordinate), and nearest-neighbor queries (e.g., find the point that is nearest to a given coordinate).

3.3.5 K-Nearest Neighbor Query

The goal of a k-nearest neighbor query is to find k points that are closest to a given query point. Mathematically, given a set of points (or locations) P, query point q, and positive integer k, we want to choose a set of point $R \subseteq P$ such that $|R| = k$ and $\forall r \in R, \forall q \in P \setminus R, \text{Dist}(q, r) \leq \text{Dist}(q, p)$. This problem is illustrated in Figure 3.8, where the red circle is the query point, the green circles are points, and the yellow circles are the closest neighbors (for $k = 3$). An example of a k-nearest neighbor query is to find the five closest restaurants to the user's current coordinate. This can be practical when recommending locations that are close to a query point.

With brute force, an algorithm can solve the k-nearest neighbor query for query point q by accessing every point p, calculating $\text{Dist}(q, p)$, and choosing the k points with the smallest distance calculation. The problem is that the algorithm would have to access every point from the

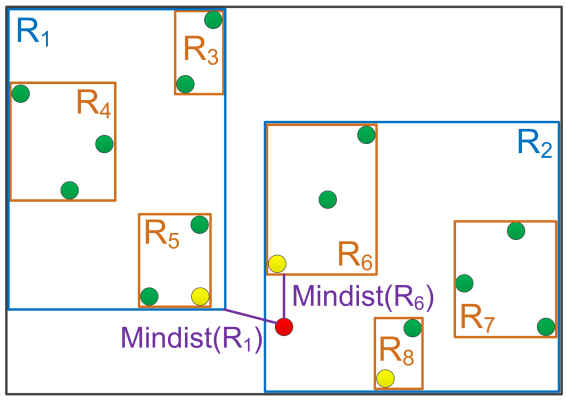


Figure 3.8: K-nearest neighbor representation for 2D points

Algorithm 2 K-Nearest Neighbor Query

```
1: //Input: query point q, Number of nearest neighbors k, R-Tree root page root
2: //Output: List sorted in increasing distance from q of nearest neighbors NN
3: Let NN be an empty list of locations
4: Let PQ be a min priority queue for mindist for R-Tree pages and locations
5: Enqueue PQ with root
6: while PQ is not empty and |R| < k do
7:    Dequeue PQ and let obj be the spatial object that was dequeued
8:    if obj is a location then
9:       Append obj to NN
10:   else
11:      for all children (or locations) c of obj do
12:         Enqueue PQ with c
13:      end for
14:   end if
15: end while
```

database, which would incur a large number of disk accesses if the number of points to choose from is not limited. However, the query can be more efficiently implemented with searching for points in an R-Tree. The key to the efficiency of the algorithm is searching for locations in increasing order of Mindist(P), which is the minimum distance from q to any possible point in the minimum bounding rectangle of P [48]. If there a set of k points such that Dist$(q, p) \leq$ Mindist(P) for p being one of the k points, then the tree rooted at P will not be explored. This enables the algorithm to perform efficiently. Using this, Algorithm 2 provides pseudocode for the k-nearest search query, based off the algorithm for distance browsing [25].

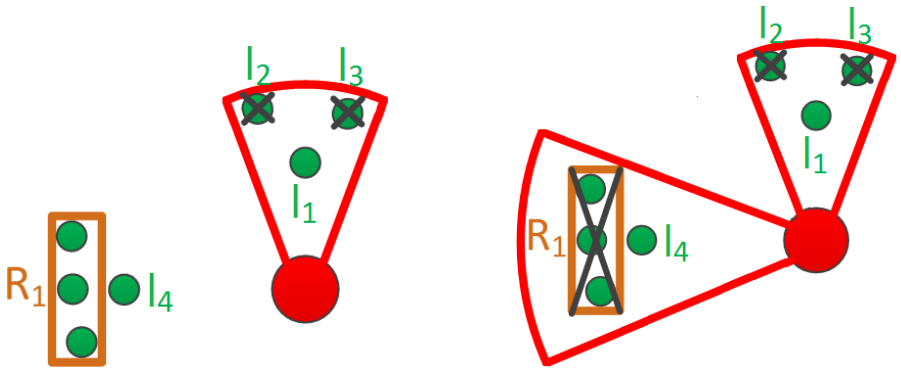
3.3.6 Index-Based Diverse Browsing

A variant of the k-nearest neighbor problem is the k-nearest diverse neighbor problem, which is to choose the k closest points/locations that are "diverse" [24, 28, 34, 35]. Below is the current state-of-the-art algorithm for calculating the k-nearest diverse neighbor problem using angular

spatial diversity.

3.3.6.1 Algorithm

Kucuktunc et al. explore this problem for angular spatial diversity and created Index-Based Diverse Browsing (IBDB), which modifies the k-nearest neighbor approach above to return diverse locations [34, 35]. The algorithm works by choosing locations and adding them to a set of recommended locations. If a location or R-Tree page is considered not diverse, it is pruned.



(a) Pruning a location (b) Pruning an R-Tree page

Figure 3.9: Pruning based off MinDivDist

Similar to *mindist* in the k-nearest neighbor algorithm, the authors defined *mindivdist* that considers the linear combination (with control parameter λ) of *mindist* as well as diversity in comparison to already chosen locations. Instead of choosing the k locations with the smallest *mindist*, the intuition is to choose k locations with the smallest *mindivdist* score. Similar to *mindist*, the access order of *mindivdist* is mainly based on distance, but the difference is that the ranking of locations that are not diverse in *mindivdist* are penalized, in relation to locations already chosen. Let R be the set of locations that have already been chosen and q be the query point. If we want to calculate *mindivdist* for a location *obj*, we compare *obj* with each $p \in R$. If *obj* is within a certain angle of any $p \in R$ and nearby in distance, it is pruned. Intuitively, for each already chosen location, we can draw a sector as shown in 3.9a such that all locations in the sector will be pruned. Otherwise, we finish the calculation for *mindivdist* of *obj*, which is $\max_{p \in R} \lambda * \text{sim}_\theta(q, obj, p) + (1 - \lambda) * \text{Dist}(q, obj)$. The constant θ signifies the minimum angle between two locations (in relation to a query point) that is considered diverse and the sim function signifies how angularly similar two locations are, which is defined as follows.

$$\text{sim}_\theta(q, obj, p) = \begin{cases} 1 - \phi(q, obj, p) & \text{if } \phi(q, obj, p) < \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

If *obj* is an R-Tree page (i.e. minimum bounding rectangle), we want to calculate the minimum linear combination of *mindist* and diversity for a point in the page. Let C be the corners of the

Algorithm 3 MinDivDist

```
1: //Input: query point q, location or R-Tree page obj, set of points R, control parameter λ
2: //Output: score mindivdist
3: θ_s = 2π/(k+ε)
4: r_s = 1 + λ
5: δ = mindist(q, obj)
6: if obj is a location then
7:    max_s = 0
8:    for all p ∈ R do
9:       s = sim_θ(q, obj, p)
10:      if s > 0 and δ < |qp| * r_s then
11:         return Prune
12:      end if
13:      max_s = max(max_s, s)
14:   end for
15:   return λ * max_s + (1 − λ) * δ
16: else
17:   mindivdist = 0
18:   Let C be the set of corners of obj
19:   for all p ∈ R do
20:      s = min_{c∈C} sim_θ(c, q, p)
21:      Let C' ⊆ C be the points that are in sector (q, qp, θ_s, |qp| * r_s)
22:      if |C'| = |C| then
23:         return Prune
24:      else if |C'| ≥ 1 then
25:         δ = min_{c'∈C'}(|qp| * r_s, |qc'|)
26:      end if
27:      mindivdist = max(mindivdist, λ * s + (1 − λ) * δ)
28:   end for
29:   return mindivdist
30: end if
```

minimum bounding rectangle. Similar to when *obj* is a location, the algorithm compares *obj* with each $p \in R$. With p, the algorithm creates a sector $S = (q, \overrightarrow{qp}, \frac{2\pi}{k+\epsilon}, |\overline{qp}| * (1 + \lambda))$ If all of the corners are within S (i.e., all of the corners are not considered diverse in comparison to p), the R-Tree page is pruned (shown in 3.9b). Otherwise, we calculate *mindivdist* of *obj*, which maximizes (for each $p \in R$) the lowest possible combination of diversity and distance. More details for *mindivdist* are shown in the pseudocode for Algorithm 3. One observation is that the *mindivdist* calculation for an R-Tree page and a location depend on the set of already chosen locations. Since the set increases when a new location is separate, the *mindivdist* calculation needs to be redone.

Now that *mindivdist* has been discussed, we show how the IBDB algorithm works, which is similar to k-nearest neighbor. The authors use a minimum priority queue (based off *mindivdist*) to decide which object (either location or R-Tree page) to dequeue. If the object is a location, then it is added to the recommended set. If the object is an R-Tree page, its children pages (or it is a leaf page, its locations) are enqueued as long as the *mindivdist* calculation does not

Algorithm 4 Index-Based Diverse Browsing

```
1: //Input: query point q, number of locations to recommend k, R-Tree root page root, control
   parameter λ
2: //Output: list of recommended locations R
3: Let R be an empty set of points
4: Let PQ be a min priority queue for mindivdist with tuples (spatial object – point or R-Tree
   page, counter, mindivdist)
5: Enqueue PQ with (root, cts, 0)
6: while PQ is not empty and |R| < k do
7:    while the top of PQ has count less than |R| do
8:       Dequeue PQ and let obj be the spatial object that was dequeued
9:       Enqueue PQ with (obj, cts, MinDivDist(q, obj, R, λ)) if Prune is not returned
10:   end while
11:   Dequeue PQ and let obj be the spatial object that was dequeued
12:   if obj is a location then
13:      Add obj to R
14:   else
15:      for all children child of obj do
16:         Enqueue PQ with (obj, |R|, MinDivDist(q, child, R, λ)) if Prune is not returned
17:      end for
18:   end if
19: end while
20: return R
```

prune any pages or locations. To store extra information in the priority queue, each element of the priority queue is a 3-tuple (obj, ctr, mdd), where obj is the spatial object, ctr is the counter when the tuple was inserted into the priority queue, and mdd is the *mindivdist* value of obj. The algorithm will continue this process of enqueuing and dequeuing until enough locations are recommended. The problem with this simplistic algorithm is that when a new location is added to the set of recommended locations, the *mindivdist* values for every spatial object may change. Thus, the min priority queue keeps track of the number of recommended points when each object was inserted into the priority queue. IBDB can dequeue the priority queue and if the counter is not equal to the current number of recommended points, it recalculates *mindivdist* for the object and enqueues it. Since the authors of the algorithm prove that *mindivdist* values get only larger with the addition of the new recommended location, the first dequeued object with the correct counter is guaranteed to have the lowest *mindivdist* in the priority queue [34, 35]. The pseudocode for the IBDB algorithm is shown in Algorithm 4.

3.3.6.2 Example Calculation

To show how IBDB works, we will step through the example illustrated in Figure 3.10. In the illustration, the red circle is the query point and the green circles are candidate locations. The rectangles represent R-Tree pages, where R_1 and R_2 are children of $root$, R_3 and R_4 are children of R_1, and R_5 and R_6 are children of R_2. In addition, the radius of the inner dotted circle center upon the query point is 5 km while the radius of the outer dotted circle is 10 km. For

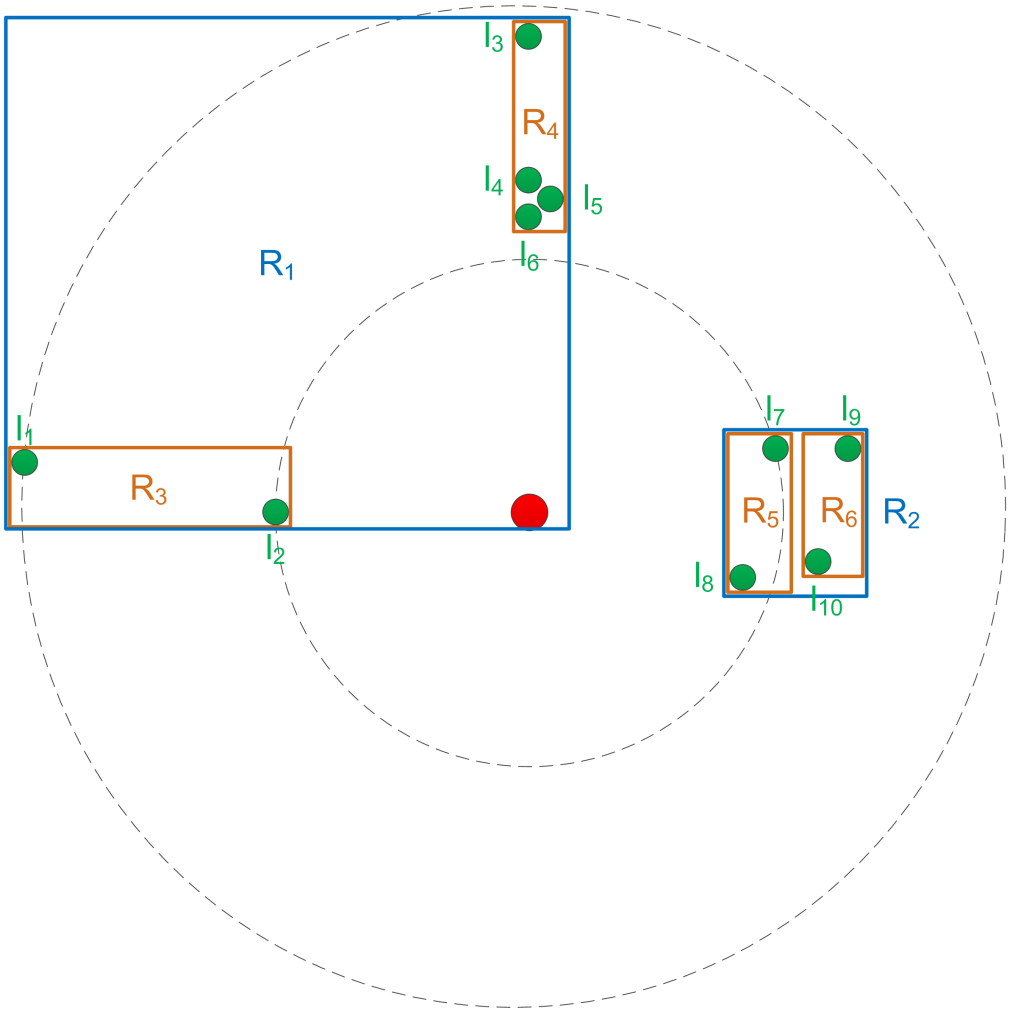
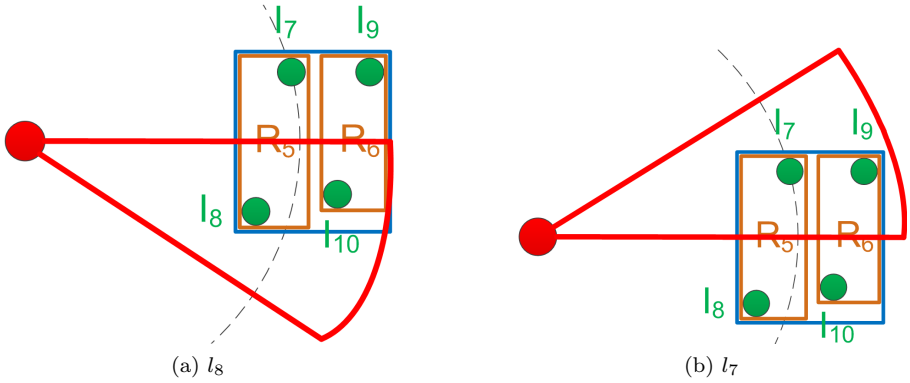


Figure 3.10: Example calculation for IBDB algorithm

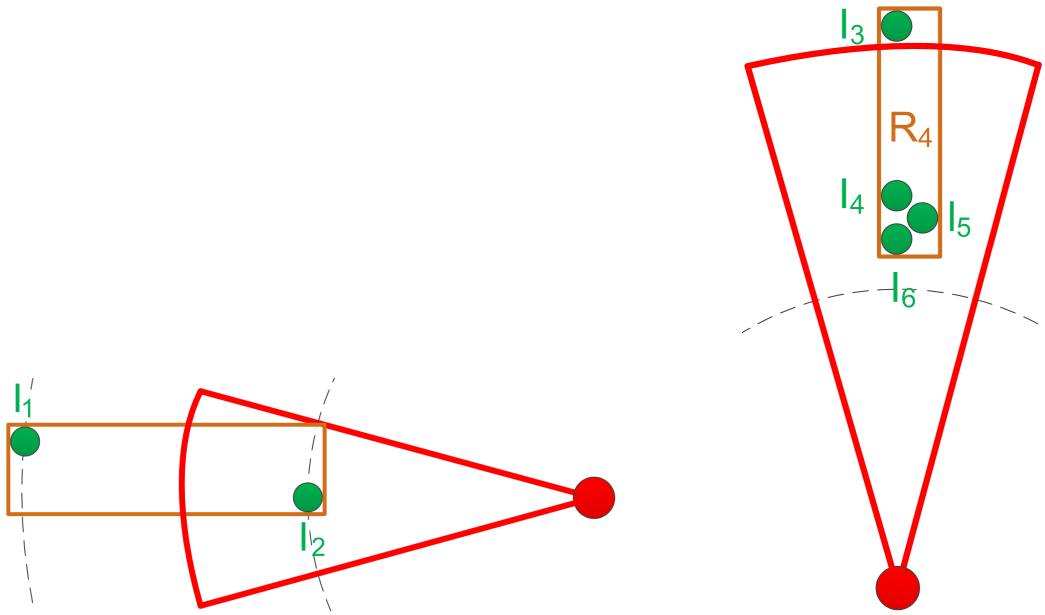
our example, we will recommend five locations. For the constants in the algorithm, let $\lambda = 0.5$, $\theta = \frac{\pi}{10} = 18°$.

Tables 3.2 and 3.3 show step-by-step instructions how the IBDB algorithm executes for the example in Figure 3.10. The first column represents the step number, the second column represents the current priority queue, the third column represents the already chosen locations, and the fourth column gives a description of how the algorithm moves from the current step to the next step. For the second column, each element in the priority queue is represented as a 3-tuple (obj, ctr, mdd), where obj is the spatial object, ctr is the counter when the tuple was inserted into the priority queue, and mdd is the *mindivdist* value of obj. Throughout the descriptions, let MDD = MinDivDist.



(a) l_8 (b) l_7

Figure 3.11: Pruning area for l_8 and l_7



(a) l_2 (b) l_6

Figure 3.12: Pruning area for l_2 and l_6

Table 3.2: Index-Based Diverse Browsing example

Step #	PQ	R	Description
1	$(Root, 0, 0)$	empty	Since *Root* is an R-Tree page, we add its children - R_1 and R_2 - to PQ. No locations are in R, so the similarity calculation will always be 0. Since $\text{MDD}(R_1) = 0.5 * 0 + 0.5 * 0 < \text{MDD}(R_2) = 0.5 * 0 + 0.5 * 4$, R_1 is at the top of PQ.
2	$(R_1, 0, 0)$ $(R_2, 0, 2)$	empty	Since R_1 is an R-Tree page, we add its children - R_3 and R_4 - to PQ. Since $\text{MDD}(R_3) = 0.5 * 0 + 0.5 * 5 < \text{MDD}(R_4) = 0.5 * 0 + 0.5 * 6$, R_3 is before R_4, but after R_2.
3	$(R_2, 0, 2)$ $(R_3, 0, 2.5)$ $(R_4, 0, 3)$	empty	Since R_2 is an R-Tree page, we add its children - R_5 and R_6 - to PQ. With this, we have $\text{MDD}(R_5) = 0.5 * 0 + 0.5 * 4$ and $\text{MDD}(R_6) = 0.5 * 0 + 0.5 * 6$.
4	$(R_5, 0, 2)$ $(R_3, 0, 2.5)$ $(R_6, 0, 3)$ $(R_4, 0, 3)$	empty	Since R_5 is an R-Tree page, we add its locations - l_7 and l_8 - to PQ. With this, we have $\text{MDD}(l_7) = 0.5 * 0 + 0.5 * 5$ and $\text{MDD}(l_8) = 0.5 * 0 + 0.5 * 4$.
5	$(l_8, 0, 2)$ $(l_7, 0, 2.5)$ $(R_3, 0, 2.5)$ $(R_6, 0, 3)$ $(R_4, 0, 3)$	empty	Since l_8 is a location, it is added to R. Since the counter of l_7 in PQ is incorrect, we dequeue and enqueue with the correct counter and $\text{MDD}(l_7) = 0.5 * 0 + 0.5 * 5$. Since the angle between l_7 and l_8 is greater than θ (as shown being outside the sector in Figure 3.11a), the smiilarity calculation for l_7 is 0.
6	$(l_7, 1, 2.5)$ $(R_3, 0, 2.5)$ $(R_6, 0, 3)$ $(R_4, 0, 3)$	l_8	Since l_7 is a location, it is added to R. Since the counter of R_3 in PQ is incorrect, we dequeue and enqueue with the correct counter and $\text{MDD}(R_3) = 0.5 * 0 + 0.5 * 5$ with the similarity calculation being 0 because its angle with l_7 and l_8 is greater than θ.
7	$(R_3, 2, 2.5)$ $(R_6, 0, 3)$ $(R_4, 0, 3)$	l_8, l_7	Since R_3 is an R-Tree page, we add its locations - l_1 and l_2 - to PQ. Since both locations' angle with l_7 and l_8 is greater than θ, we have $\text{MDD}(l_1) = 0.5 * 0 + 0.5 * 5$ and $\text{MDD}(l_2) = 0.5 * 0 + 0.5 * 10$.
8	$(l_2, 2, 2.5)$ $(R_6, 0, 3)$ $(R_4, 0, 3)$ $(l_1, 2, 5)$	l_8, l_7	Since l_2 is a location, it is added to R. Since the counter of R_6 in PQ is incorrect, we dequeue and enqueue with the correct counter and $\text{MDD}(R_6) = 0.5 * 0.9 + 0.5 * 5$ with the similarity score being large because the angle between l_7 and the top left corner of R_6 is small.

Table 3.3: Index-Based Diverse Browsing example (continued)

Step #	PQ	R	Description
9	$(R_6, 3, 2.95)$ $(R_4, 0, 3)$ $(l_1, 2, 5)$	$l_8, l_7,$ l_2	Since R_6 is an R-Tree page, we consider adding its locations - l_9 and l_{10} - to PQ. Since l_{10} is in the sector of l_8 (see Figure 3.11a) and l_9 is in the sector of l_7 (see Figure 3.11b), both locations are pruned. Since the counter of R_4 is incorrect, we dequeue and enqueue with the correct counter and MDD$(R_4) = 0.5 * 0.0 + 0.5 * 6$.
10	$(R_4, 3, 3)$ $(l_1, 2, 5)$	$l_8, l_7,$ l_2	Since R_4 is an R-Tree page, we add its locations - l_3, l_4, l_5, and l_6 - to PQ. Since all of these locations have an angle greater than θ compared to the already chosen locations, all similarity scores are 0. Therefore, we have MDD$(l_3) = 0.5 * 0 + 0.5 * 9$; MDD$(l_4) = 0.5 * 0 + 0.5 * 7$; MDD$(l_5) = 0.5 * 0 + 0.5 * 6.5$; and MDD$(l_6) = 0.5 * 0 + 0.5 * 6$.
11	$(l_6, 3, 3)$ $(l_5, 3, 3.25)$ $(l_4, 3, 3.5)$ $(l_3, 3, 4.5)$ $(l_1, 2, 5)$	$l_8, l_7,$ l_2	Since l_6 is a location, it is added to R. Since the counter of l_5 in PQ is incorrect, we dequeue and recalculate MinDivDist. Since l_5 is in the sector of l_6 (see Figure 3.12a), l_5 is pruned. The same process occurs with l_4, where it is pruned because of l_6 as well. Next is l_3, which we dequeue and enqueue with the correct counter and MDD$(l_3) = 0.5 * 1.0 + 0.5 * 9$ with the similarity score being one because the angle of l_6 and l_3 are equal.
12	$(l_3, 4, 5)$ $(l_1, 2, 5)$	$l_8, l_7,$ l_2, l_6	Since l_3 is a location, it is added to R. Since five locations have been chosen, R is returned.
13	$(l_1, 2, 5)$	$l_8, l_7,$ $l_2, l_6,$ l_3	End of algorithm.

Chapter 4

Data Analysis

After introducing the datasets used in this paper, we look into experiments using user-based collaborative filtering upon real datasets to discuss issues arising when making recommendations for mobile users traveling out of town. Then, we perform experiments to explore the mobility of users to see the distance users travel to visit locations as well as factors that are important in recommendation.

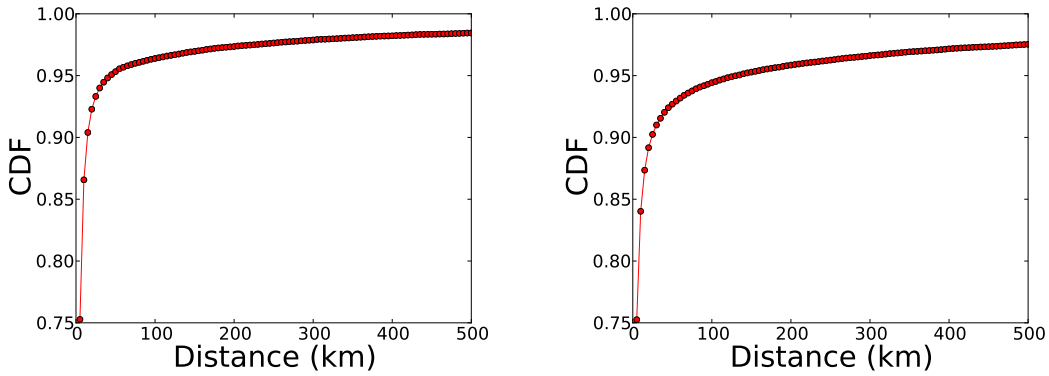
4.1 Datasets

The Foursquare dataset includes $202,932$ users and $155,321$ locations while the Gowalla dataset includes $116,889$ users and $1,070,338$ locations. With the check-in data, the user check-in matrix density for Foursquare is 2.42×10^{-5} while the user check-in matrix density for Gowalla is 4.09×10^{-5}. This means that each user has an average of 3.76 and 43.81 check-ins for Foursquare and Gowalla, respectively. In addition, there is a total of $1,713,965$ social connections (or called *friend*) pairs in the Foursquare dataset (note that friendship is mutual, so user u being a friend with user v and v being a friend with u count as 1 pair) and $267,505$ friendship pairs in the Gowalla dataset. Therefore, the average user has 8.45 and 2.29 friends and the user-user friendship matrix density is 8.32×10^{-5} and 1.95×10^{-5} for the Foursquare and Gowalla datasets, respectively. Lastly, the Foursquare dataset contains the home location of each user, which is self-reported by each user.

4.2 Location Recommendation for Out-of-town Users

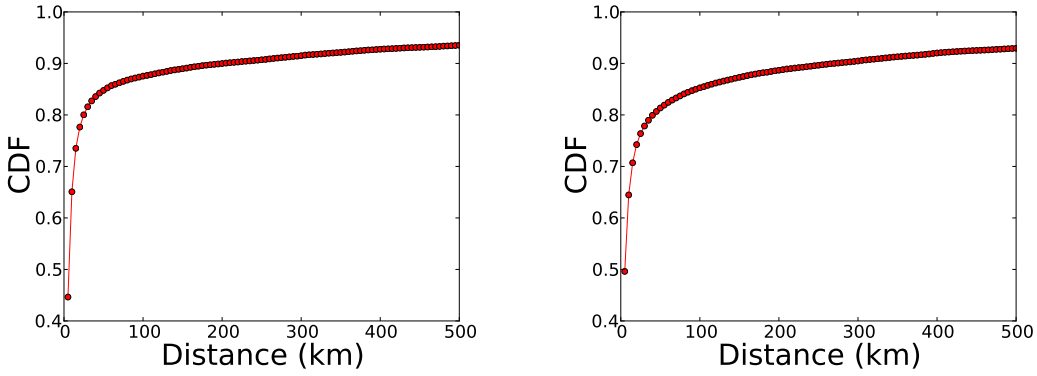
We aim to support location recommendation for both in-town and out-of-town scenarios. Thus, we first conduct experiments using user-based collaborative filtering upon two real datasets collected from Foursquare and Gowalla. Our goal is to observe how effective user-based collaborative filtering performs when making recommendations to users located at different distances from their

home regions (i.e., the region where the majority of their check-in activities occur). Particularly, we are interested in studying the phenomenon when users visit locations far away from their home regions.



(a) Foursquare (b) Gowalla

Figure 4.1: Probability distribution of users that have 50% of recommended locations within distance of a visited location

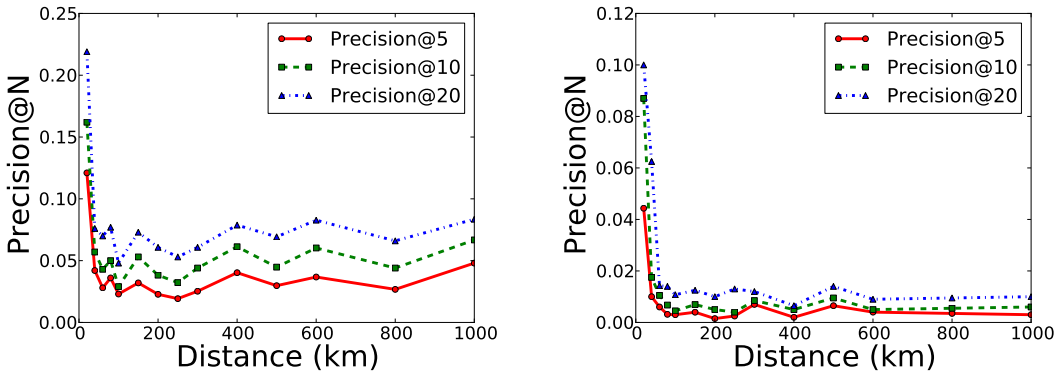


(a) Foursquare (b) Gowalla

Figure 4.2: Probability distribution of users that have 80% of recommended locations within distance of visited location

First, we evaluate how far the user-based collaborative filtering algorithm recommends from previously visited locations for target users. For each user, we recommend 20 locations and calculate the distance between each recommended location and the closest visited location. The cumulative distribution function shows the probability distribution of users that have 50% or 80% of visited locations within the distance on the x-axis in Fig. 4.1 and 4.2, respectively. As shown, many users' recommended locations are near previously visited locations, with 95.3% and

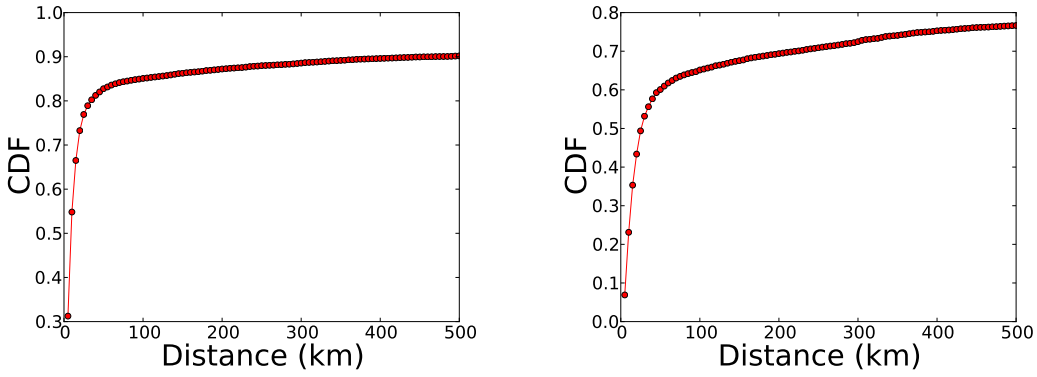
92.7% of users in Foursquare and Gowalla having 50% or more of their locations within 50 km and 84.8% and 81.3% of users in Foursquare and Gowalla having 80% or more of their locations within 50 km. Thus, if a user travels away from her visited locations, the user-based collaborative filtering algorithm may recommend locations far away from the current coordinate of the user, which provides an inferior recommendation.



(a) Foursquare (b) Gowalla

Figure 4.3: Precision of user-based collaborative filtering

To confirm our beliefs, we evaluate the user-based collaborative filtering algorithm using the check-ins as ground truth. By marking off certain randomly selected locations within distance ranges under examination as "not-visited", we measure how well the recommendations recover these marked-off locations (the detailed evaluation process will be elaborated later in Section 7.1.1). By varying the distance from the home region, Fig. 4.3 plots the *precision@N* for different distance ranges (e.g., 0-20 km, 20-40 km, ..., up to 1000 km). As shown, when the marked-off locations are close to home regions of users, the user-based collaborative filtering method performs reasonably well. However, as soon as the marked-off locations are 20-40 km away from the home regions of users, the precision degrades. This phenomenon appears in both the Foursquare and Gowalla datasets consistently, acknowledging our concerns that collaborative filtering may not work well when a user is out of town. We argue that this degradation is due to: (a) the recommended locations derived from candidate locations previously visited by the top similar users of the target user are likely to be close to her home region and thus too far away from her current region; and (b) some of the most similar users may not have visited locations near the target user's current region. To address (a), an idea is to incorporate a *proximity constraint* to filter locations far away from the user's current region. To address (b), we need to extend the base of similar users from which the candidate locations are derived. As prior studies have shown that friends tend to exhibit similar behaviors (and we assume that includes far-away cities or scenic places traveled), we provide data analysis into the mobility of users and similar users/friendship in the following sections and then propose to incorporate friends into our



(a) 50% of Locations

(b) 80% of Locations

Figure 4.4: Probability distribution of traveling distance of users from their home location in the Foursquare dataset

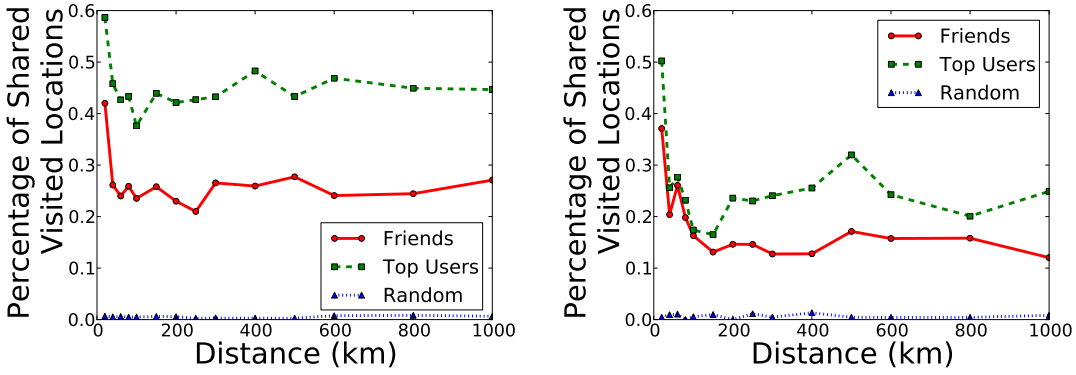
collaborative recommendation framework.

4.3 Mobility of Users

For the mobility of users, we explore how far users tend to travel to visit locations. For this experiment, we use the Foursquare dataset since it contains home locations that users report on their user profile. Analyzing users that have visited at least 10 locations, we examine how close 50% and 80% of the user's visited locations are to their home location to show how far users typically travel. The probability distribution to display how far users travel from their home location to visit locations is shown in Fig. 4.4, where a data point on the graph represents the percentage of users that have at least a certain percentage of locations (50% or 80%) within the given distance on the x-axis. The figure shows that many users visit locations nearby their home location, with 83.7% of users visiting 50% of their locations and 63.3% of users visiting 80% of their locations within 100 km of their home location. Therefore, we see that many users visit a large majority of their locations near their home location. This confirms our intuition that users tend to visit nearby places. Thus, location recommender systems should recommend places to users that are a short distance away from the current standing point of the user.

4.4 Similar Users and Friends

When users are far away from their other visited locations, we would like to know how this distance affects how similar users and friends influence the decision to visit locations. We examine how many friends have visited the far away location. Requiring that each user has at least 5 friends and visited 10 locations, we find all the user location pairs (u, l) (where user u has visited



(a) Foursquare

(b) Gowalla

Figure 4.5: Percentage of friends and top users that share visited location of target user

location l) such that l is at least 1,000 km away from the other visited locations of u. We observe that 28.9% and 12.7% of the users also have a friend visit the far away location for the Foursquare and the Gowalla dataset respectively. In addition, for each location that a person visits that is far away from her home region, how often does at least one of the top-m users (from the user-based collaborative filtering algorithm) also visit the location? Requiring that each user visited at least 10 locations and the distance threshold being 1000 km, 74.8% and 54.6% of top-50 users as well as 47.8% and 29.3% of top-10 users have also visited the far away locations for the Foursquare dataset and Gowalla dataset, respectively. When we lower the distance threshold to 500 km, the percentage of top-10 users drops to 34.8% in the Foursquare dataset and 25.3% in the Gowalla dataset. This shows that when users are far away from their home region, top-10 users and friends have a high probability to share these visited locations. If we apply collaborative filtering algorithm to complete location recommendation, the top similar users and friends are good references for the targeting users.

To further examine this, we want to observe how important friends and top-10 users are as a user is farther away from her home region. To do this, we found user location pairs (user u and location l) for different ranges (e.g., 0-20 km, 20-40 km) from her home regions such that each user had at least 10 friends and 10 visited locations. For each user location pair (u, l), we identify whether a friend or a top-10 user of u has visited l. In addition, we choose random users (with quantity equal to the number of friends of the user) and identify whether these random users have visited l. Lastly, the results are aggregated for the different distances from home ranges by calculating the percentage of time a friend or a top-10 user of u has visited l. Figure 4.5 shows how often friends, top-10 users, and random users share at least one visited location with a target user.

Both friends and top users of u had a larger percentage (42.0% and 58.7% respectively for the Foursquare dataset and 37.1% and 50.2% respectively for the Gowalla dataset) when the

distance equaled 0-20 km, and then the percentage immediately dropped. After the initial high percentages, both friends and top users' percentages stay mostly level, hovering at around 25% and 45% for the Foursquare dataset and 15% and 25% for the Gowalla dataset. This shows that both friends and top-10 users of a target user behave similarly to the target user, with top-10 users having more similarity. In addition, we observe that both friends and top users always perform significantly better than just randomly choosing users. Thus, no matter how far the target user is from her home region, both a friend and top-10 user of a target user are more likely to visit a location of the target user than a random user.

Chapter 5

Location Recommendation Algorithm

In this chapter, we introduce the User Preference, Proximity and Social-Based Collaborative Filtering framework, which adds social relationships and a proximity constraint to provide a superior algorithm. Then, we show an example calculation.

5.1 User Preference, Proximity and Social-Based Collaborative Filtering Framework

Based on the previous observations, we propose the User Preference, Proximity and Social-Based Collaborative Filtering (UPS-CF) recommendation framework.

The basic idea of UPS-CF follows the user-based collaborative filtering algorithm to explore the implicit preferences of top similar users in making location recommendation. However, due to the constraint of human mobility, locations that are far away from the current location of the target user should be excluded from consideration. Thus, UPS-CF adopts a proximity constraint, denoted as d_p, to filter candidate locations which are located outside the circle of radius d_p, rooted at the current position of the target user. Notice that d_p can be adapted based on application requirements. For example, for a mobile map application on an LBSN, the location recommendation should be tailored based on the displayed region on the map. Thus, the recommendation engine may set the proximity constraint accordingly to filter candidate locations of no interests.

Additionally, most similar users, who share many commonly visited locations with the target user, may not be the best source to decide candidate locations because those similar users may not have visited the current region of the target user. Thus, UPS-CF incorporates friends (i.e., socially connected users) in order to broaden the selection base. The idea of adopting friends in

recommender systems is reasonable as the homophily and social influence phenomenon among friends indicate that friends tend to have similar behaviors. Therefore, we aim to incorporate this factor and investigate its roles (in comparison with similar users) in UPS-CF under in-town and out-of-town scenarios.

Algorithm 5 User Preference, Proximity and Social-Based Collaborative Filtering Algorithm

```
1: //Input: user u', query point q, number of locations to recommend k
2: //Output: list of recommended locations R
3: for all users u ≠ u' do
4:    Calculate w_{u,u'} = (1 − α) * w^F_{u,u'} + α * w^E_{u,u'}
5: end for
6: Save top-m users with largest similarity weight into U'
7: Let L' be all locations that satisfy the proximity constraint d_p from q
8: for all locations l ∈ L' do
9:    Calculate prediction value p_{u',l}
10: end for
11: return List of N locations with highest prediction value
```

The processing flow of UPS-CF is described in Algorithm 5, where $U' \subset U$ is the set of top-m users, $c_{u,l} = 1$ represents user u has a check-in at location l, and $c_{u,l} = 0$ represents user u has no record of a check-in at location l. As shown, UPS-CF incorporates the notion of a proximity constraint by filtering out candidate locations that are farther away from the current standing location of the target user. Similar to the user-based collaborative filtering algorithm, UPS-CF defines a ranking score as the probability of a user u visiting location l, denoted $p_{u,l}$

$$p_{u,l} = \frac{\sum_{v \in U'} c_{v,l} w_{u,v}}{\sum_{v \in U'} w_{u,v}} \tag{5.1}$$

where $w_{u,v}$ indicates the importance (i.e., weight) of a user v contributing to a recommendation targeting on user u.

Notice that the similarity weight $w_{u,v}$ used in Eq. (5.1) is different from that in Eq. (3.3). The weight $w_{u,v}$ here is used to combine the roles of a similar user and a friend the user v may play for the target user u. Thus, we use a control parameter α (where $0 \leq \alpha \leq 1$) to balance the weight $w^E_{u,v}$ for the role of a similar user and the weight $w^F_{u,v}$ for the role of a friend and define $w_{u,v}$ as follows.

$$w_{u,v} = (1 - \alpha) * w^F_{u,v} + \alpha * w^E_{u,v}$$

Accordingly, UPS-CF is able to take advantage of similar behavior between close social friends in a social network as well as user preference between similar users. Notice that user preference among similar users is derived based on the widely-known belief that similar users tend to visit similar places. This is the main belief that the user-based collaborative filtering employs to decide its top-m users [5, 8, 31, 51]. Let L be the location set and let $c_{u,l}$ be a boolean value that represents whether user u has a record of a check-in with location l. The similarity weight between users u and v, in terms of their common experiences in check-ins, is defined as follows.

$$w^E_{u,v} = \frac{\sum_{l \in L} c_{u,l} c_{v,l}}{\sqrt{\sum_{l \in L} c^2_{u,l}} \sqrt{\sum_{l \in L} c^2_{v,l}}} \tag{5.2}$$

On the other hand, we believe that users are more likely to go to places that friends have previously visited [69]. This is based on the tendency for people to be similar to their friends, due to homophily and social influences among friends, which has been shown to exist in social networks [42]. The friendship between users u and v is denoted as $f_{u,v}$, where $f_{u,v} = 1$ represents that u is friends with v and $f_{u,v} = 0$ represents no record that u is friends with v. Therefore, the similarity weight of social influence between users u and v is defined as follows.

$$w^F_{u,v} = f_{u,v} \tag{5.3}$$

5.2 Example Calculation

To show how the algorithm works, we execute an example problem based off the example in 3.3.2.

Table 5.1: Example table of user check-ins

	l_1	l_2	l_3	l_4	l_5	l_6
u_1	x	x				
u_2			x	x	x	
u_3		x			x	
u_4	x	x			x	x
u_5	x	x		x		

Table 5.2: Example table of social relationships

	u_1	u_2	u_3	u_4	u_5
u_1		x	x		x
u_2	x		x		x
u_3	x	x		x	
u_4			x		x
u_5	x	x		x	

Table 5.3: Example table of location coordinates

	Latitude	Longitude
l_1	0	0
l_2	1	1
l_3	1	-1
l_4	1	1
l_5	-2	1
l_6	2	2

For our example, Table 5.1 shows which users have visited which locations, Table 5.2 shows which users are friends with other users, and Table 5.3 shows the coordinates of the locations. Let us recommend $k = 2$ new locations for u_1 to visit, with $m = 2$ top users. In addition, we set the user's current coordinate (query point) to be $q = (0, 0)$ as well as $\alpha = 0.7$ and $d_p = 3.5$. Lastly, Manhattan distance is used to measure the distance between a location and a query point.

As with user-based collaborative filtering, the first step is to calculate the top-2 users, which is shown below.

$$w_{u_1,u_2} = 0.3 * 1 + 0.7 * \frac{0}{\sqrt{2}\sqrt{3}} = 0.300 \qquad w_{u_1,u_4} = 0.3 * 0 + 0.7 * \frac{2}{\sqrt{2}\sqrt{4}} = 0.495$$

$$w_{u_1,u_3} = 0.3 * 1 + 0.7 * \frac{1}{\sqrt{2}\sqrt{2}} = 0.650 \qquad w_{u_1,u_5} = 0.3 * 1 + 0.7 * \frac{2}{\sqrt{2}\sqrt{3}} = 0.872$$

Thus, u_3 and u_5 are the top-2 users. Next, we calculate the distance from the query point to each location that u_1 has not visited yet to see if any location needs to be filtered.

$$\text{Dist}(q, l_3) = \text{Dist}((0, 0), (1, -1)) = 2 < 3.5 \qquad \text{Dist}(q, l_5) = \text{Dist}((0, 0), (-2, 1)) = 3 < 3.5$$

$$\text{Dist}(q, l_4) = \text{Dist}((0, 0), (1, 1)) = 2 < 3.5 \qquad \text{Dist}(q, l_6) = \text{Dist}((0, 0), (2, 2)) = 4 \not< 3.5$$

Since l_6 is outside the proximity constraint, it is removed from consideration. Lastly, we calculate the prediction value for each location using the same technique as user-based collaborative filtering to determine which locations are recommended.

$$p_{u_1,l_3} = \frac{0 * 0.650 + 0 * 0.872}{0.650 + 0.872} = 0.000 \qquad p_{u_1,l_5} = \frac{1 * 0.650 + 0 * 0.872}{0.650 + 0.872} = 0.427$$

$$p_{u_1,l_4} = \frac{0 * 0.650 + 1 * 0.872}{0.650 + 0.872} = 0.573$$

Therefore, l_4 and l_5 are recommended.

Chapter 6

K-Nearest Diverse Neighbor Algorithms

In this chapter, we introduce several algorithms that solve the k-nearest diverse neighbor problem using angular spatial diversity. First we explain deficiencies in the Index-Based Diverse Browsing algorithm and propose a modified version. Next, we introduce the Distance-Based Diverse Browsing framework, which uses an initial solution and replacement operations to improve the solution. With this framework, we explore two approaches for choosing the initial selection of locations: Distance-First Distance-Based Diverse Browsing and Diversity-First Distance-Based Diverse Browsing

6.1 Modified Index-Based Diverse Browsing

As explained in Section 3.3.6, IBDB is a state-of-the-art algorithm for solving the k-nearest diverse neighbor problem with angular spatial diversity. However, there are deficiencies that either lower performance, in terms of lower quality results, or increase the number of disk accesses. In the following subsections, we introduce Modified Index-Based Diverse Browsing (Mod-IBDB) that makes two improvements: 1) we remove the pruning that occurs in the *mindivdist* calculation and 2) fix the *mindivdist* calculation for R-Tree page to give a more accurate bound for *mindivdist*.

6.1.1 Remove Pruning

The intuition behind the algorithm is to choose k locations that have the smallest *mindivdist* score. However, by pruning locations/R-Tree pages, it may remove locations that would have otherwise been chosen. For example, let us look at choosing two locations in Figure 6.1, where the two locations to the left of the query point (l_1 and l_2) are at the same angle and nearby (i.e., 1 and 1.01 km, respectively) while the location on the far right (l_3) is farther away (i.e., 20 km).

Figure 6.1: Example where pruning may is not optimal

The algorithm will first choose l_1 since it is the closest location. Even though l_2 most likely now has the lowest *mindivdist* score, it is pruned because it is at the same angle as l_1 and about the same distance away. Thus, the chosen locations are l_1 and l_3. Depending on the diversity and distance metrics, this may not be the optimal answer, especially if distance is weighted more importantly than diversity. If we removed pruning from the algorithm, the decision between l_2 and l_3 would depend on which location has the lowest *mindivdist* score, which follows the intuition of the algorithm. Therefore, pruning may lead to less optimal answers.

One argument for pruning is that it will decrease the number of disk accesses. When an R-Tree page is pruned, the algorithm will no longer explore the subtree rooted with the removed page, which saves disk accesses. If the page does not have a low *mindivdist* score, the algorithm will typically not explore the page since other spatial objects will be considered first, which does not increase the number of disk accesses. If the *mindivdist* score is low, it is worthwhile exploiting the tradeoff of adding disk accesses to find locations that may be better choices to recommend. Also, pruning locations does not save disk accesses since the locations have already been read from disk. In actuality, pruning locations can increase disk accesses because the algorithm will have to complete more searching if it prunes locations it would have otherwise chose. This can involve investigating more R-Tree pages for candidate locations, which will increase the number of disk accesses.

6.1.2 Fix Incorrect Mindivdist Calculation

Similar to *mindist* being the minimum distance between any point in the R-Tree page and the query point, the intuition for *mindivdist* of an R-Tree page is that it should be equal to the minimum *mindivdist* value of any point in the R-Tree page. Since the algorithm accesses locations from smallest to largest *mindivdist*, it may access the locations out of order if the above is not true. For example, if the *mindivdist* score for an R-Tree page is 2.0 and the *mindivdist* score for a location is 1.5, the algorithm will access all spatial objects (including locations) that have a *mindivdist* score lower than 2.0 before visiting the location with score 1.5. If this occurs, the algorithm will not correctly choose the k locations with the lowest *mindivdist* score. Unfortunately, this occurs in IBDB.

Let us explore calculating *mindivdist* for the R-Tree page in Figure 6.2, where the center red circle is the query point q, l_1 is the already chosen location (1 km away and angle $\frac{3\pi}{4}$ from q), and l_2, l_3, and l_4 in the rectangle are locations in the minimum bounding rectangle of the R-Tree page. In addition, the dotted circle represents 1 km away from the query point. The sector represents

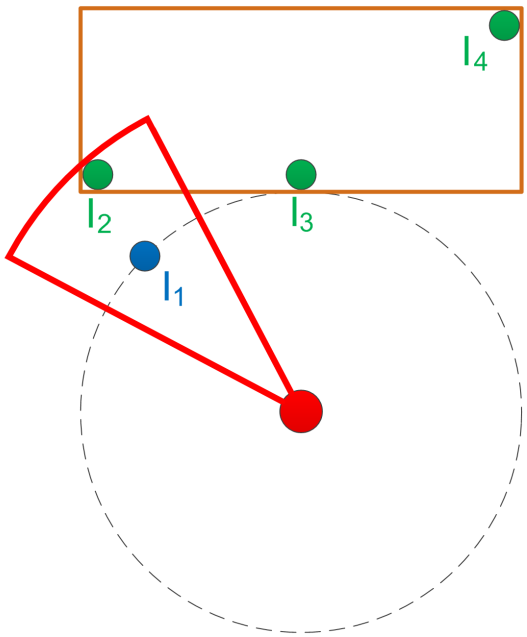


Figure 6.2: Example where *mindivdist* is incorrectly calculated for R-Tree Page in IBDB

the pruning area of l_1, where the width of the sector is $2 * \theta$. Lastly, let $\lambda = 0.5$. For calculating *mindivdist* using the IBDB method, the similarity calculation is $s = \min_{c \in C} \text{sim}_\theta(c, q, p)$, where C is the corners of the minimum bounding rectangle. Since the right corners are outside the θ threshold, $s = 0$. For the distance calculation, since the bottom left corner of the rectangle c is located in the sector, the calculation for the distance portion of *mindivdist* (i.e., δ) is as follows.

$$\delta = \min(|\overline{ql_1}| * (1 + \lambda), |\overline{qc}|) = \min(1 * 1.5, \sqrt{2}) = \sqrt{2}$$

Thus, the calculation of *mindivdist* for the R-Tree page is $0.5 * 0 + 0.5 * \sqrt{2} = \frac{\sqrt{2}}{2}$. However, let us calculate *mindivdist* for l_3. Since it is outside the θ threshold of l_1, $s = 0$. In addition, its distance to the query point is $\delta = 1.0$. Thus, l_2's *mindivdist* score is $0.5 * 0 + 0.5 * 1.0 = 0.5 < \frac{\sqrt{2}}{2}$. Thus, IBDB has the problem of having a location in an R-Tree page with a lower *mindivdist* value.

The problem occurs because it assumes that the lowest *mindivdist* score for a location in the R-Tree page is at the corner. However, the example above shows that this is not necessarily true. To fix this problem, we have to calculate the lowest possible *mindivdist*. Let *rect* be the minimum bounding rectangle of the R-Tree page. Our goal is to find the smallest possible similarity calculation value for any point (does not have to be a specific location) in the rectangle, which is $s = \min_{p' \in rect} \max_{p \in R} \text{sim}_\theta(p', obj, p)$, where R is the set of already chosen locations. In addition, the minimum distance calculation is $\delta = \text{mindist}(q, rect)$. Therefore, the *mindivdist* calculation for an R-Tree page is as follows.

$$\text{MinDivDist}(q, rect, R, \lambda)) = \lambda * \min_{p' \in rect} \max_{p \in R} \text{sim}_\theta(p', obj, p) + (1 - \lambda) * \text{mindist}(q, rect) \quad (6.1)$$

By giving the smallest distance and the smallest diversity calculation, this calculation of *mindivdist* is definitely a lower bound for any point inside the rectangle. However, there is no guarantee that this is a tight lower bound.

6.1.3 Algorithm

Algorithm 6 MinDivDist-Mod

```
1: //Input: query point q, location or R-Tree page obj, set of points R, control parameter λ
2: //Output: score mindivdist
3: δ = mindist(q, obj)
4: if obj is a location then
5:    s = max_{p∈R} sim_θ(q, obj, p)
6: else
7:    Let rect be the minimum bounding rectangle of obj
8:    s = min_{p'∈rect} max_{p∈R} sim_θ(p', obj, p)
9: end if
10: return λ * s + (1 − λ) * δ
```

Putting together the new calculation of *mindivdist* as well as removing pruning, Algorithm 6 shows the new *mindivdist* calculation algorithm for Mod-IBDB method. Other than the above modifications, Mod-IBDB uses the same algorithm as IBDB to choose k locations.

6.1.4 Example Calculation

To show the differences in calculating *mindivdist*, Figure 6.3 illustrates two examples. Similar to the previous subsection, we have the center red circle as the query point q; l_1 as the already chosen location; l_2, l_3, and l_4 as candidate locations; the dotted circle representing 1 km away from the query point; $\lambda = 0.5$; and the sector as the pruning area of l_1, with the width of the sector being $2 * \theta$. For the left example of calculating the *mindivdist* value for l_2, the original method would prune l_2 since it is inside the pruning sector of l_1. For the modified version, this will not occur since pruning is removed. Since l_1 and l_2 have the same angle in relation to q, $s = \text{sim}_\theta(q, l_1, l_2) = 1$. Since l_2 intersects with the dotted circle, $\delta = 1$ km. Therefore, the modified *mindivdist* value is $0.5 * 1 + 0.5 * 1 = 1$.

As calculated in the previous subsection, the *mindivdist* of the R-Tree page for the right example is $\frac{\sqrt{2}}{2}$ even though l_3's *mindivdist* value is smaller (0.5). For the modified version, since there exists points of the rectangle outside the angles of the sector, $s = 0$. In addition, the *mindist* value for the R-Tree page is $\delta = 1$ km since its closest point intersects with the dotted circle. Therefore, the modified *mindivdist* value is $0.5 * 0 + 0.5 * 1.0 = 0.5$. Unlike the

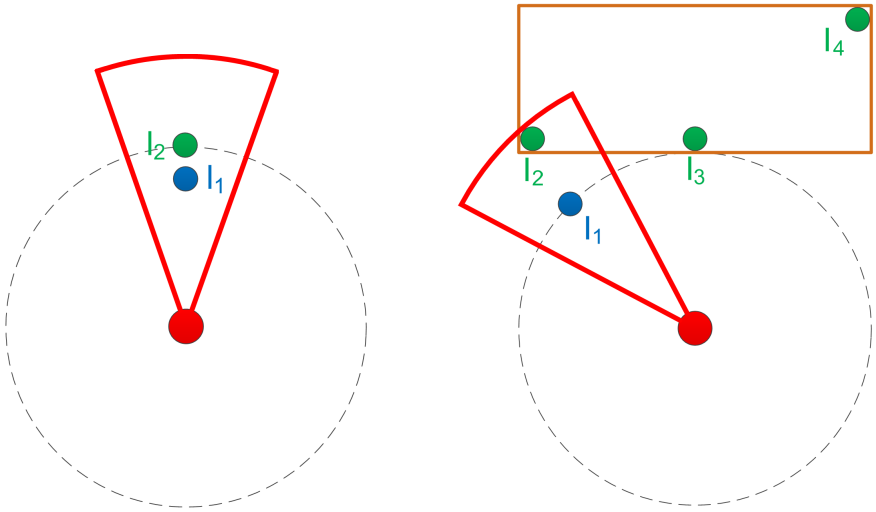


Figure 6.3: Examples for modified *mindivdist* calculation

original calculation, the modified *mindivdist* value for the R-Tree page is not larger than l_3, which correctly follows the intuition of the Index-Based Diverse Browsing algorithm.

6.2 Distance-Based Diverse Browsing

We propose the Distance-Based Diverse Browsing (DBDB) framework to choose k locations that are close to the query point and spatially diverse. The algorithm works by starting with an initial solution of k locations and improving the solution by replacing a candidate location with a location already chosen. The intuition of the framework is based upon search algorithms, which find an initial solution in a hierarchal data structure (i.e., tree) and use backtracking and pruning to improve the solution until a stopping condition has been met or there is no more items to examine. Algorithm 7 shows the framework in detail. After choosing an initial solution with k locations, the algorithm visits locations in increasing order of distance to the query point, in a similar fashion to the distance browsing method for the k-nearest neighbor query. When location l is considered by the algorithm, it is considered to be added to the current solution R if $l \notin R$. If this is true, the algorithm adds l to R and considers which location $r \in R$ should be removed from the solution evaluating the objective function $\lambda * \text{Div}(q, R \setminus r) + (1 - \lambda) * \text{Rel}(q, R \setminus r)$ for each $r \in R$. For the objective function, Div and Rel are diversity and relevance (closeness of locations to query point) metrics, which are detailed in Section 7.2.2, and λ is control parameter for determining the importance of diversity and relevance, where a larger λ means more weight on diversity. The algorithm continues until the stopping condition has been met. For our algorithm, the stopping condition is based upon the consecutive times that a location l is considered but not kept in the solution (i.e., no replacement occurred). In other words, for stopping condition number *stoppingCondition*, if the algorithm attempts replacing *stoppingCondition* consecutive

locations unsuccessfully, the algorithm terminates and returns the current solution.

Algorithm 7 Distance-Based Diverse Browsing

```
1: //Input: query point q, number of locations to recommend k, R-Tree root page root, control
   parameter λ
2: //Output: list of recommended locations R
3: Let R be an initial solution with k locations
4: Let PQ be a min priority queue for mindist for R-Tree pages and locations
5: Enqueue PQ with root
6: counter = 0
7: while PQ is not empty and counter < stoppingCondition do
8:    Dequeue PQ and let obj be the spatial object that was dequeued
9:    if obj is a location then
10:      if obj ∉ R then
11:         R = R ⋃ obj
12:         Find r such that max_{r∈R} λ * Div(q, R \ r) + (1 − λ) * Rel(q, R \ r)
13:         R = R \ r
14:         if r equals obj then
15:            counter = counter + 1
16:         else
17:            counter = 0
18:         end if
19:      end if
20:   else
21:      for all children (or locations) c of obj do
22:         Enqueue PQ with c
23:      end for
24:   end if
25: end while
26: return R
```

One important question remains: how do we choose an initial set of locations? The Distance-Based Diverse Browsing algorithm is similar to other search algorithms/problems, where we traverse a hierarchal data structure (i.e. tree) till we reach a leaf node, create a solution, and then backtrack to find better solutions. The better the initial solution, the more pruning and less backtracking required to find a good solution, which is better for efficiency. Therefore, choosing k random locations would not provide an adequate solution. We could brute-force search for an initial solution, but this would be a very inefficient operation. Therefore, we would want to choose k locations that would provide a good initial solution of being close to the query point as well as spatially diverse, without incurring a large number of disk accesses. With this, we develop two heuristics based on the two metrics for k-nearest diverse neighbor. First, there is the Distance-First Distance-Based Diverse Browsing algorithm, which initially chooses locations close to the query point. Similarly, we develop the Diversity-First Distance-Based Diverse Browsing algorithm, which initially chooses a set of locations that are spatially diverse.

6.2.1 Distance-First Distance-Based Diverse Browsing

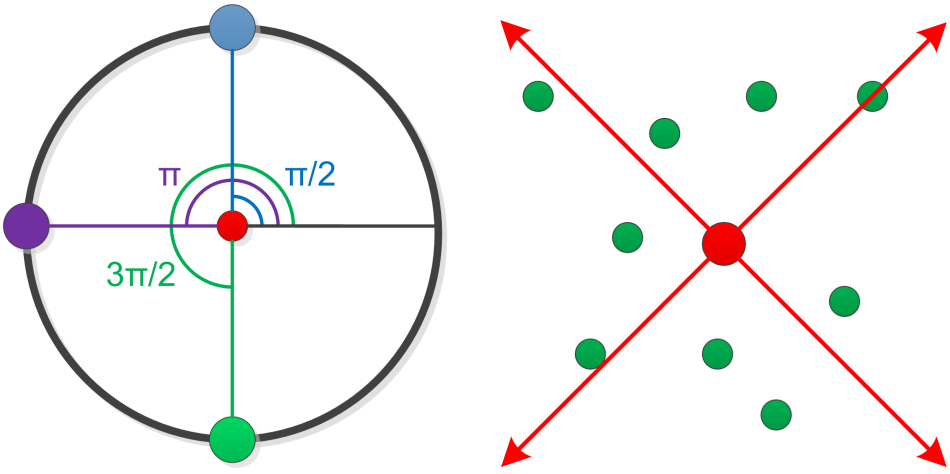
Algorithm 8 Choose Initial Locations: Distance-First

1: //Input: query point q, number of locations to recommend k, R-Tree root page *root*
2: //Output: initial solution set of recommended locations R
3: **return** kNearestNeighbors$(q, k, root)$

The Distance-First Distance-Based Diverse Browsing (Dist-DBDB) uses the heuristic of choosing an initial set of locations that are close in proximity to the user. By choosing the closest k locations to the query point, we can maximize the relevance metric in the objective function of DBDB. This operation can be performed via distance browsing, which is explained in Section 3.3.5.

6.2.2 Diversity-First Distance-Based Diverse Browsing

On the other hand, the Diversity-First Distance-Based Diverse Browsing (Div-DBDB) uses a mostly opposite approach to finding the initial set of locations. It chooses k locations that are spatially diverse, which allows for the diversity in the objective function of DBDB to receive a large score. Unlike Dist-DBDB, finding the optimal k locations to maximize the metric for angular diversity is a much harder problem. The k-most diverse locations can be calculated via brute-force search, but this becomes very inefficient, even for relatively small datasets. We develop an algorithm that calculates k angles that are diverse and chooses a location near each angle, which is shown in Algorithm 9.



(a) Angle Definition (b) Choosing Diverse Locations

Figure 6.4: Angle definition and visualization of choosing initial set of diverse locations

Instead of choosing k locations from all possible locations, we constrain the search to the n-nearest neighbors to the query point, where $n \geq k$. With these locations, we sort them by increasing angle in relation to the query point (Figure 6.4a shows a visualization of calculating

Algorithm 9 Choose Initial Locations: Diversity-First

1: //Input: query point q, number of locations to recommend k, R-Tree root page $root$, number of candidate locations n
2: //Output: initial solution set of recommended locations R
3: $NN = \text{kNearestNeighbors}(q, n, root)$
4: Sort NN in increasing angle in relation to q
5: Let θ_{start} be the angle of $NN[0]$ in relation to q
6: Add $NN[0]$ to R
7: **for** $i = 1$ to $k - 1$ **do**
8: Let $\theta = (\theta_{start} + \frac{i*2\pi}{k}) \mod 2\pi$
9: Use binary search to choose location $l \in NN$ that has angle closest to θ in relation to q
10: Add l to R
11: **end for**
12: **return** R

angle for three different points). To choose a first location, we select the first location in the array (smallest angle) and insert it into the chosen set. Then, we create k angles such that they are evenly spread (i.e. each angle is separated by $\frac{2\pi}{k}$) with one of the angles being the angle of the first selected location. Lastly, we select a location that is closest to each angle, and these k locations are the initial set of locations. Choosing one location can be completed in $O(\log n)$ using binary search on the sorted array. Since we choose locations that are near the spread angles, the chosen locations will also be spread, which would allow for higher spatial diversity. Figure 6.4b illustrates an example of executing the algorithm for choosing four locations. Since the location with the smallest angle is at about $\frac{\pi}{4}$, we can draw four evenly spaced vectors around the query point. The algorithm will choose the location that is closest to each vector.

If there are no locations on one side of the query point, there is a chance that a location may be closest to two angles/vectors. In the following case, we will select that location and another random location. Since the objective is to efficiently find a diverse solution, we do not want to spend much effort finding an additional location.

6.2.3 Example Calculation

To show how DBDB works, we perform an example problem shown in Figure 6.5. The larger, red circle towards the center is the query point while the other circles represent locations and the rectangles represent R-Tree pages. We will use Div-DBDB to choose four locations that are diverse. In addition, let the stopping condition in DBDB equal three and let $n = 10$ in Div-DBDB.

The first step is to choose four locations out of the ten shown in Figure 6.5. After sorting the locations in increasing order of angle in relation to the query point, l_7 is the first location chosen since it has the smallest angles. Using this angle, we create three other angles that are evenly spread (separated by $\frac{\pi}{2}$), which are denoted by the red dotted lines in Figure 6.6. Since l_3 is closest to the northern line, l_2 is closest to the western line, and l_9 is closest the southern line, the initial solution is $\{l_7, l_3, l_2, l_9\}$. Next, we use DBDB to consider locations to swap in

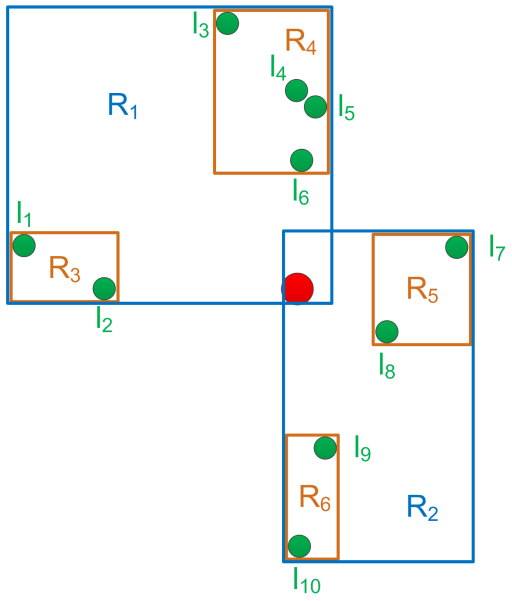


Figure 6.5: Example for Distance-Based Diverse Browsing

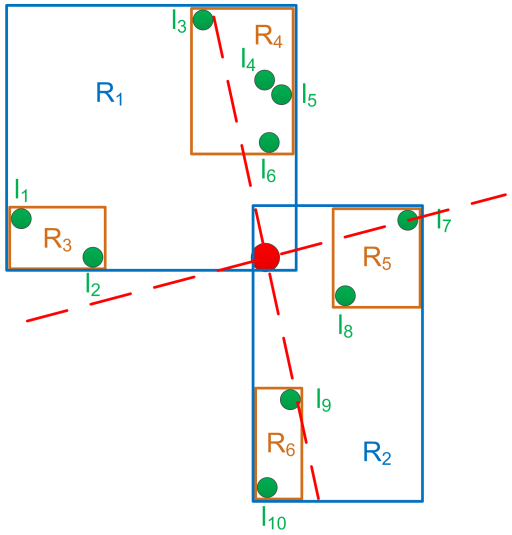


Figure 6.6: Choosing diverse initial set of locations

increasing order of distance from the query point, which is shown in Table 6.1. The table shows the step number, current solution, counter, location to be considered (candidate location), and a description for advancing to the next step. Since the distance and diversity metrics for the objective function have yet to be introduced, we will intuitively explain whether a location should be replaced.

Table 6.1: Distance-Based Diverse Browsing

Step #	Solution	Counter	Cand. Location	Description
1	l_7, l_3, l_2, l_9	0	l_8	Due to the fact that choosing l_8 would lessen the diversity score and replacing it with l_9 or l_7 would not bring a large change for the distance metric, we do not perform the replacement operation, so the counter is increased by 1.
2	l_7, l_3, l_2, l_9	1	l_6	Since l_6 is close in proximity to the query point and its angle is close to l_3, replacing l_6 for l_3 would increase the objective function, so we change the initial solution and reset the counter to 0.
3	l_7, l_6, l_2, l_9	0	l_9	l_9 is already in the current solution, so it is skipped.
4	l_7, l_6, l_2, l_9	0	l_7	l_7 is already in the current solution, so it is skipped.
5	l_7, l_6, l_2, l_9	0	l_5	Since l_5 would not increase the objective function due to its similar angle to l_6 and farther distance, the replacement operation is not performed, and the counter is incremented to 1.
6	l_7, l_6, l_2, l_9	1	l_2	l_2 is already in the current solution, so it is skipped.
7	l_7, l_6, l_2, l_9	1	l_4	Since l_4 would not increase the objective function due to its similar angle to l_6 and farther distance, the replacement operation is not performed, and the counter is incremented to 2.
8	l_7, l_6, l_2, l_9	2	l_{10}	Since l_{10} would not increase the objective function due to its similar angle to l_9 and farther distance, the replacement operation is not performed, and the counter is incremented to 3. Since this is the threshold for the stopping condition, the algorithm terminates and $\{l_7, l_6, l_2, l_9\}$ is returned.
9	l_7, l_6, l_2, l_9	3	–	End of algorithm.

Chapter 7

Performance Evaluation

In this chapter, we conduct comprehensive experiments with real datasets from Foursquare and Gowalla as well as synthetic datasets to validate our ideas and evaluate our proposed algorithms. In the first part, we detail the experiments of increasing the effectiveness of the location recommendation algorithm. For the second part, we show results related to choosing locations that are close in proximity to the user's current coordinate and spatially diverse.

7.1 Increasing Effectiveness

For increasing the effectiveness of location recommendation algorithms, our experiments are designed to achieve the following goals. (1) We want to study the optimal setting of the parameter α for UPS-CF under different distance ranges from home regions. This will allow us to see how similar users and social friends play a role in location recommendations under different distance ranges from home regions. (2) We want to compare UPS-CF against several collaborative filtering variants and baseline algorithms (i.e., distance-based and popularity-based), especially with their performance under different distance ranges from home regions. (3) We want to test how well the different algorithms perform for the cold start problem for users who have very few existing check-ins, which is a well-known problem for collaborative filtering algorithms.

7.1.1 Evaluation Process

For evaluating the recommendation algorithms, we adopt a widely used approach for data mining and machine learning research. Given the check-ins in the collected Foursquare and Gowalla datasets, the general idea is to mark off some data points in the datasets (e.g., a user u has visited a location l). Using the fact that u has a check-in at l as the ground truth, we evaluate how well the recommendation algorithms are able to recover the mark-off l in their location recommendations. Therefore, we can compare UPS-CF with other algorithms to see which ones provide better recommendations. The process for evaluation is as follows:

1. Randomly remove some check-in records that a user u has visited a location l (i.e., mark-off location l from u's check-in lists).

2. Randomly select a query location coordinate q (current standing location of u) that is distance d_{ql} away from location l. This allows our application scenario to be more realistic because u will most likely not be at the same geo-coordinate as l.

3. For each recommendation algorithm, recommend N locations for u to visit.

4. Track each recommendation algorithm to see if l was one of the N recommended locations.

When this process is complete, we calculate *precision*@N for each algorithm, i.e., the percentage that the removed locations were recovered when N locations are recommended. For our evaluation, we choose N to be 5, 10, and 20 and alternate d_{ql} between 5, 10, 20, 50, and 100 km. Notice that the predetermined pool of d_{ql} has a bias towards short distances (i.e., 5, 10 and 20 km). Intuitively, users will not travel long distances from their current standing location to a recommended location. For example, if a user is looking to eat at a restaurant, she will almost never travel hundreds of kilometers to go to a restaurant that far away; instead, she will almost always choose to eat at a place within a small driving distance. Thus, we do not set d_{ql} to be a very long distance.

In our evaluation, we compare two baseline algorithms and four variants of the collaborative filtering method.

1. **Most Visited (MV):** Based on popularity, the algorithm recommends the most visited locations.

2. **Closest Locations (CL):** Based on distance to travel, the algorithm recommends the closest locations to the user's current standing point.

3. **User-Based CF (U):** The user-based collaborative filtering algorithm (as explained in Section 3.3.2).

4. **User and Proximity-Based CF (UP):** The user-based collaborative filtering algorithm with the proximity constraint, i.e., it filters out locations outside the radius d_p of the current standing location of the target user.

5. **User and Social-Based CF (US):** The UPS-CF method (see below) without the proximity constraint.

6. **User, Proximity and Social-Based CF (UPS):** The proposed new collaborative framework (as explained in Section 5.1).

When we evaluate the algorithm, the user location pairs (u, l) are divided into short distance (0-20 km), medium distance (20-200 km), and long distance (200-1000 km) in accordance with the distance of a location l from the home region of user u. This will allow us to see not only

Table 7.1: Optimal α for US and UPS

		Short Distance	Medium Distance	Long Distance
Foursquare	US	0.8	0.1	0.1
	UPS	0.8	0.1	0.1
Gowalla	US	0.9	0.6	0.1
	UPS	0.9	0.1	0.1

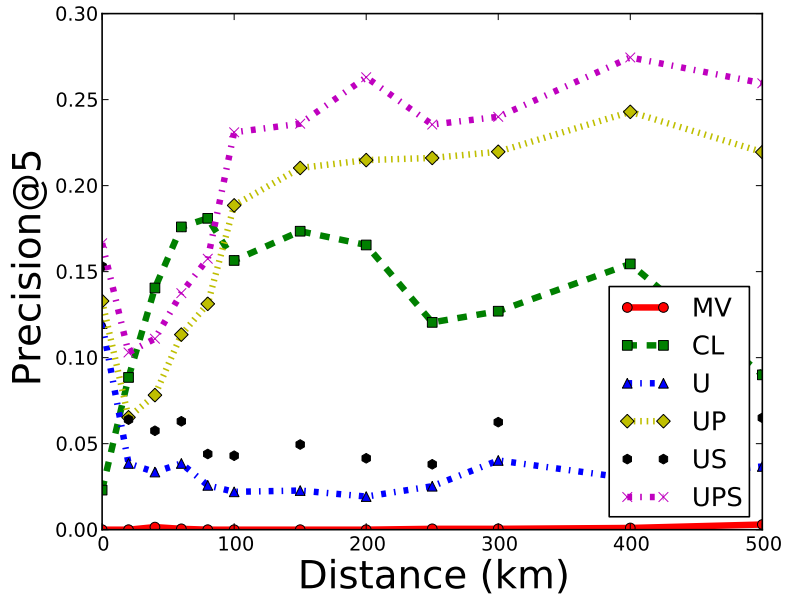
how the algorithms compare overall, but also how well the algorithms perform as the distance from their home region increases, i.e., from in-town scenario to out-of-town scenario. When we evaluate the algorithms, four-fold cross validation is performed, where $\frac{3}{4}$ of the data is the training data (to optimize parameters for US and UPS, as explained in Section 7.1.2), and $\frac{1}{4}$ is the testing data. After finding the optimal parameters, we iterate through each user location pair in the testing data as stated above to evaluate the different algorithms.

7.1.2 Parameter Tuning

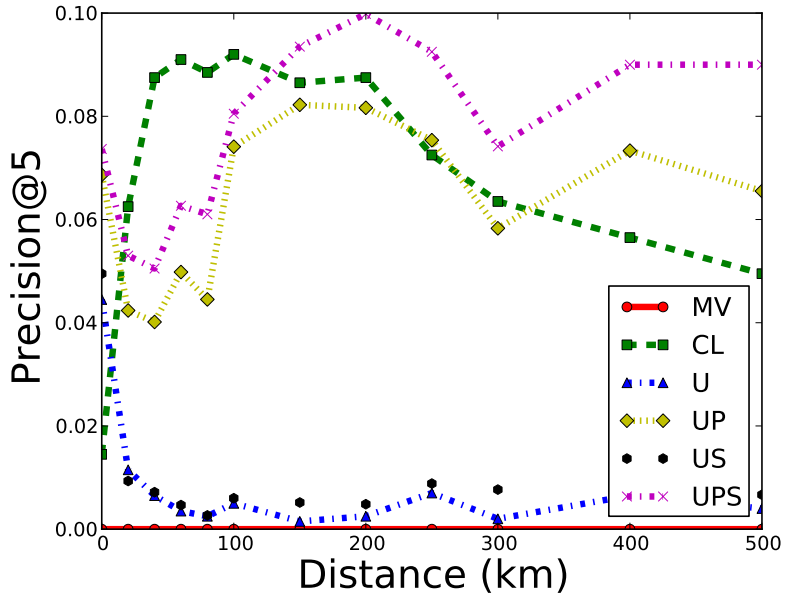
Before we can evaluate the collaborative filtering algorithms, we need to tune some parameters to optimize the effectiveness of the algorithms. In UP and UPS, we need to set the proximity constraint d_p for filtering out locations that are too far away. We set d_p to be 100 km so a reasonable number of candidate locations remain for recommendations while still being a realistic distance for a user traveling this distance to visit the location. On the other hand, for US and UPS, we empirically tune α to obtain its optimal settings for different distance ranges from the user's current location to her home location. Notice that α is the weight for the role of a similar user and $1 - \alpha$ is the weight for the role of a friend.

We use the average of *precision*@5, *precision*@10 and *precision*@20 as the overall performance metric to tune α for US and UPS. In the experiments, we divide the Foursquare and Gowalla datasets based on distance ranges from home into three categories, i.e., short (0-20 km), medium (20-200 km) and long (200-1000 km) distance. Table 7.1 shows the optimal settings for average precision of US and UPS under each category. Notice that a large α means the role of similar users are important while a small α means friends are important. For both US and UPS algorithms, the optimal α is large (0.8 for Foursquare and 0.9 for Gowalla) for the short distance category (i.e., users are in town). Nevertheless, the optimal α is small (0.1 for Foursquare and Gowalla) when users are farther away from other visited locations (i.e., users are out of town). In other words, when users are in town, similar users contribute more to effective recommendations while social friends play a more important role when users are out of town. The wide discrepancy between the in-town α and the out-of-town α shows that users have a different decision process for in-town and out-of-town scenarios. One possible explanation for a small α in out-of-town scenarios is that users may travel to various places to visit friends. Therefore, friends are important in this scenario.

7.1.3 Effectiveness

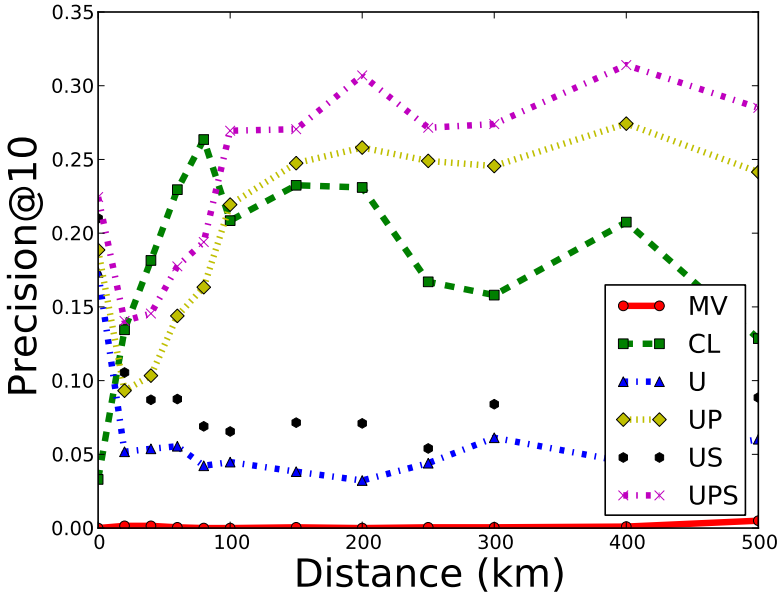


(a) Foursquare

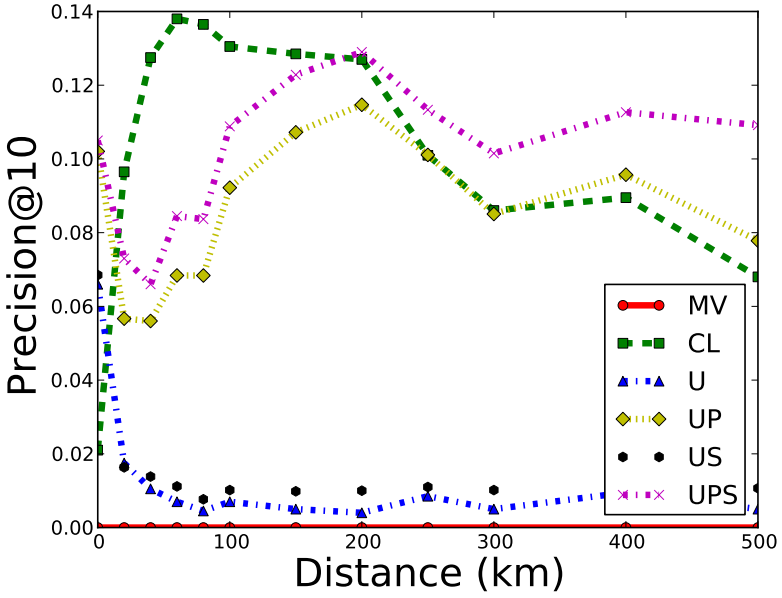


(b) Gowalla

Figure 7.1: Effectiveness of algorithms - Precision@5

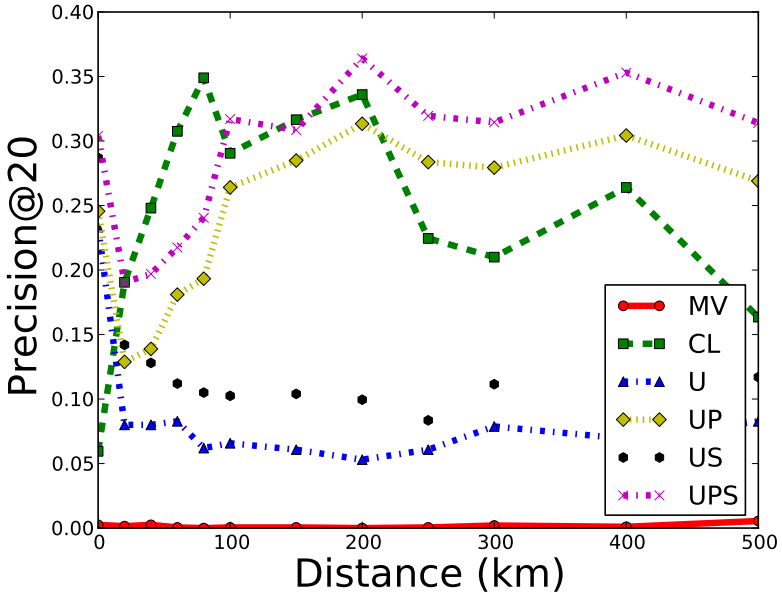


(a) Foursquare

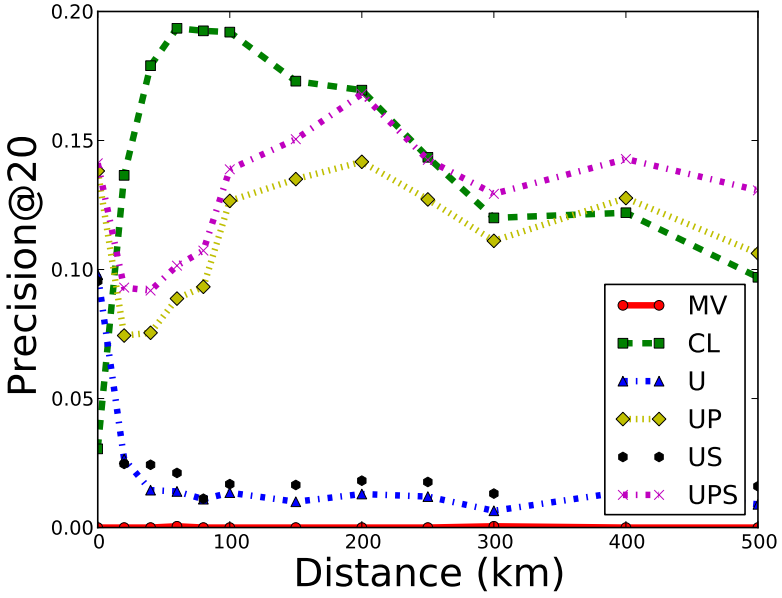


(b) Gowalla

Figure 7.2: Effectiveness of algorithms - Precision@10



(a) Foursquare



(b) Gowalla

Figure 7.3: Effectiveness of algorithms - Precision@20

Next, we evaluate the recommendation performance of UPS by comparing it with other approaches (i.e., US, UP, U, CL, and MV). The results are shown in Figures 7.1, 7.2 and 7.3, where each data point represents the average precision for corresponding distance ranges (e.g., 0-20 km, 20-40 km) from home. When the user is in the 0-20 km range (in town), UPS performs the best in both datasets, followed by US, UP, U, CL, and MV in the Foursquare dataset and UP, US, U, CL, and MV in the Gowalla dataset. Since UP performs better than U and UPS performs better than US, we conclude that removing locations farther away (i.e., the effect of proximity constraint) improves recommendation performance. In addition, UPS performs better than UP and US performs better than U, which means friends (i.e., the effect of social connections) also help for recommendations.

When we approach the 40-80 km range, we see that all of the collaborative filtering recommendation algorithms degrade in precision in both datasets. This can be explained by the observation that a collaborative filtering recommendation algorithm typically recommends locations near the home region of the target user. Since users typically check in to many places near their home location, the top-m users (those who visit many common locations as the target user) may likely live close to the home region of the target user and thus a large portion of the candidate locations to recommend will likely be near the home region. Since the visited locations are most likely within the 100 km radius, the proximity constraint will not remove these locations from consideration, i.e., all collaborative filtering recommendation algorithms will perform poorly in this experiment. Since CL and MV do not use user preference for recommendation, they do not have the sharp decrease in precision, which causes CL to perform well in the 20-200 km range.

When we get to a larger distance range (i.e., > 80, out of town), we see that the precision of UPS and UP strengthens while the precision of US and U degrades in both datasets, which shows that filtering by proximity constraint is very important in these scenarios. US and U do not perform well because some of the recommended locations may be too far away. Thus, if a user is on vacation or moving to a new location far away from his previous home region, US and U will give unsatisfactory recommendations. However, UPS and UP filter farther away locations and recommend only locations near the current location of the user. In addition, we see that UPS outperforms UP and US outperforms U, which means that social friends are important for recommendation. Since geographical proximity is important, CL performs average for larger distance range, but it does not take advantage of the similar users and social friends like UPS. Again, MV performs poorly, having at most one percent precision.

Therefore both filtering based on proximity constraint and social connections help to improve collaborative filtering (which exploits the power of implicit preferences among similar users). We observe that the difference between UP and UPS as well as U and US stays relatively constant while the difference between U and UP as well as US and UPS increases for larger distance range from home. This shows that the social factor affects the algorithms at a constant rate with the increase in distance from home while filtering based on radius provides a greater positive effect with the increase in distance from home. Therefore, unlike U and US, UPS's effectiveness does not deteriorate as the distance from home increases. In addition, we see that CL can sometimes

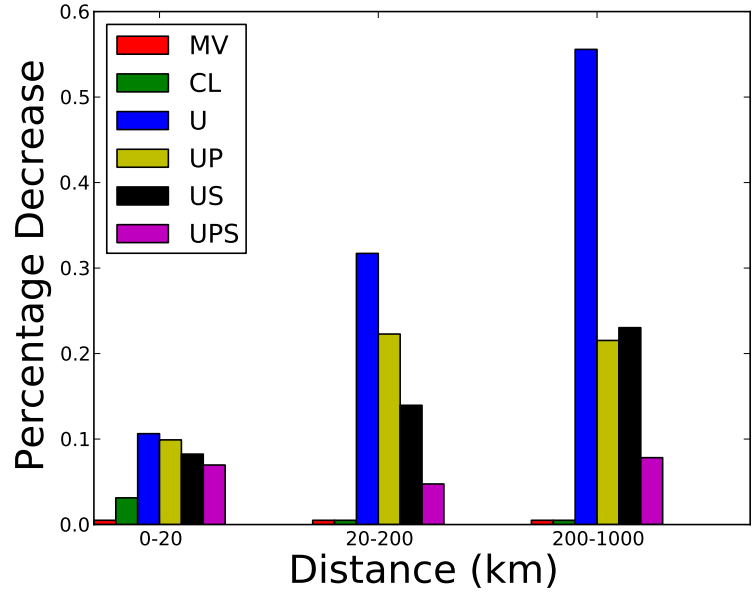
perform well since it takes advantage of the geographical proximity of locations, but it does not include user preference and social factor, which negatively affects its ability to recommend locations. Lastly, MV always performs extremely poorly for every distance range, with precision never increasing above one percent.

Between the Foursquare and Gowalla datasets, we observe a similar phenomenon when comparing the different algorithms. However, it is interesting that the precision of the algorithms in Foursquare is higher than the precision of the algorithms in Gowalla. Since the Gowalla dataset has over 6 times as many locations as the Foursquare dataset, the algorithms have more locations to choose from in the Gowalla dataset, which could cause a decrease in precision. In addition, the Gowalla dataset has 72.9% less friends per user than the Foursquare dataset, which could cause US and UPS to perform not as well. More experimentation is needed to give a more definitive answer for the decrease in precision between the Foursquare and Gowalla datasets.

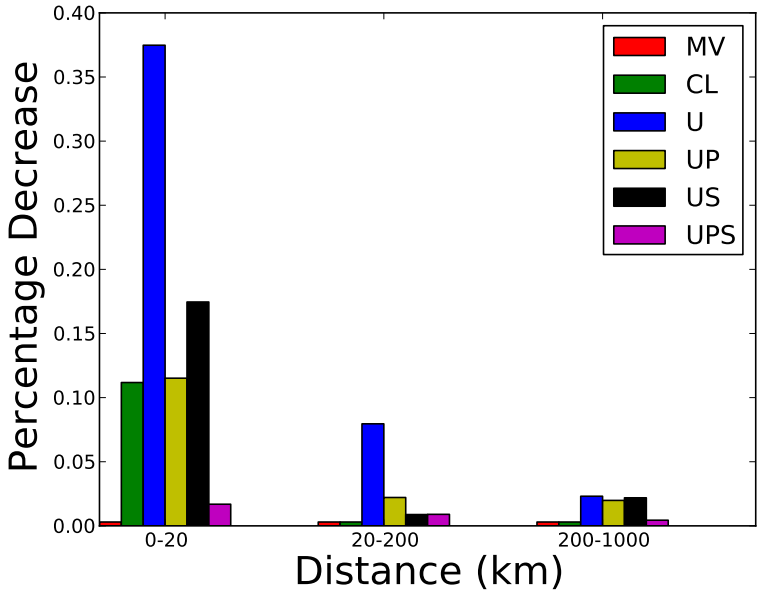
7.1.4 Cold Start Problem

Finally, we look into the effectiveness of the different recommendation algorithms for cold start users. Cold start is a problem in collaborative filtering algorithms that exists when a user has not yet visited a location or has visited only a few locations. Due to the lack of information about the users, recommendations could perform poorly. To see how our algorithms perform, we use the same training and testing sets as before, except that we keep at most 2 locations visited for each user in the testing sets. Figures 7.4, 7.5, and 7.6 show the results of the experiment, with each bar representing the percentage decrease between the regular and cold start effectiveness experiments for a recommendation algorithm for a certain distance range from home. With MV and CL, the precision stays relatively constant for most scenarios. This occurs because both of these algorithms do not use history to make recommendations, so a lack of history in the cold start problem has minimal effect. For the collaborative methods, we see a drop in precision, with the lowest percentage drop in almost all cases being UPS.

We can see a few interesting trends in the results for both the Foursquare and Gowalla datasets. First, we see that U has a consistently larger percentage drop than UP in addition to US having a consistently larger percentage drop than UPS. This shows that filtering by radius leads to a lower percentage decrease. Since filtering by radius removes locations that are too far away from the user, UP and UPS will have fewer incorrect candidate locations, even though the cold start situation has very little history. In addition, as the distance from home increases, US has a lower percentage decrease than U while UPS has a lower percentage decrease than UP. This occurs because the UPS and US algorithms also use social connections to help recommendations. Even though little history exists for which locations users visited, the social factor helps give a better recommendation. Lastly, we see that the combination of filtering based on the proximity constraint as well as using combined strengths of similar users and social friends helps UPS have the lowest percentage decrease among all collaborative filtering methods for medium and long distances.

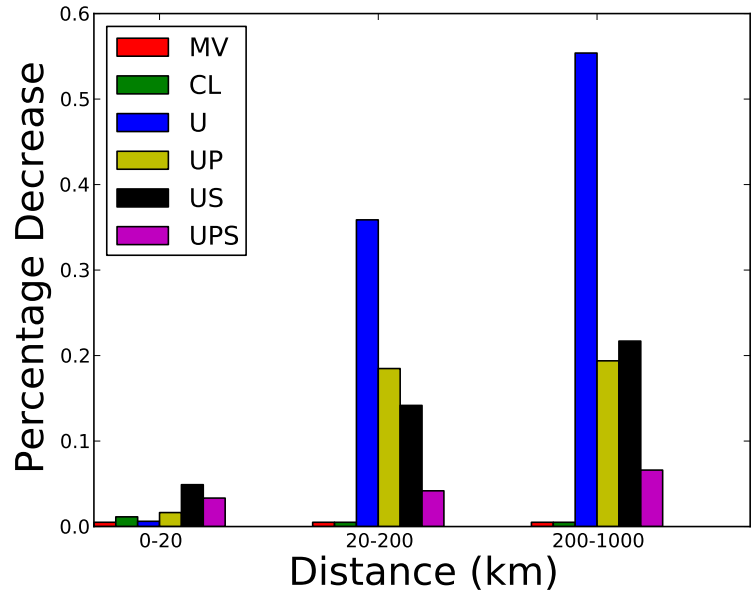


(a) Foursquare

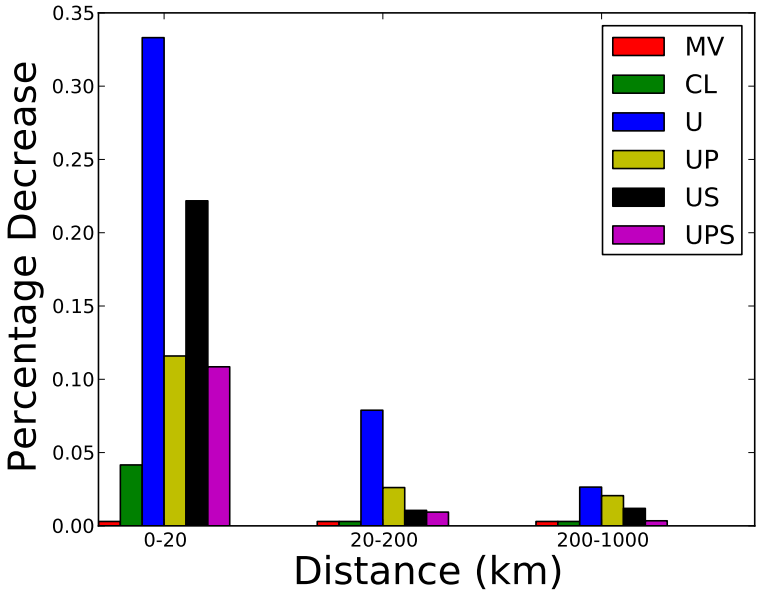


(b) Gowalla

Figure 7.4: Effectiveness of cold start users - Precision@5

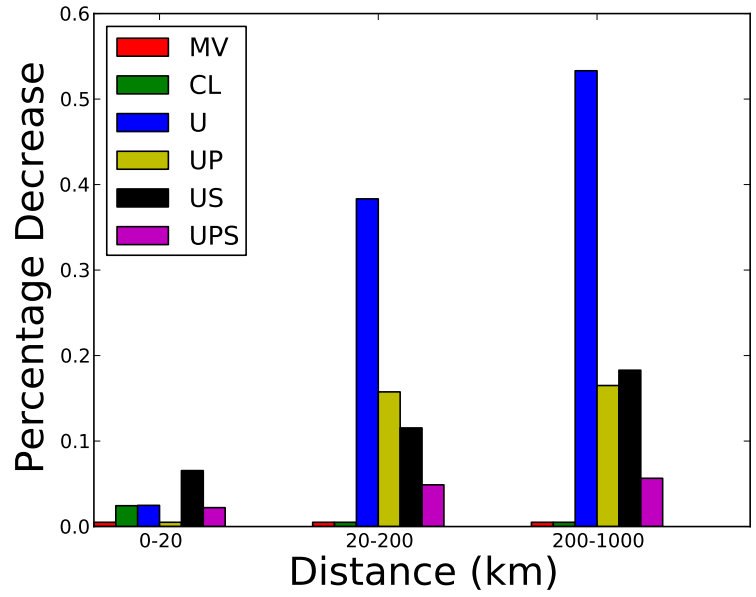


(a) Foursquare

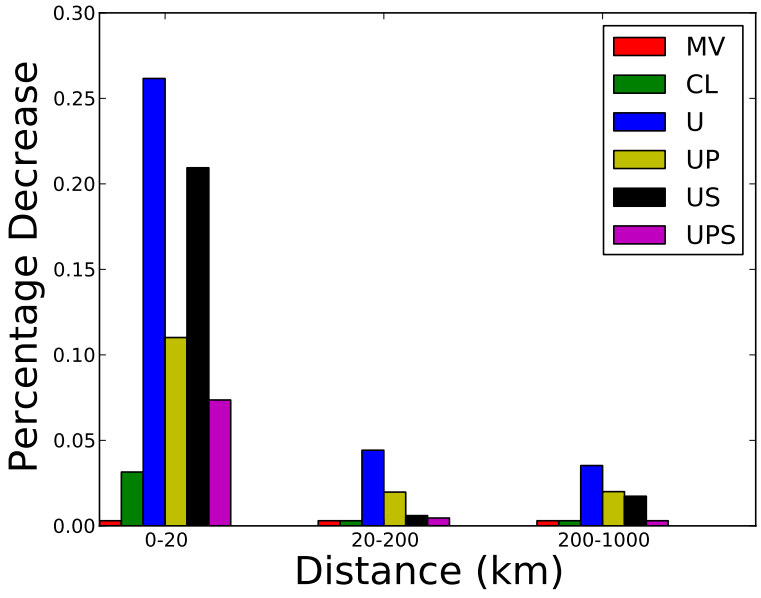


(b) Gowalla

Figure 7.5: Effectiveness of cold start users - Precision@10



(a) Foursquare



(b) Gowalla

Figure 7.6: Effectiveness of cold start users - Precision@20

7.2 K-Nearest Diverse Neighbor

For choosing locations close to the query point that are spatially diverse, we perform experiments using Foursquare, Gowalla and synthetic datasets. First, we explore the θ parameter in IBDB and Mod-IBDB, the *stoppingCondition* parameter in Dist-DBDB and Div-DBDB, and the size of locations n to choose an initial solution from in Div-DBDB. After we have chosen the optimal parameters, we compare the following algorithms and a baseline algorithm (i.e., k-nearest neighbor) with different λ to show which algorithms perform best.

7.2.1 Evaluation Process

For evaluating the k-nearest diverse neighbor algorithms, we implemented an R-Tree [23] for spatial data, which is explained in Section 3.3.4. In addition, the R-Tree is created using STR bulk loading [37]. With our own implementation, this allows us to have finer control of disk pages pages as well as to accurately keep track of disk accesses of the algorithm.

When executing experiments, we use the following process.

1. Choose control parameter λ to decide the importance of proximity to the query point versus spatial diversity. The larger the λ means that spatial diversity is more important.
2. Randomly choose a coordinate to be the query point of the user.
3. For each algorithm, choose k locations that are close in proximity to the query point and spatially diverse.

At the completion of the evaluation process, we will compare the results using different evaluation metrics explained in Section 7.2.2.

Our evaluation compares a baseline algorithm as well as four other algorithms for solving the k-nearest diverse neighbor problem.

1. **K-Nearest Neighbor (KNN):** The K-Nearest Neighbor algorithm, as explained in Section 3.3.5.
2. **Index-Based Diverse Browsing (IBDB):** The Index-Based Diverse Browsing algorithm, as explained in Section 3.3.6.
3. **Modified Index-Based Diverse Browsing (Mod-IBDB):** The proposed Modified Index-Based Diverse Browsing algorithm, as explained in Section 6.1.
4. **Distance-First Distance-Based Diverse Browsing (Dist-DBDB):** The proposed Distance-First Distance-Based Diverse Browsing framework, as explained in Section 6.2.1.
5. **Diversity-First Distance-Based Diverse Browsing (Div-DBDB):** The proposed Diversity-First Distance-Based Diverse Browsing framework, as explained in Section 6.2.2.

To complete the experiments, we have to choose optimal parameters, such as θ in IBDB and Mod-IBDB as well as *stoppingCondition* in Dist-DBDB and Div-DBDB. Using standard machine learning and data mining methods, we perform four-fold cross validation to optimize the parameters.

7.2.2 Evaluation Metrics

This subsection introduces the different evaluation metrics that are used to compare the different algorithms. First, we explain the diversity-relevance metric in Section 7.2.2.1. Next we introduce a new spatial diversity metric in Section 7.2.2.2, which fixes the deficiencies in other versions of the metric. Lastly, we explain the relevance and efficiency metrics in Section 7.2.2.3 and Section 7.2.2.4, respectively.

7.2.2.1 Diversity-Relevance Metric

The diversity-relevance metric (DivRel) is used to compare the linear combination of diversity as well as relevancy (proximity of locations to the query point) using control parameter λ. Letting R be the set of chosen locations, the diversity-relevance metric is defined as follows

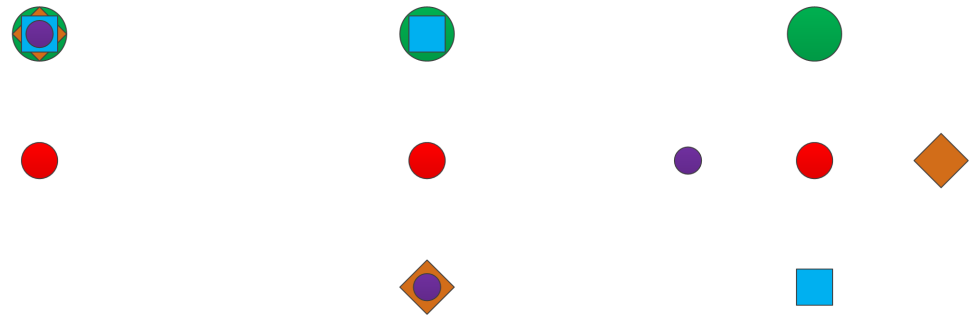
$$\text{DivRel}(q, R) = \lambda * \text{Div}(q, R) + (1 - \lambda) * \text{Rel}(q, R) \tag{7.1}$$

where Div is the diversity metric and Rel is the relevance metric. The diversity-relevance metric is equivalent to the objective function that we attempt to maximize for the k-nearest diverse neighbor problem. Therefore, the algorithm with the largest value for this metric is the algorithm with the best performance.

7.2.2.2 Diversity Metric

To evaluate the angular spatial diversity of chosen locations, we need a spatial diversity metric. From Section 3.3.3, there are three potential diversity metrics: Div_{min}, Div_{pair}, and Div_{mean}. Unfortunately, all of these diversity metrics have deficiencies that show that they are not adequate for our use.

Figure 7.7 shows three examples of four chosen locations (larger green circle - l_1, brown diamond - l_2, blue square - l_3, and smaller purple circle l_4) that are normalized (distance to query point is 1 km) around the query point (red circle). In the left example, the four locations are located at the same angle ($\frac{\pi}{2}$) in relation to the query point, which is the worst possible diversity for four locations. In the right example, the four locations are evenly spread (angles 0, $\frac{\pi}{2}$, π, and $\frac{3\pi}{2}$) in relation to the query point. Intuitively, this would be the best possible diversity for four locations since all locations are perfectly spread. In the middle example, there are two groups of two locations: two above the query point (angle is $\frac{\pi}{2}$) and two below the query point (angle is $\frac{3\pi}{2}$). Since the two groups are angularly far apart from each other, the example should have higher diversity, but each group has two locations at the same angle. Therefore, it should



(a) Low Spatial Diversity (b) Medium Spatial Diversity (c) High Spatial Diversity

Figure 7.7: Comparison of low, medium and high spatial diversity for angular diversity

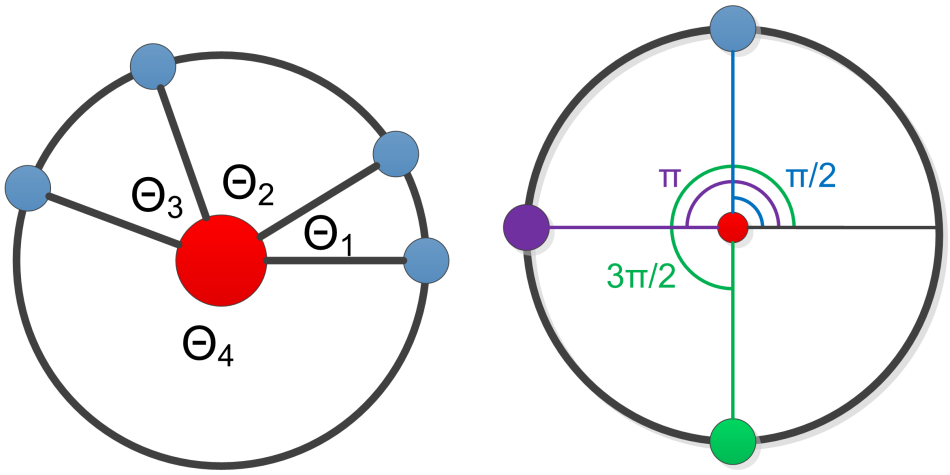
Table 7.2: Diversity calculations with different diversity metrics and examples

	Div_{min}	Div_{pair}	Div_{mean}
Left Example	$4 * \frac{0}{2\pi} = 0$	$\frac{0 * 6}{4 * 3 * \pi} = 0$	$1 - \frac{4}{4} = 0$
Center Example	$4 * \frac{0}{2\pi} = 0$	$\frac{0 * 2 + \pi * 4}{4 * 3 * \pi} = \frac{1}{3}$	$1 - \frac{0}{4} = 1$
Right Example	$4 * \frac{\pi/2}{2\pi} = 1$	$\frac{\frac{\pi}{2} * 4 + \pi * 2}{4 * 3 * \pi} = \frac{1}{3}$	$1 - \frac{0}{4} = 1$

have a diversity score larger than the left example but smaller than the right example. Let us calculate the diversity scores of each example using each of the three diversity scores, shown Table 7.2.

For Div_{min}, the left and center examples have the same diversity score since both examples have two locations with the same angle. The average angle for both the center and right examples is $\frac{\pi}{3}$, which causes Div_{pair} to return the same value. Lastly, for Div_{mean}, the center and right examples have equal diversity because the mean of the four locations is at the same point, which is where the query point is located. These answers go against the intuition of the left example having the least diversity, followed by the center example, followed by the right example. This provides the motivation to develop a new diversity metric.

We introduce the partition diversity metric Div_{part}. The intuition, as shown in Figure 7.8a, is to partition the unit circle into sectors, where each line connects a normalized location to the query point. For choosing k locations, the average angle will always be $\frac{2\pi}{k}$ since there are n angles and the angles sum to 2π. Therefore, the idea of the partition diversity metric is to



(a) Partition Diversity Metric (b) Angle Definition

Figure 7.8: Visualization of the partition diversity metric and angle definition

calculate the variance of the angles. The larger the variance between the angles, the smaller the diversity. When the locations are evenly spread around the unit circle, their angles tend to be near $\frac{2\pi}{k}$, which has small variance. However, when the locations are not diverse (i.e. locations group together into one area), there exists small and large angles, which causes the variance to be large.

To formally define the partition diversity metric, $\phi_{cc}(q, l_i, l_j)$ denotes the angle between $\overline{ql_i}$ and moving counter-clockwise to $\overline{ql_j}$, as shown in Figure 7.8b (this differs with $\phi(q, l_i, l_j)$ such that ϕ is the minimal angle created between $\overline{ql_i}$ and $\overline{ql_j}$, which means it is always less than or equal to π). In addition, when we consider locations $l_1, l_2, \cdots, l_k$, let $\phi_{cc}(q, l_0, l_1) = \phi_{cc}(q, l_k, l_1)$.

Property 7.2.1. *Let $l_1, l_2, \cdots, l_k$ be sorted in increasing angle in relation to the query point (*Angle(q, l) *as defined in Section 3.1). The following holds:*

$$\sum_{i=1}^{k} \phi_{cc}(q, l_{i-1}, l_i) = 2\pi \tag{7.2}$$

The property intuitively makes sense because completing one full trip around the circle is 2π. Assuming that the locations are sorted in increasing angle in relation to the query point (which will hold throughout the rest of the thesis), we can define the variances of the angles as follows.

$$\text{Var}_{part}(q, l_1, l_2, \cdots, l_k) = \frac{\sum_{i=1}^{k} (\frac{2\pi}{k} - \phi_{cc}(q, l_{i-1}, l_i))^2}{k} \tag{7.3}$$

As mentioned before, the larger the variance of the angles means the smaller the diversity. Therefore, our goal for choosing k diverse locations is to minimize the variance. One other approach for choosing k locations is to minimize the square of each $\phi_{cc}(q, l_{i-1}, l_i)$. If one of the angles between two consecutive locations is very large, it will provide a very large increase in the

score, which means that the locations are not as diverse. In actuality, these two approaches are equivalent.

Theorem 7.2.1. *For* $l_1, l_2, \cdots, l_k \in L$, $\text{Var}_{part}(q, l_1, l_2, \cdots, l_k)$ *is directly proportional to* $\sum_{i=1}^{k} \phi_{cc}(q, l_{i-1}, l_i))^2$.

Proof. Starting with $\text{Var}_{part}(q, l_1, l_2, \cdots, l_k)$ and squaring the summed term, we get:

$$\frac{\sum_{i=1}^{k}(\frac{2\pi}{k} - \phi_{cc}(q, l_{i-1}, l_i))^2}{k} = \frac{\sum_{i=1}^{k}((\frac{2\pi}{k})^2 - \frac{4\pi\phi_{cc}(q,l_{i-1},l_i)}{k} + \phi_{cc}(q, l_{i-1}, l_i)^2)}{k}$$

Since $(\frac{2\pi}{k})^2$ is a constant, we can move the factor outside of the summation. This yields:

$$\frac{\sum_{i=1}^{k}((\frac{2\pi}{k})^2 - \frac{4\pi\phi_{cc}(q,l_{i-1},l_i)}{k} + \phi_{cc}(q, l_{i-1}, l_i)^2)}{k} = \frac{\frac{4\pi^2}{k} + \sum_{i=1}^{k}(-\frac{4\pi\phi_{cc}(q,l_{i-1},l_i)}{k} + \phi_{cc}(q, l_{i-1}, l_i)^2)}{k}$$

Rewriting the equation to move the division by k to each term, we get:

$$\frac{\frac{4\pi^2}{k} + \sum_{i=1}^{k}(-\frac{4\pi\phi_{cc}(q,l_{i-1},l_i)}{k} + \phi_{cc}(q, l_{i-1}, l_i)^2)}{k} = \frac{4\pi^2}{k^2} + \sum_{i=1}^{k}(-\frac{4\pi\phi_{cc}(q, l_{i-1}, l_i)}{k^2} + \frac{\phi_{cc}(q, l_{i-1}, l_i)^2}{k})$$

Next, we split the two terms being summed and move constants outside of the summations, which yields:

$$\frac{4\pi^2}{k^2} + \sum_{i=1}^{k}(-\frac{4\pi\phi_{cc}(q, l_{i-1}, l_i)}{k^2} + \frac{\phi_{cc}(q, l_{i-1}, l_i)^2}{k}) = \frac{4\pi^2}{k^2} - \frac{4\pi}{k^2}\sum_{i=1}^{k}\phi_{cc}(q, l_{i-1}, l_i) + \sum_{i=1}^{k}\frac{\phi_{cc}(q, l_{i-1}, l_i)^2}{k}$$

Using Property 7.2.1, we replace $\sum_{i=1}^{k}\phi_{cc}(q, l_{i-1}, l_i)$ with 2π. Therefore, we get:

$$\frac{4\pi^2}{k^2} - \frac{4\pi}{k^2}\sum_{i=1}^{k}\phi_{cc}(q, l_{i-1}, l_i) + \sum_{i=1}^{k}\frac{\phi_{cc}(q, l_{i-1}, l_i)^2}{k} = \frac{4\pi^2}{k^2} - \frac{8\pi^2}{k^2} + \sum_{i=1}^{k}\frac{\phi_{cc}(q, l_{i-1}, l_i)^2}{k}$$

After combining the constant terms and moving $\frac{1}{k}$ to the outside of the summation, we get final result:

$$\frac{4\pi^2}{k^2} - \frac{8\pi^2}{k^2} + \sum_{i=1}^{k}\frac{\phi_{cc}(q, l_{i-1}, l_i)^2}{k} = \frac{-4\pi^2}{k^2} + \frac{1}{k}\sum_{i=1}^{k}\phi_{cc}(q, l_{i-1}, l_i)^2 \propto \sum_{i=1}^{k}\phi_{cc}(q, l_{i-1}, l_i)^2$$

□

Therefore, minimizing the variance of the angles is equivalent to minimizing the squares of each $\phi_{cc}(q, l_{i-1}, l_i)$.

Using this intuition that the smaller variance means larger diversity, we can define a normalized partition diversity metric as follows

$$\text{Div}_{part}(q, l_1, l_2, \cdots, l_k) = 1 - \frac{\text{Var}_{part}(q, l_1, l_2, \cdots, l_k) - \text{Var}_{part-min}(k)}{\text{Var}_{part-max}(k) - \text{Var}_{part-min}(k)} \tag{7.4}$$

where $\text{Var}_{part-min}(k)$ is the minimum possible variance of any k locations and $\text{Var}_{part-max}(k)$ is the maximum possible variance of any k locations. The only questions that remain are what are the minimum and maximum possible variance values for k locations. We will prove the minimum and maximum bound below.

Theorem 7.2.2. *For recommending k locations, the minimum variance (*$\text{Var}_{part-min}(k)$*) is 0.*

Proof. Since each term in the numerator of Equation 7.3 is squared and the denominator is a positive number, $\text{Var}_{part}(q, l_1, l_2, \cdots, l_k)$ cannot be a negative number. Let us consider the solution where $\text{Angle}(q, l_i) = \frac{(i-1)*2\pi}{k}$ for $i = 1, 2, \cdots, k$. Therefore, $\phi_{cc}(q, l_{i-1}, l_i) = \frac{2\pi}{k}$ for $i = 1, 2, \cdots, k$. Inserting the values into the variance equation, we get the following.

$$\text{Var}_{part}(q, l_1, l_2, \cdots, l_k) = \frac{\sum_{i=1}^{k}(\frac{2\pi}{k} - \phi_{cc}(q, l_{i-1}, l_i))^2}{k} = \frac{\sum_{i=1}^{k}(\frac{2\pi}{k} - \frac{2\pi}{k})^2}{k} = \frac{0}{k} = 0$$

Since there exists a solution with variance equaling 0 and no possible solution with negative variance, the minimum variance is 0. □

To prove the maximum possible variance for n locations, we need to prove the following lemmas first.

Lemma 7.2.3. *The solution $x_i = 2\pi$, $x_1 = x_2 = \cdots = x_{i-1} = x_{i+1} = \cdots = x_k = 0$ is a globally optimal solution for the quadratic programming problem:*

$$\begin{cases} \max \;\; x_1^2 + x_2^2 + \cdots + x_k^2 \\ \;\; s.t. \;\; x_1 + x_2 + \cdots + x_k = 2\pi \\ \qquad x_1, x_2, \ldots, x_k \geq 0 \end{cases}$$

Proof. Let us assume by contradiction that $x_i = 2\pi$, $x_1 = x_2 = \cdots = x_{i-1} = x_{i+1} = \cdots = x_k = 0$ is not a globally optimal solution. Since no other feasible solution exists with one or less $x_i > 0$, let there exist a globally optimal solution S with $x_1, \cdots, x_k$; $x_i, x_j > 0$; and $i \neq j$. Consider another distinct, feasible solution S' with $x'_i = x_i + x_j$; $x'_j = 0$; and $x'_k = x_k$ for $k \neq i, j$. Evaluating S' with the objective function, we get:

$$x_1^{'2} + x_2^{'2} + \cdots + x_i^{'2} + \cdots + x_j^{'2} + \cdots + x_k^{'2}$$

Replacing x'_i and x'_j as well as expanding results yield:

$$x_1^2 + x_2^2 + \cdots + (x_i + x_j)^2 + \cdots + 0^2 + \cdots + x_k^2 = x_1^2 + x_2^2 + \cdots + x_i^2 + 2x_i x_j + x_j^2 + \cdots + 0 + \cdots + x_k^2$$

Since $x_i, x_j > 0$, we know that $2x_ix_j > 0$. Therefore, we have:

$$x_1^2 + x_2^2 + \cdots + x_i^2 + 2x_ix_j + x_j^2 + \cdots + 0 + \cdots + x_k^2 > x_1^2 + x_2^2 + \cdots + x_i^2 + \cdots + x_j^2 + \cdots + x_k^2$$

The equation $x_1^2 + x_2^2 + \cdots + x_i^2 + \cdots + x_j^2 + \cdots + x_k^2$ is precisely the objective function for S. Therefore, the objective function for optimal solution S is smaller than the objective function for S', which is a contradiction because S is supposed to be optimal. Thus, the optimal solution cannot have two or more x_i greater than 0. The only feasible solution that satisfies this constraint is $x_i = 2\pi$, $x_1 = x_2 = \cdots = x_{i-1} = x_{i+1} = \cdots = x_k = 0$, which means it is a globally optimal solution. □

Lemma 7.2.4. *For recommending k locations, the maximum possible variance is achieved when* $\phi_{cc}(q, l_k, l_1) = 2\pi$ *and* $\phi_{cc}(q, l_{i-1}, l_i) = 0$ *for* $i = 2, 3, \cdots k$.

Proof. Since the variance of k locations is directly proportional to the squaring of the angles (as shown in Theorem 7.2.1), this problem is equivalent to maximizing $\phi_{cc}(q, l_{i-1}, l_i)^2$. The constraints for $\phi_{cc}(q, l_{i-1}, l_i)$ for $i = 1, 2, \cdots, k$ are that $\sum_{i=1}^{k} \phi_{cc}(q, l_{i-1}, l_i) = 2\pi$ and $\phi_{cc}(q, l_{i-1}, l_i) \geq 0$. Letting $x_i = \phi_{cc}(q, l_{i-1}, l_i)$, maximizing $\text{Var}_{part}(q, l_1, l_2, \cdots, l_k)$ is equivalent to solving the quadratic programming problem in Lemma 7.2.3. Since a solution for the quadratic programming problem is $x_i = 2\pi$, $x_1 = x_2 = \cdots = x_{i-1} = x_{i+1} = \cdots = x_k = 0$, the solution $\phi_{cc}(q, l_k, l_1) = 2\pi$ and $\phi_{cc}(q, l_1, l_2) = \phi_{cc}(q, l_2, l_3) = \cdots = \phi_{cc}(q, l_{n-1}, l_k) = 0$ maximizes $\text{Var}_{part}(q, l_1, l_2, \cdots, l_k)$. □

Theorem 7.2.5. *For recommending k locations, the maximum variance (*$\text{Var}_{part-min}(k)$*) is:*

$$\frac{4\pi^2(k-1)}{k^2}$$

Proof. From Lemma 7.2.4, the solution to maximizing variance for k locations is $\phi_{cc}(q, l_k, l_1) = 2\pi$ and $\phi_{cc}(q, l_{i-1}, l_i) = 0$ for $i = 2, 3, \cdots k$. When calculating $\text{Var}_{part}(q, l_1, l_2, \cdots, l_k)$, we get:

$$\text{Var}_{part}(q, l_1, l_2, \cdots, l_k) = \frac{\sum_{i=1}^{k}(\frac{2\pi}{k} - \phi_{cc}(q, l_{i-1}, l_i))^2}{k} = \frac{(\frac{2\pi}{k} - 2\pi)^2 + \sum_{i=2}^{k}(\frac{2\pi}{k} - 0)^2}{k} =$$
$$\frac{(\frac{2\pi}{k} - 2\pi)^2 + (k-1) * (\frac{2\pi}{k} - 0)^2}{k} = \frac{(\frac{2\pi}{k} - 2\pi)^2 + (k-1) * (\frac{2\pi}{k})^2}{k} =$$
$$\frac{\frac{4\pi^2}{k^2} - \frac{8\pi^2}{k} + 4\pi^2 + \frac{4\pi^2}{k} - \frac{4\pi^2}{k^2}}{k} = \frac{4\pi^2(1 - \frac{1}{k})}{k} = \frac{4\pi^2(k-1)}{k^2}$$

□

Using Theorems 7.2.2 and 7.2.5 we can rewrite the definition of Div_{part} in Equation 7.4 as follows.

$$\text{Div}_{part}(q, l_1, ..., l_k) = 1 - \frac{\text{Var}_{part}(q, l_1, ..., l_k)}{\frac{4\pi^2(k-1)}{k^2}} = 1 - \frac{k^2 * \text{Var}_{part}(q, l_1, ..., l_k)}{4\pi^2(k-1)} \tag{7.5}$$

Table 7.3: Diversity calculations with examples for the partition diversity metric

	Div_{part}
Left Example	$1 - \frac{16 * \frac{1}{4}((\frac{2\pi}{4} - 2\pi)^2 + 3(\frac{2\pi}{4} - 0)^2)}{12\pi^2} = 1 - \frac{4(\frac{9\pi^2}{4} + \frac{3\pi^2}{4})}{12\pi^2} = 1 - 1 = 0$
Center Example	$1 - \frac{16 * \frac{1}{4}(2(\frac{2\pi}{4} - 0)^2 + 2(\frac{2\pi}{4} - \pi)^2)}{12\pi^2} = 1 - \frac{4(\frac{\pi^2}{2} + \frac{\pi^2}{2})}{12\pi^2} = 1 - \frac{1}{3} = \frac{2}{3}$
Right Example	$1 - \frac{16 * \frac{1}{4} * 4(\frac{2\pi}{4} - \frac{\pi}{2})^2}{12\pi^2} = 1 - 0 = 1$

To compare the partition diversity metric against the other metrics, Table 7.3 shows the calculation of the partition diversity metric for the three examples in Figure 7.7. Unlike the other metrics, the results for Div_{part} agree with the logic that the left example has the smallest diversity, the right example has the highest diversity, and the middle example is in between. In addition, the metric is correctly normalized since the smallest possible diversity has score 0, and the largest possible diversity has score 1 for four locations. Therefore, the following experiments will use Div_{part}. All future references to the diversity metric are equivalent to the partition diversity metric.

7.2.2.3 Relevance Metric

The relevance metric for the k-nearest diverse neighbor problem measures how close in proximity the locations are to the query point. The metric should be normalized so that choosing the k-nearest neighbors to q gives a score of 1 and the farther away the locations are, the smaller the score. We adopt the relevance metric used in previous papers [34, 35]. Letting K be the k-nearest neighbors of query point q, the relevance of the set of locations R (where $|R| = k$) is as follows.

$$\text{Rel}(q, R) = \frac{\sum_{l \in K} \text{Dist}(q, l)}{\sum_{l \in R} \text{Dist}(q, l)} \tag{7.6}$$

7.2.2.4 Efficiency Metric

Since the bottleneck operation in database systems is accessing the disk, the efficiency metric used in this thesis is the number of disk pages accessed per query. Since we created our own R-Tree, we can easily count the number of disk accesses, without worrying about the database management system performing background operations that can incur more disk I/O. Like the other metrics, this efficiency metric is system independent, which means that executing the same

experiments on different systems will provide the same solution. This is different than other efficiency metrics, such as execution time.

7.2.3 Parameter Comparison

Before we compare the algorithms for solving the k-nearest diverse neighbor problem, we need to tune parameters that are used in the algorithms. For IBDB and Mod-IBDB, we need to set the θ parameter for determining the threshold for a pair of locations to be diverse. In Div-DBDB and Dist-DBDB, we need to tune the *stoppingCondition* parameter for deciding how long the algorithms should execute before they terminate. Lastly, we need to set the number of locations to consider for selecting the initial solution for Div-DBDB. Using the training data in the evaluation, we aim to empirically tune these parameters based on maximizing the diversity-relevance metric, which is the objective function for the k-nearest diverse neighbor problem.

For the IBDB and Mod-IBDB algorithms, we change the θ parameter to decide the threshold for a pair of locations to be relatively diverse. Figures 7.9, 7.10, 7.11 and 7.12 show the results for changing θ for $\lambda = 0.5$ for diversity-relevance, disk accesses, diversity, and relevance, respectively. Generally, the performance and efficiency of IBDB and Mod-IBDB are similar when θ is small (i.e., 0.1), but as θ increases, the performance and efficiency of IBDB degrades while the performance of Mod-IBDB either stays constant or slightly improves. For performance, this phenomenon can be explained because IBDB performs the extra pruning operation. Even though a location may have a small *mindivdist* since it is close in proximity to the query point, it may be pruned because the location is nearby an already chosen location. This explains why the relevance metric for IBDB has a large decrease for an increasing θ. Since the pruning was added to IBDB to choose more diverse results, the diversity metric increases more for IBDB than Mod-IBDB, but the increase in diversity for IBDB does not match its decrease in relevance. For efficiency, both algorithms show an increase in disk accesses with the increase in the threshold of diversity since locations are considered less diverse, which means more pages will be accessed to search for locations that are diverse. IBDB has a steeper increase in disk accesses, which can be contributed to pruning locations (versus pruning R-Tree pages) that already have been accessed from disk and otherwise would have been selected in the final solution.

Next, we analyze changing the stopping condition parameter for the Dist-DBDB and Div-DBDB algorithms. For Div-DBDB, we kept the number of initial locations to consider to be 50. The results are shown in Figures 7.13, 7.14, 7.15 and 7.16 for diversity-relevance, disk accesses, diversity, and relevance, respectively, using $\lambda = 0.5$. In the performance metrics, Dist-DBDB and Div-DBDB converge to about the same value, which occurs since the same swapping operations are attempted in both operations. Since Dist-DBDB has an initial solution with high relevance, it starts with a higher relevance score and vice versa for Div-DBDB with the diversity metric. Lastly, since it takes more disk accesses to find an initial diverse solution, Div-DBDB originally uses more disk accesses, but Div-DBDB will eventually attempt replacement operations with all the locations that Dist-DBDB considered for its initial solution, so they will eventually obtain

the same number of disk accesses for equal stopping conditions.

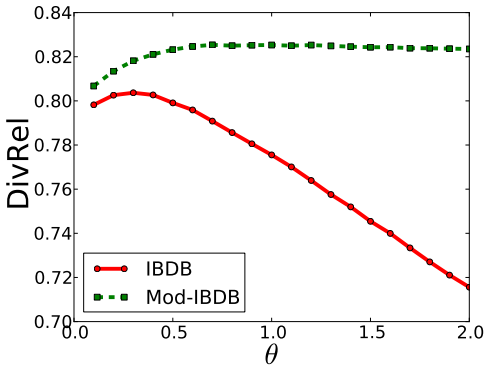
Even though both Dist-DBDB and Div-DBDB merge to solutions that provide very similar results, the advantage of having two different algorithms is the quickness in merging for solutions that have high relevancy or diversity. If the user wants a solution that emphasizes relevance over diversity (small λ), Dist-DBDB already chooses an initial solution with high relevancy, so it should quickly converge to a good solution. Even if the user wants to quickly choose the k-nearest diverse neighbor (i.e. before the algorithms converge), Dist-DBDB should have the better solution. Similarly, if the user wants a solution that emphasizes diversity (large λ), Div-DBDB should quickly converge to a good solution since its initial solution has high diversity. To compare the initial solutions, we calculate the average DivRel for $\lambda = 0.1$ (diversity not too important) and $\lambda = 0.9$ (diversity important) with Foursquare and Gowalla datasets. For $\lambda = 0.1$, the DivRel for the initial solution for Dist-DBDB is 0.958 and 0.960 while the DivRel for the initial solution for Div-DBDB is 0.464 and 0.493 for the Foursquare and Gowalla datasets, respectively. On the other hand, for $\lambda = 0.9$, the values for Div-DBDB are 0.854 and 0.862 while the values for Dist-DBDB are 0.621 and 0.639. This shows how the initial solution for Dist-DBDB has high relevance but low diversity while the initial solution for Div-DBDB has high diversity and low relevance.

Lastly, we analyze the initial size of candidate locations for choosing a diverse set of locations for Div-DBDB, keeping the stopping condition constant at 50. Figures 7.17 and 7.18 show the diversity and number disk accesses for calculating the initial set of diverse locations. As expected, the larger the size of candidate locations, the more diverse the result as well as the more disk accesses. However, the diversity tends to level off while the number of disk accesses increases linearly. Therefore, at some point, increasing the size will not give a much better solution while incurring more disk accesses.

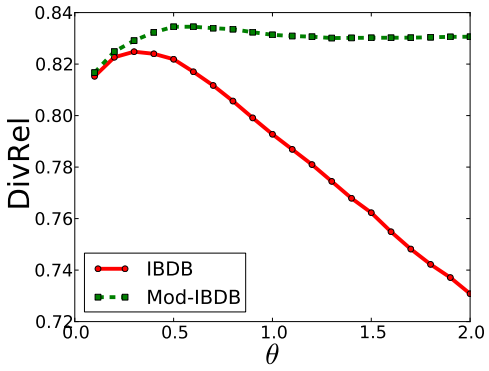
For tuning the parameters of the algorithms for the following experiments, we use four-fold cross validation (a standard in machine learning and data mining). For θ for IBDB and Mod-IBDB, we perform k-nearest neighbor queries with uniform λ for each θ value on the training data. The θ that maximizes the DivRel metric is chosen for the experiments. With the DBDB approaches, we need to choose the stopping condition number, but a problem occurs in choosing the optimal value since a larger stopping condition will always provide the same or better results. In addition, for a small λ the stopping condition can be small because nearby locations are emphasized more and the DBDB algorithms explore locations in increasing distance from the query point. When there is a large λ, the stopping condition needs to be larger to search for locations that increase the diversity of the solution. To tune the stopping condition parameter, we first find the value of the DivRel metric for $\lambda = 1.0$ (importance with diversity) for a sufficiently large stopping condition. Then, we perform queries for $\lambda = 1.0$ to find the smallest stopping condition $smallestStopCond$ that is within 99% of the value for the large stopping condition. Lastly, we dynamically set the stopping condition $stopCond$ for experiments using the function $\text{stopCond}(\lambda) = smallestStopCond * \lambda^2$ to increase the stopping condition as λ increases. Lastly, we have to tune initial size of candidate locations for choosing a diverse set of locations for

Div-DBDB. As with the stopping condition, the initial size should be smaller for smaller λ since choosing diverse locations is not as important. Similarly to the stopping condition, we choose an initial size $smallestInit$ for queries when $\lambda = 1.0$. Then, we dynamically set the initial size $init$ with the function $\text{init}(\lambda) = smallestInit * \lambda^2$.

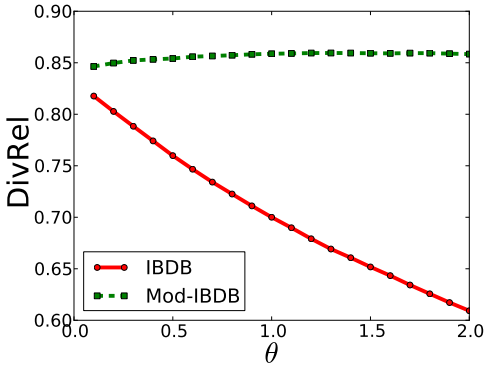
Since there are two variables (stopping condition and size of candidate locations for initial solution) for Div-DBDB to tune, we set each variable assuming that the other is constant due to the quadratic increase in the number of possible parameter combinations.



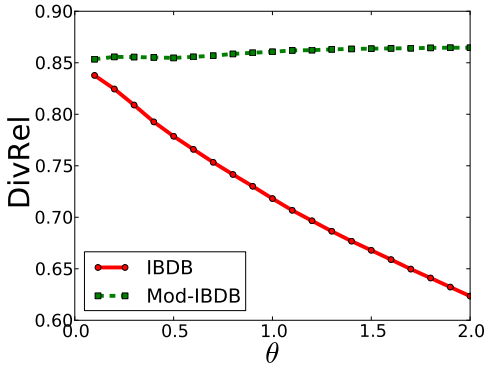
(a) 5 Locations - Foursquare



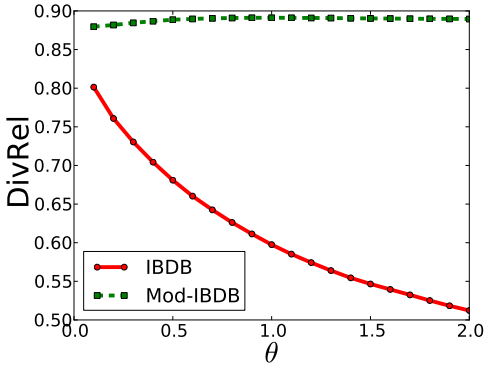
(b) 5 Locations - Gowalla



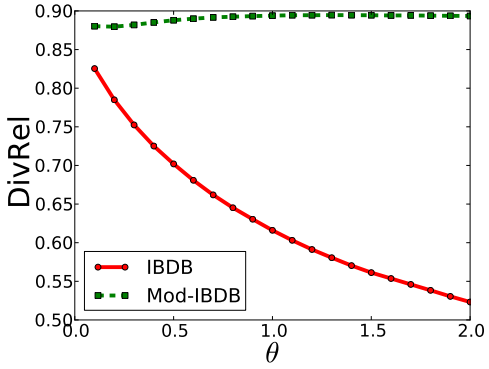
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

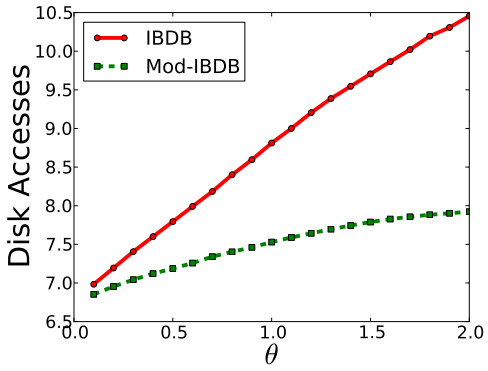


(e) 20 Locations - Foursquare

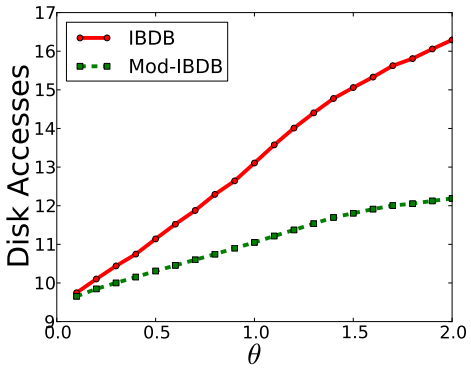


(f) 20 Locations - Gowalla

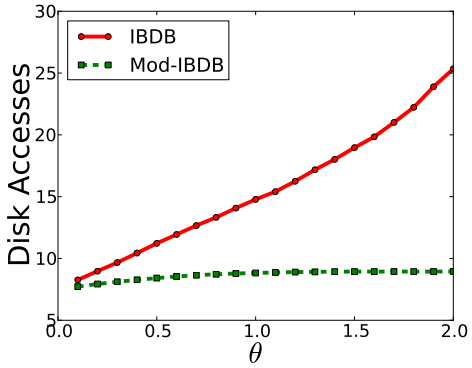
Figure 7.9: DivRel score for changing θ



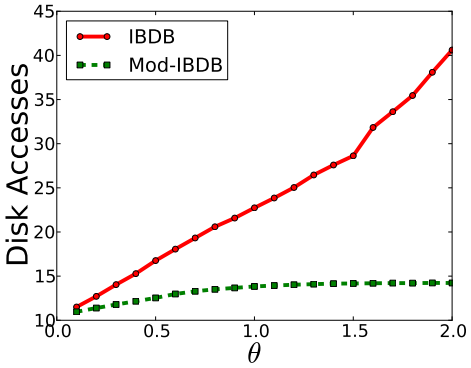
(a) 5 Locations - Foursquare



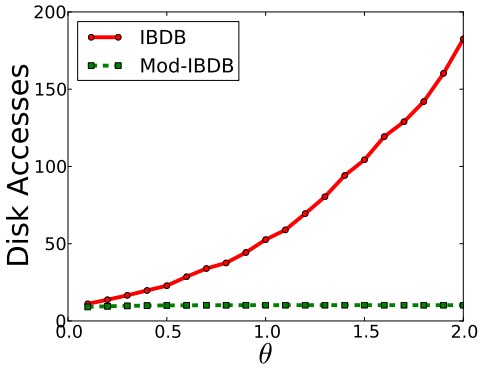
(b) 5 Locations - Gowalla



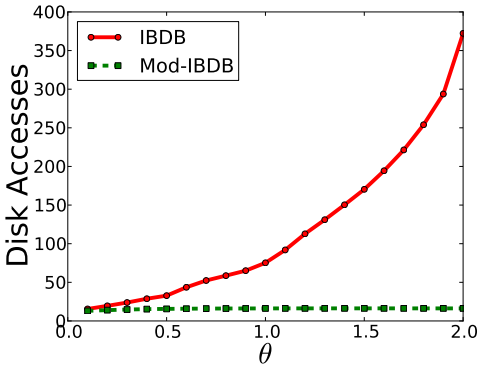
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

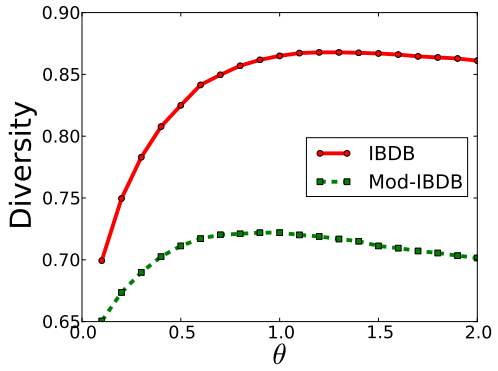


(e) 20 Locations - Foursquare

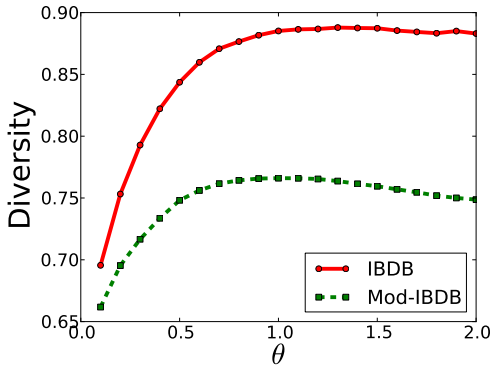


(f) 20 Locations - Gowalla

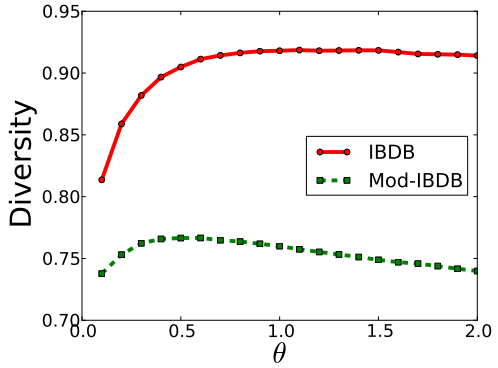
Figure 7.10: Disk accesses for changing θ



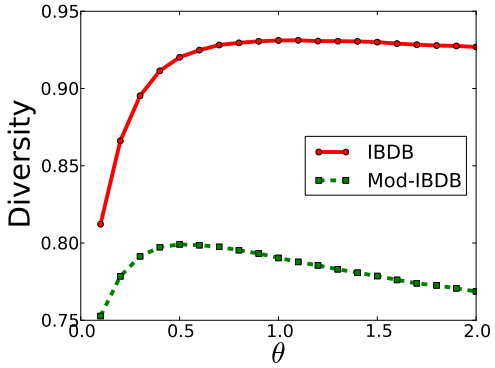
(a) 5 Locations - Foursquare



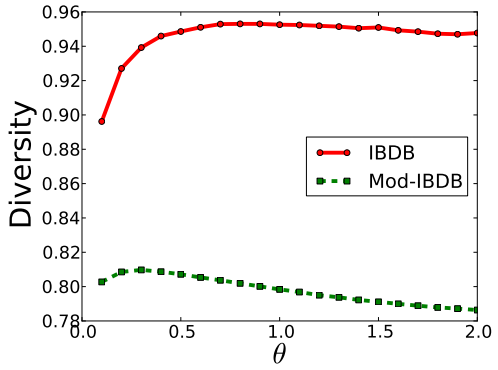
(b) 5 Locations - Gowalla



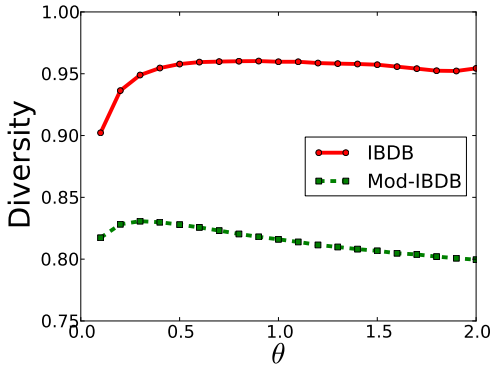
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

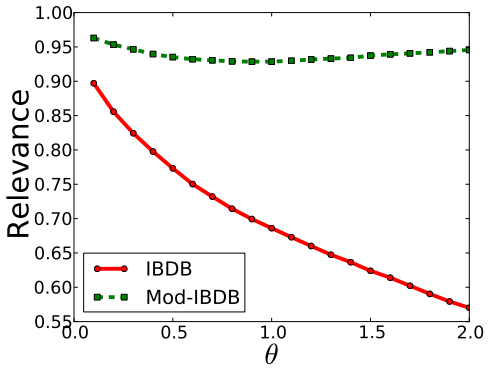


(e) 20 Locations - Foursquare

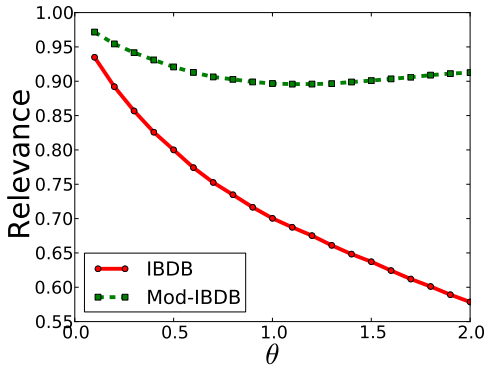


(f) 20 Locations - Gowalla

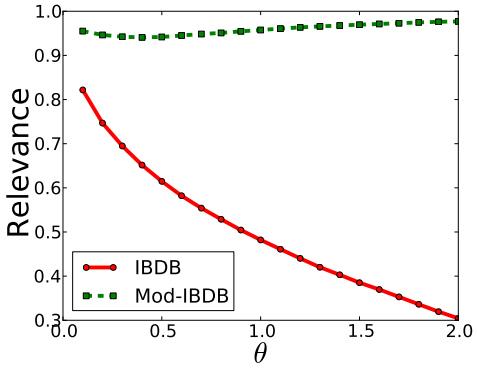
Figure 7.11: Diversity score for changing θ



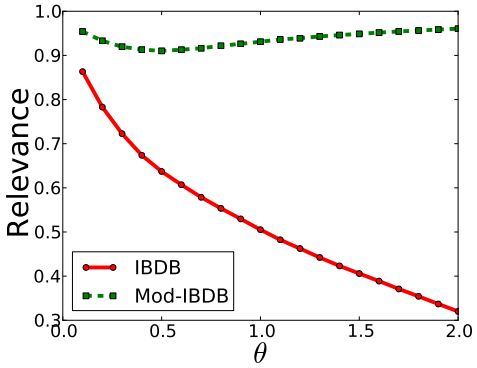
(a) 5 Locations - Foursquare



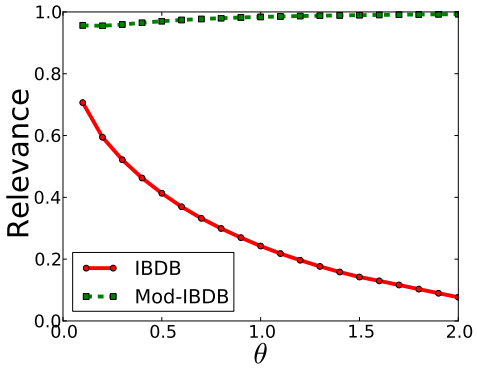
(b) 5 Locations - Gowalla



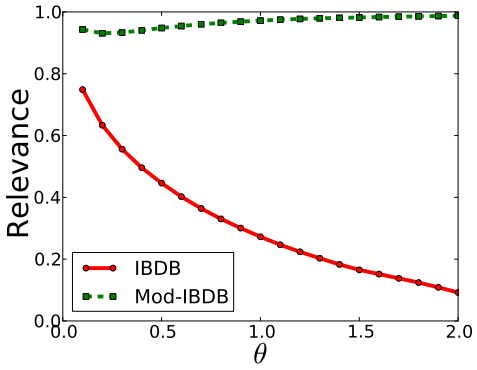
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

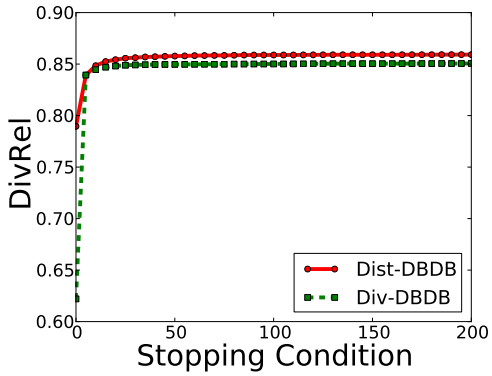


(e) 20 Locations - Foursquare

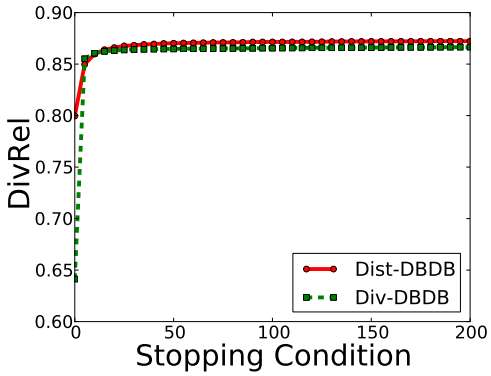


(f) 20 Locations - Gowalla

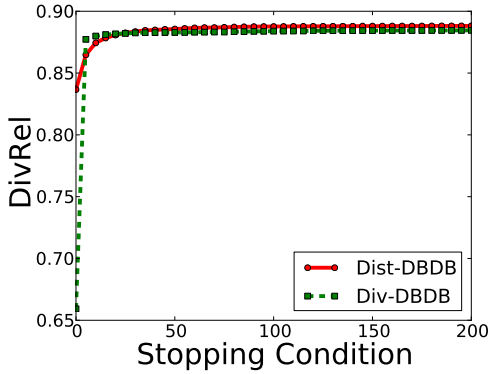
Figure 7.12: Relevance score for changing θ



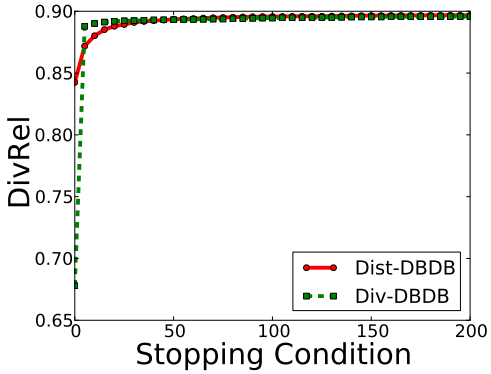
(a) 5 Locations - Foursquare



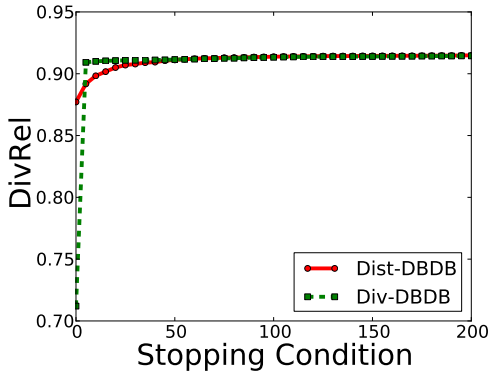
(b) 5 Locations - Gowalla



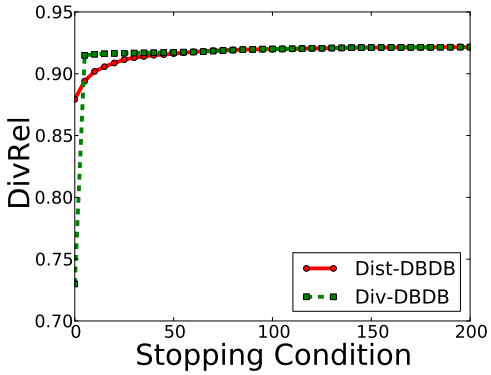
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

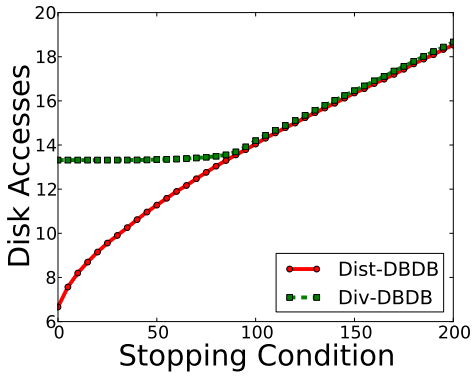


(e) 20 Locations - Foursquare

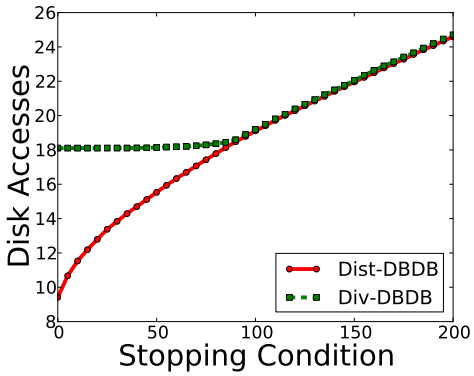


(f) 20 Locations - Gowalla

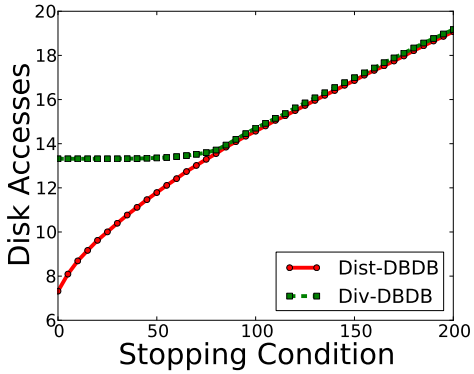
Figure 7.13: DivRel score for changing stop condition



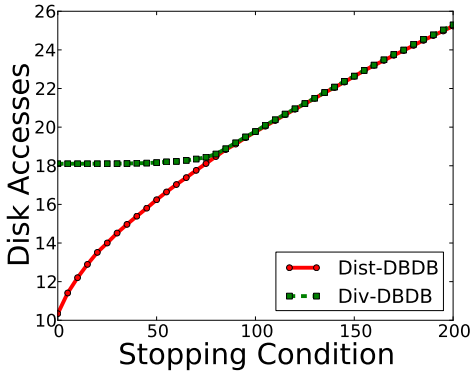
(a) 5 Locations - Foursquare



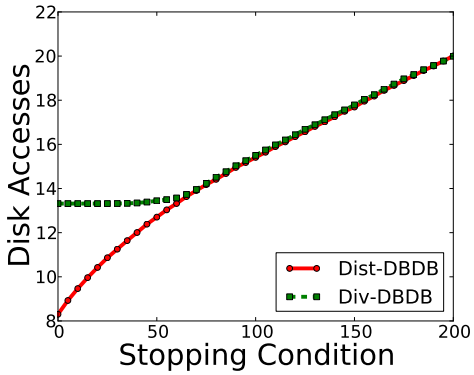
(b) 5 Locations - Gowalla



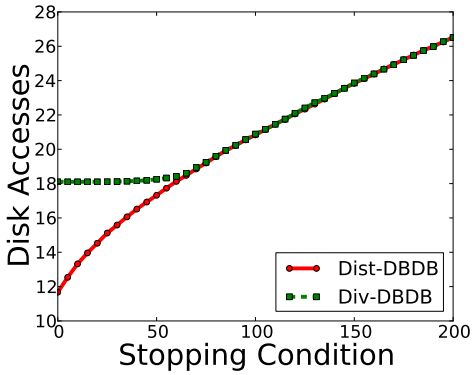
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

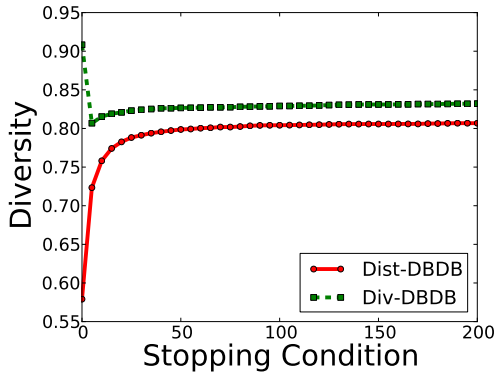


(e) 20 Locations - Foursquare

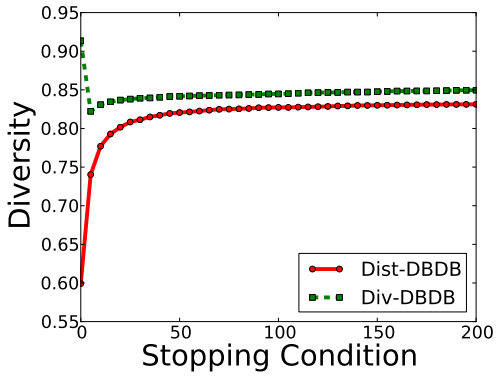


(f) 20 Locations - Gowalla

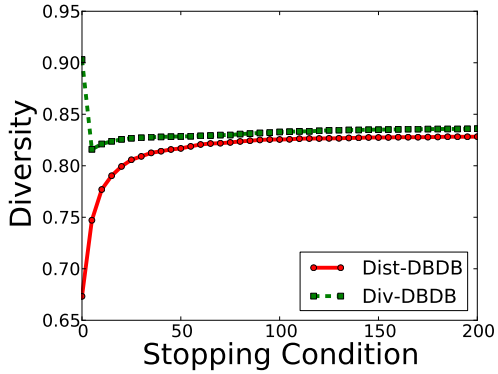
Figure 7.14: Disk accesses for changing stop condition



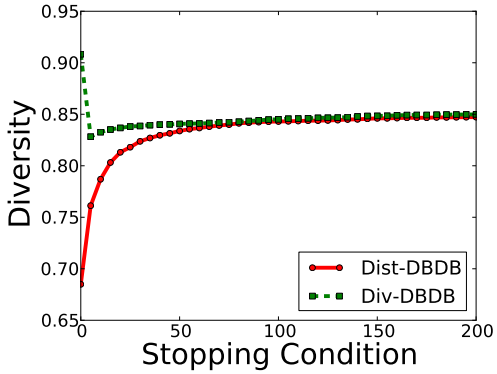
(a) 5 Locations - Foursquare



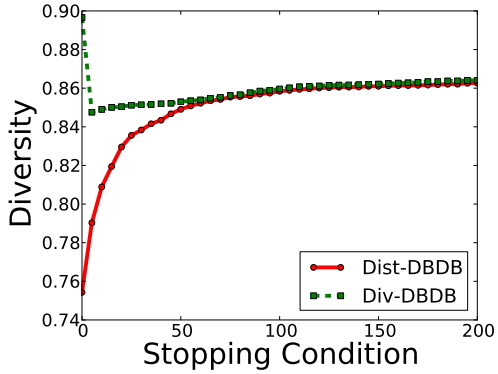
(b) 5 Locations - Gowalla



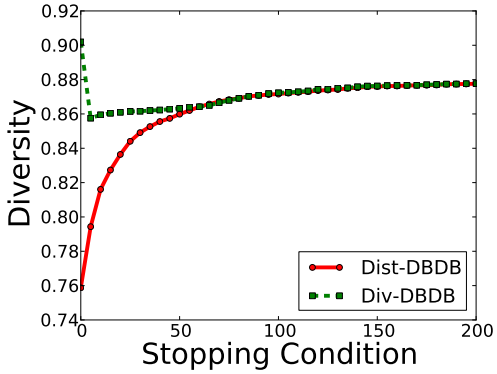
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

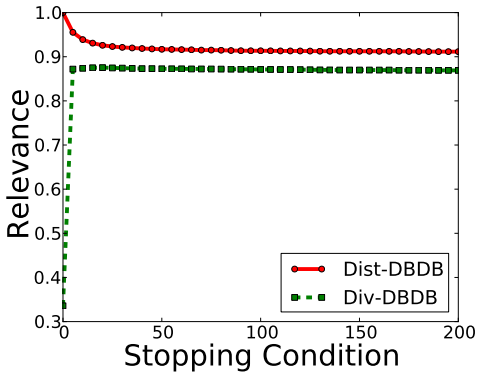


(e) 20 Locations - Foursquare

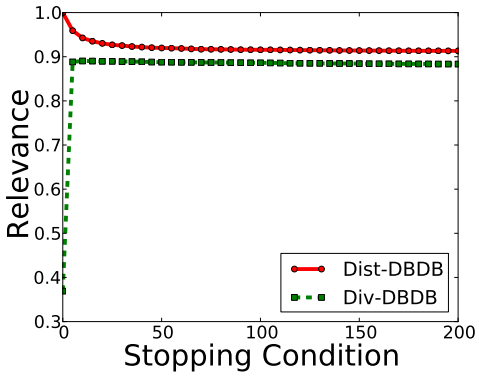


(f) 20 Locations - Gowalla

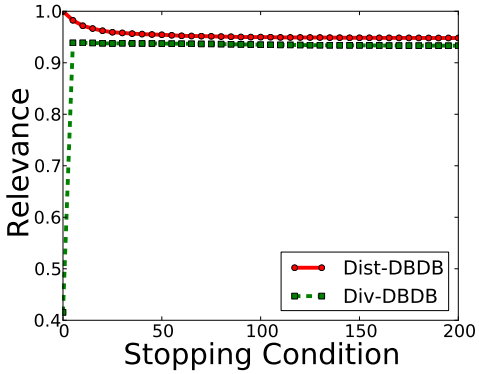
Figure 7.15: Diversity score for changing stop condition



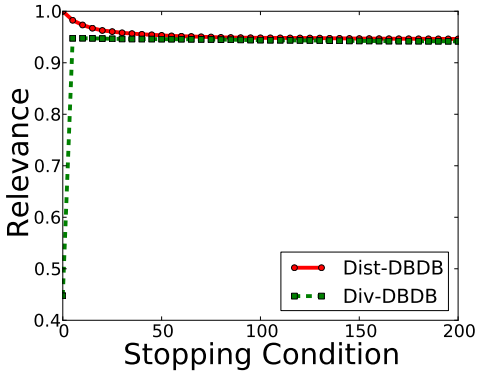
(a) 5 Locations - Foursquare



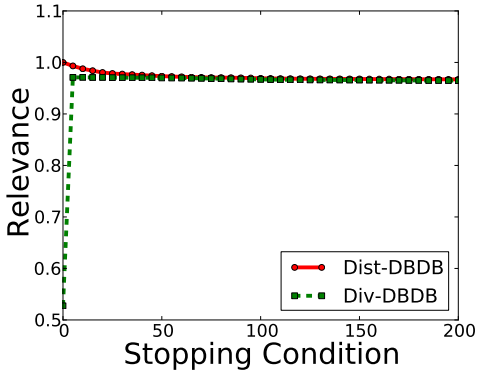
(b) 5 Locations - Gowalla



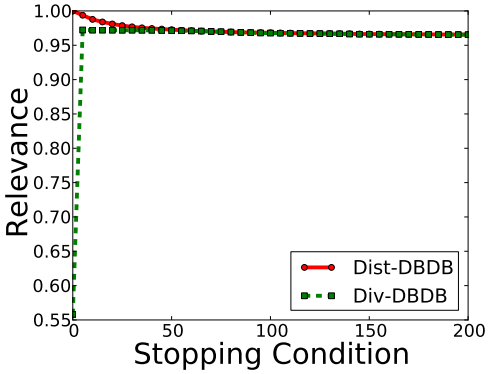
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

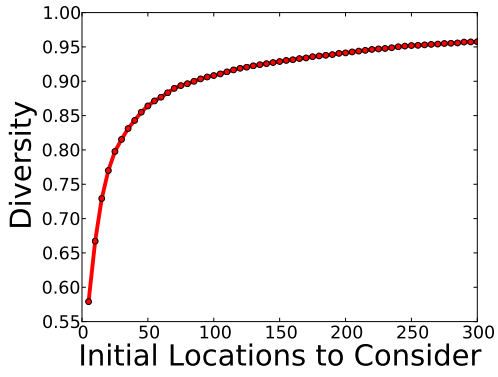


(e) 20 Locations - Foursquare

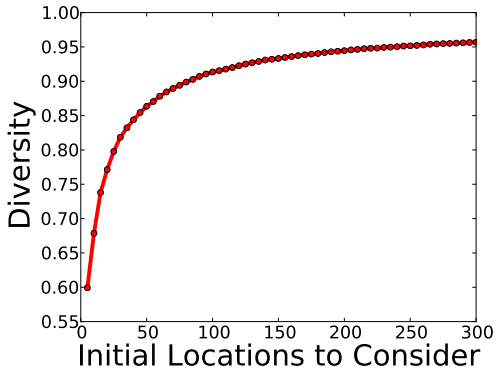


(f) 20 Locations - Gowalla

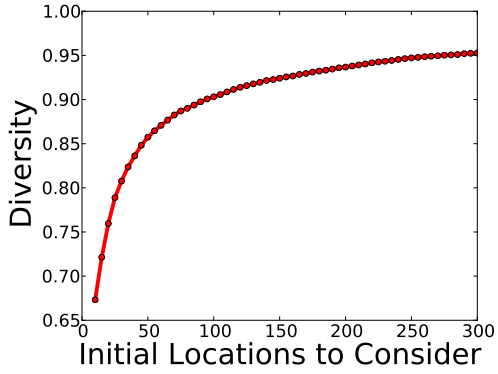
Figure 7.16: Relevance score for changing stop condition



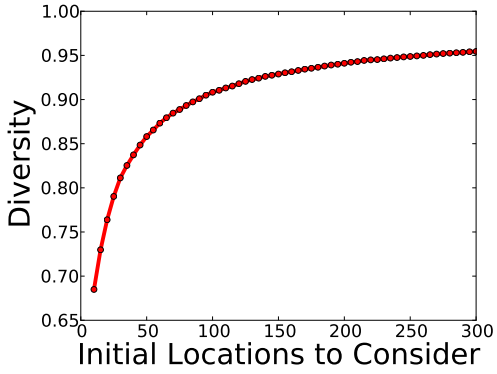
(a) 5 Locations - Foursquare



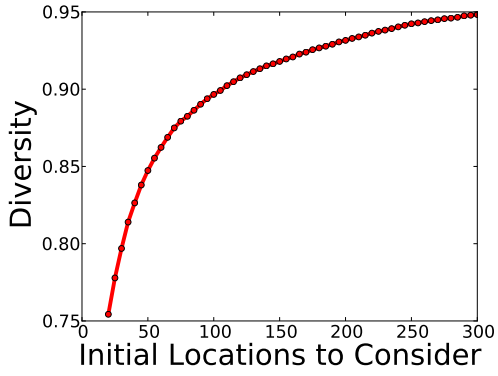
(b) 5 Locations - Gowalla



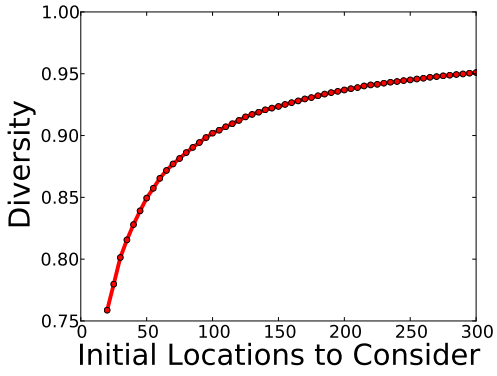
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla

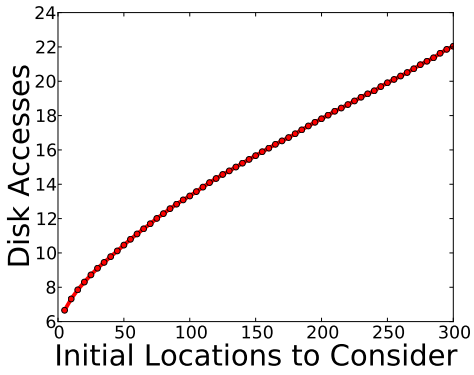


(e) 20 Locations - Foursquare

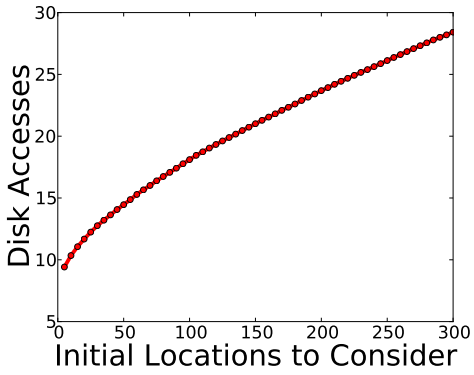


(f) 20 Locations - Gowalla

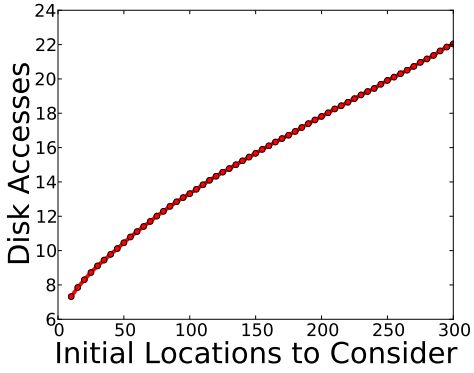
Figure 7.17: Diversity score for changing the number of initial locations to consider for Div-DBDB



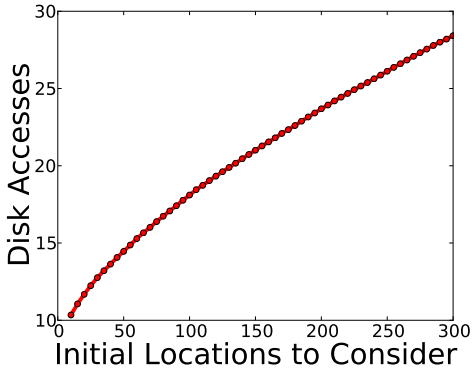
(a) 5 Locations - Foursquare



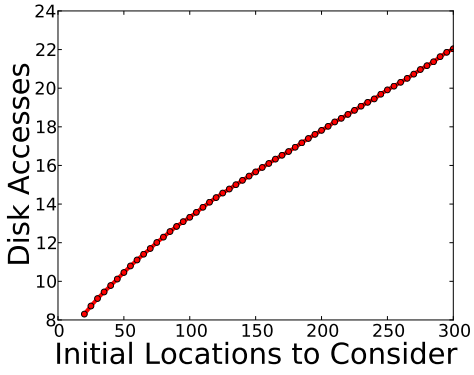
(b) 5 Locations - Gowalla



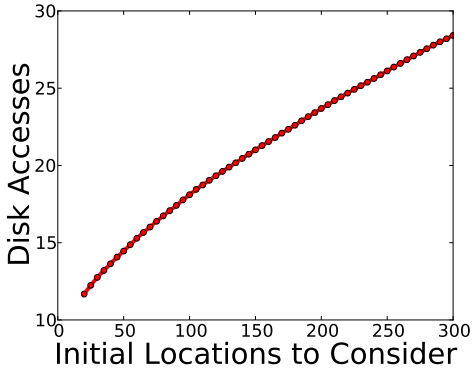
(c) 10 Locations - Foursquare



(d) 10 Locations - Gowalla



(e) 20 Locations - Foursquare



(f) 20 Locations - Gowalla

Figure 7.18: Disk accesses for changing the number of initial locations to consider for Div-DBDB

7.2.4 Evaluation

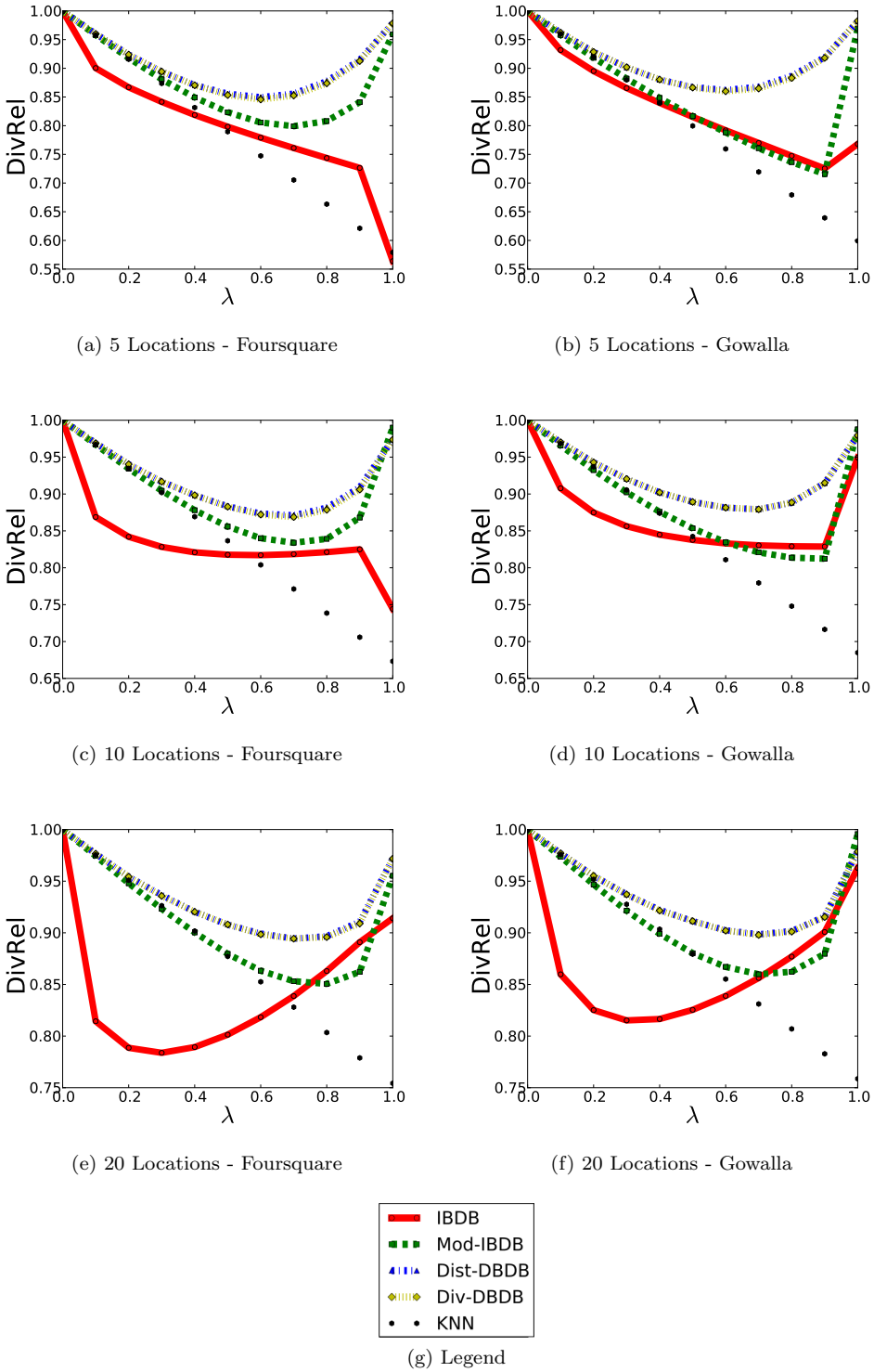
First, we explore the performance and efficiency of the k-nearest diverse neighbor algorithms with the Foursquare and Gowalla datasets followed by using synthetically-generated datasets.

7.2.4.1 LBSN Datasets

Using the optimal values of parameters, we compare the performance and efficiency of the different approaches (i.e., IBDB, Mod-IBDB, Dist-DBDB, Div-DBDB, and KNN) for the k-nearest diverse neighbor problem. The results are shown in Figures 7.19, 7.20, 7.21 and 7.22 for diversity-relevance, disk accesses, diversity, and relevance, respectively. With the exception of $\lambda = 1.0$, Dist-DBDB and Div-DBDB have a larger or equal value for the DivRel metric than the other algorithms, which shows the superiority of these methods. Since all methods return the k-nearest neighbors when $\lambda = 0.0$, they are both equal. For smaller λ (proximity to query point is more important), we see that IBDB performs worse than all other methods, with the performance degrading with more locations recommended. Since the IBDB method will prune locations nearby to already chosen locations, the pruned locations would most likely be better choices since they are closer to the query point and λ is smaller. When diversity becomes more important, KNN starts performing poorly since it does not consider diversity. In addition, Dist-DBDB and Div-DBDB perform the best, with Mod-IBDB consistently performing well and IBDB rarely performing better than Mod-IBDB. Overall, IBDB never performs better than Dist-DBDB and Div-DBDB and only sometimes performs better than Mod-IBDB when λ is larger. In general, Dist-DBDB and Div-DBDB show superior performance with the DivDist metric.

When comparing the diversity and relevance metrics, we see that for smaller λ, IBDB has the largest diversity but the smallest relevance score. Since relevance is the more important metric for smaller λ, this causes IBDB to not perform well. This problem can occur because of it pruning. As λ increases, the diversity results tend to increase linearly with the DBDB methods while IBDB does not increase greatly. For increasing λ for the diversity metric, IBDB tends to increase at a lesser rate for most λ. Comparing the two metrics, the relevance metric tends to decrease exponentially while the diversity metric tends to increase linearly, which means that results with medium-size λ can still obtain great relevance results while obtaining diverse results.

For efficiency, all algorithms reduce to the distance browsing algorithm for the k-nearest neighbor problem for $\lambda = 0$, so they have the same number of average disk accesses. When λ is smaller, the algorithms have similar number of disk accesses, with Div-DBDB occasionally having larger disk accesses due to the efficiency of choosing the initial set of locations. When λ becomes larger, Dist-DBDB and Div-DBDB tend to have a larger number of disk accesses because a larger stopping condition is needed to support choosing a better solution, though the increase is not prohibitively large. Lastly, when $\lambda = 1.0$, all algorithms typically become less efficient (with the exception of KNN) since only the diversity aspect is considered and the R-Tree index stores location based on proximity. In some cases, IBDB and Mod-IBDB performs very poorly, with Mod-IBDB's efficiency being prohibitively large.



(a) 5 Locations - Foursquare

(b) 5 Locations - Gowalla

(c) 10 Locations - Foursquare

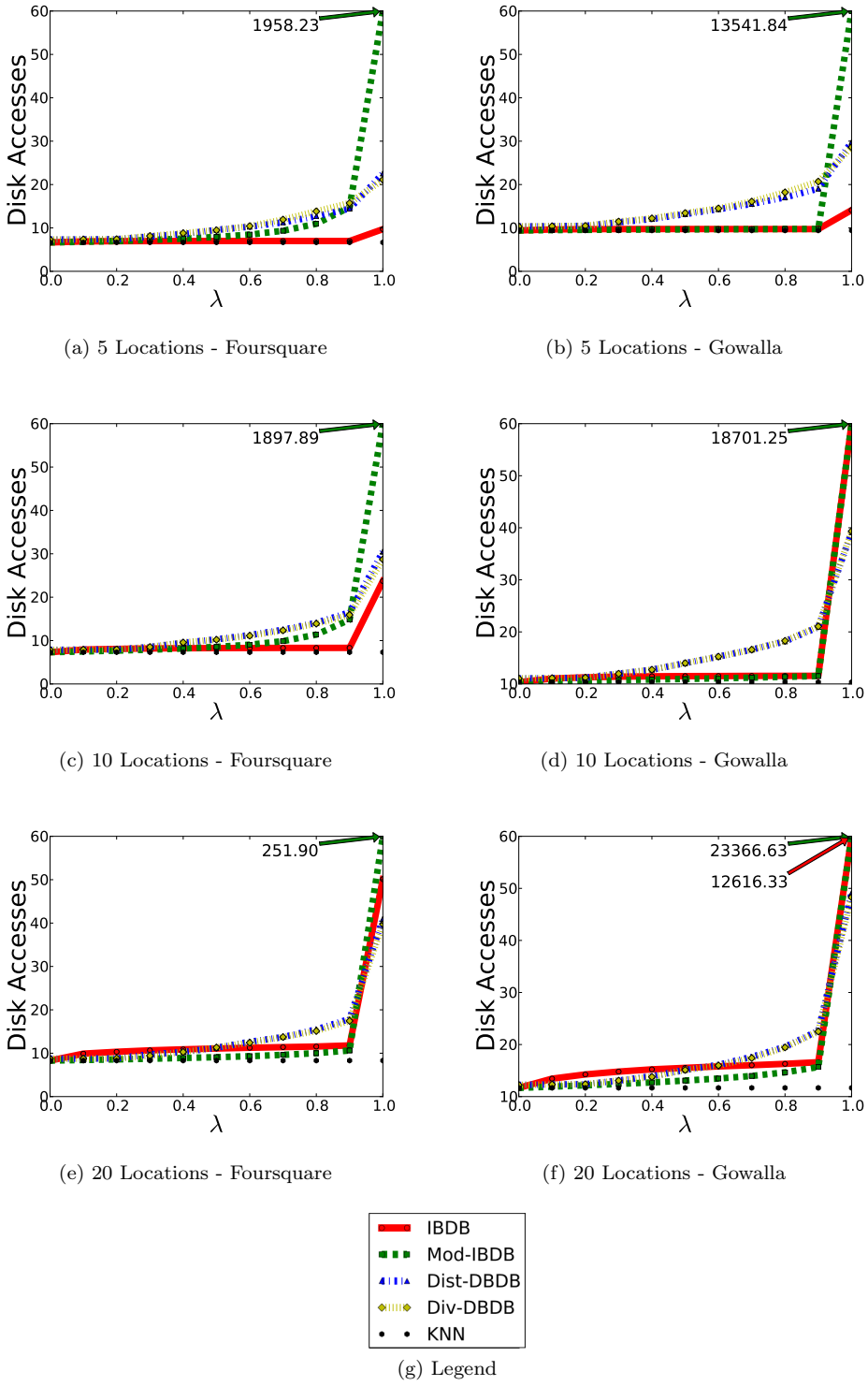
(d) 10 Locations - Gowalla

(e) 20 Locations - Foursquare

(f) 20 Locations - Gowalla

(g) Legend

Figure 7.19: DivRel score for changing λ



(a) 5 Locations - Foursquare

(b) 5 Locations - Gowalla

(c) 10 Locations - Foursquare

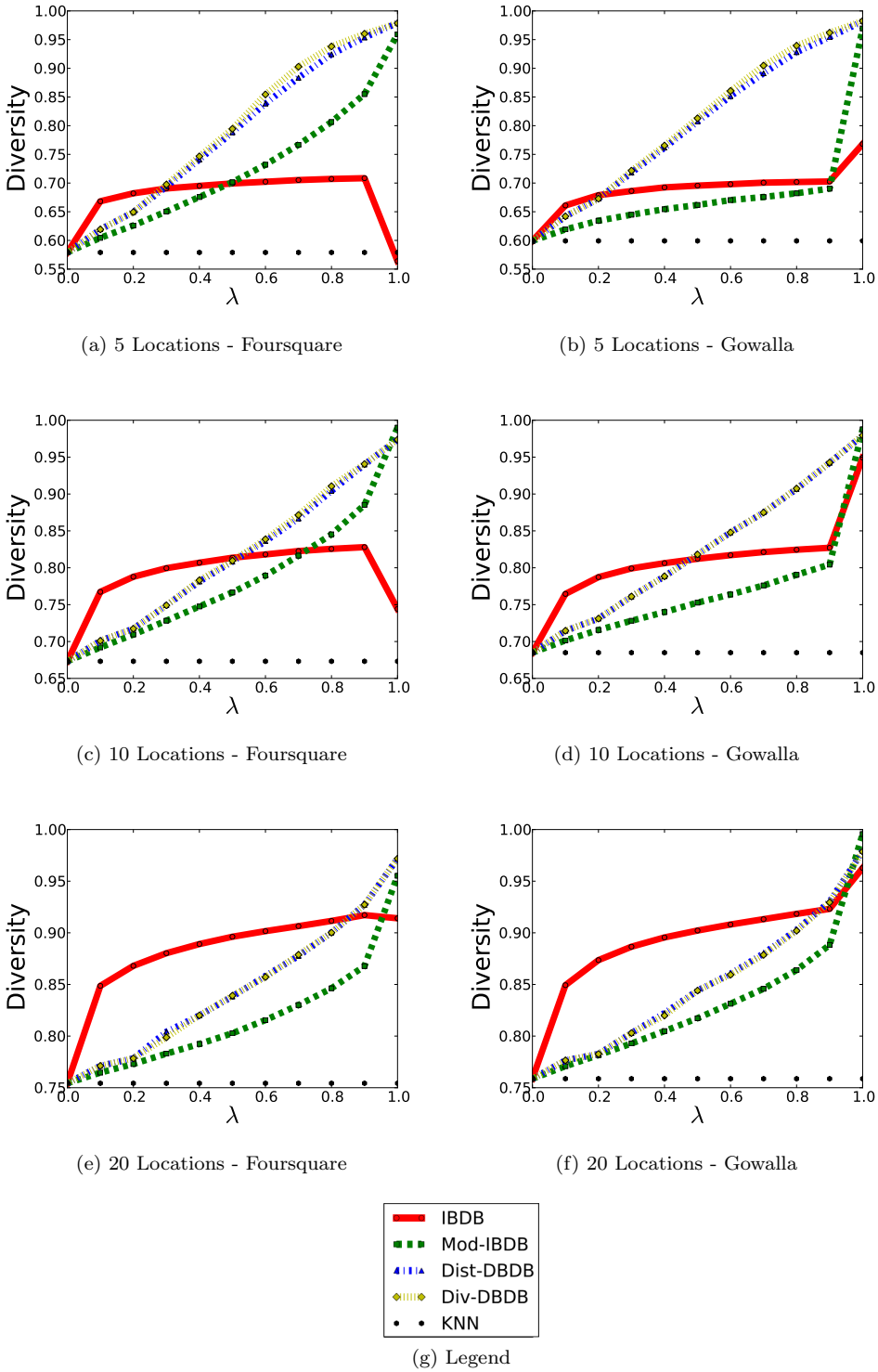
(d) 10 Locations - Gowalla

(e) 20 Locations - Foursquare

(f) 20 Locations - Gowalla

(g) Legend

Figure 7.20: Disk accesses for changing λ



(a) 5 Locations - Foursquare

(b) 5 Locations - Gowalla

(c) 10 Locations - Foursquare

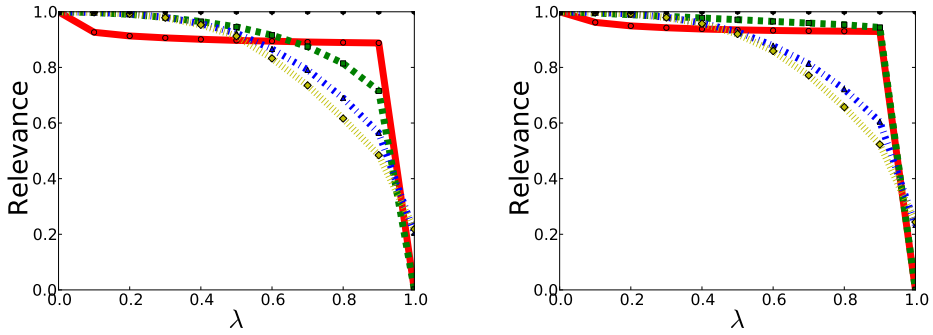
(d) 10 Locations - Gowalla

(e) 20 Locations - Foursquare

(f) 20 Locations - Gowalla

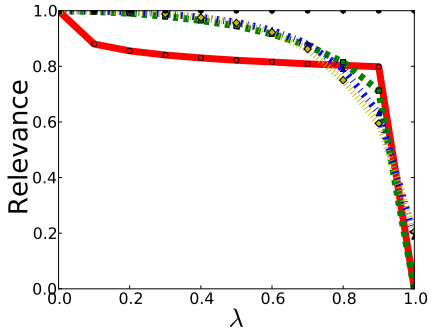
(g) Legend

Figure 7.21: Diversity score for changing λ

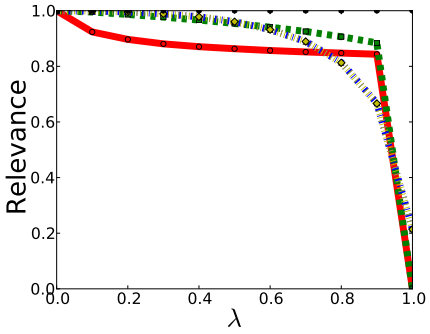


(a) 5 Locations - Foursquare

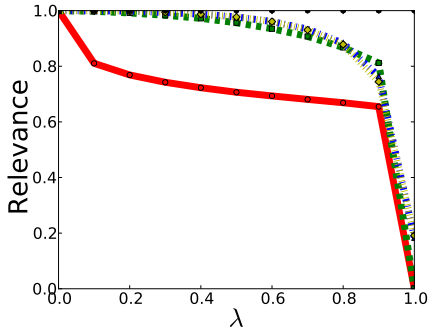
(b) 5 Locations - Gowalla



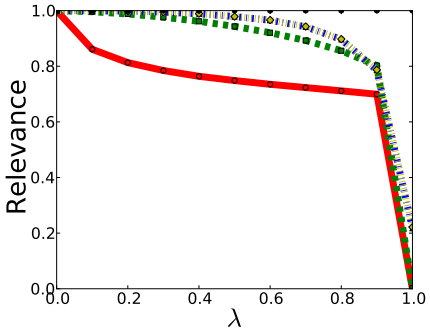
(c) 10 Locations - Foursquare



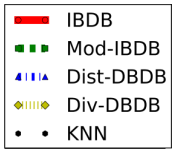
(d) 10 Locations - Gowalla



(e) 20 Locations - Foursquare



(f) 20 Locations - Gowalla



(g) Legend

Figure 7.22: Relevance score for changing λ

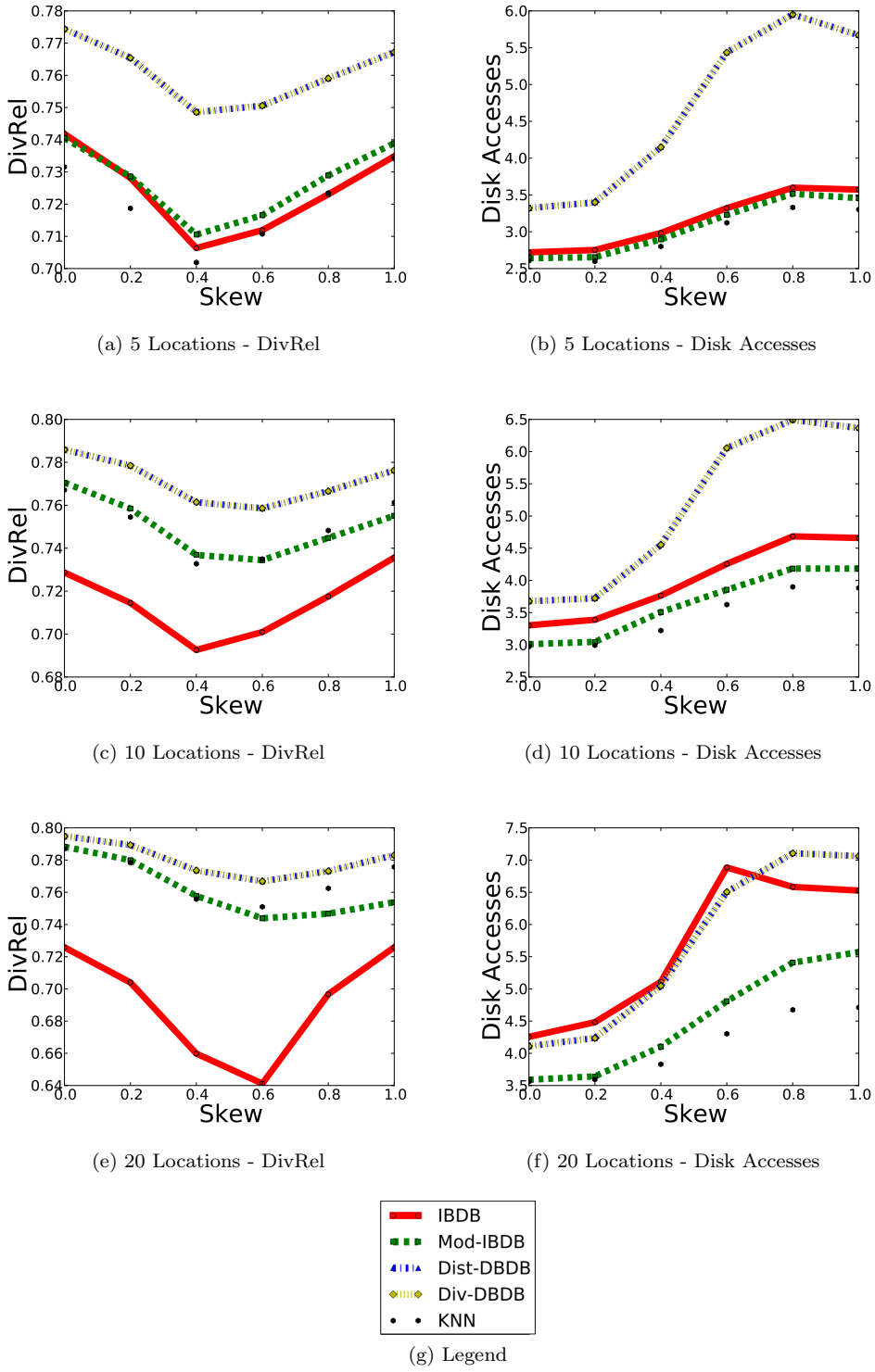
7.2.4.2 Synthetic Datasets

In addition, we compare the k-nearest diverse neighbor algorithms using synthetic datasets based from the Zipf distribution [76]. With this distribution, we can control the skewness of the locations in the dataset with a control parameter, such that 0 represents a uniform distribution while 1 represents high skewness of the data. We create ten datasets each for skewness parameters $0.0, 0.2, 0.4, 0.6, 0.8, 1.0$ to test how the k-nearest diverse neighbor algorithms perform for different types of data. Using the same process for deciding parameters as with the LBSN datasets, we evaluate the different algorithms choosing a uniform λ ($\lambda = 0.0, 1.0$ is considered outliers for considering diversity and relevance, so they are not used in the synthetic experiments) for each k-nearest diverse neighbor query and average the results for each dataset with equal skewness.

Figure 7.23 shows the results for DivRel and disk accesses using synthetic datasets for choosing 5, 10, and 20 locations. In all experiments, Dist-DBDB and Div-DBDB outperform all algorithms for DivRel for all skewness. The difference between the DBDB algorithms and the other algorithms is largest when the number of locations is smaller. Since we choose the stopping condition for the DBDB algorithms to achieve high performance, both algorithms converge to similar results. IBDB performs poorly and shows inferior performance in most cases in comparison to all other algorithms, including the baseline KNN algorithm. Since IBDB performed worse than KNN for smaller λ, the result for the synthetic dataset is logical. In addition, Mod-IBDB always outperforms IBDB which shows that the modifications improved the algorithms, and its results typically outperform the KNN algorithm, with a few exceptions for a higher skew value.

For disk accesses, the order from most efficient to least efficient is KNN, Mod-IBDB, IBDB, Dist-DBDB, Div-DBDB, with the only exception being the mostly equivalent disk access of IBDB and DBDB methods for 20 locations. Even though DBDB shows the worst efficiency, its disk accesses on average are within three of the most efficient algorithm, which shows reasonable efficiency. Mod-IBDB is always more efficient than IBDB, which shows its superiority to IBDB in terms of performance and efficiency. In all algorithms, the increase in skewness means an increase in the number of disk accesses. When the locations are uniformly distributed along a two-dimensional map, the k-nearest neighbors of a query point tend to be spread around the query point. Therefore, quality candidates for the k-nearest diverse neighbors are nearby the query point, which allows for the algorithms to obtain results without incurring many disk accesses. When the locations are more skewed, the k-nearest neighbors tend to be in the same area, which means that the locations are not diverse. Thus, it takes more extensive searching of the R-Tree to find k locations that are nearby as well as diverse.

Overall, the DBDB algorithms show superior performance with the DivRel metric while achieving reasonable efficiency.



(a) 5 Locations - DivRel

(b) 5 Locations - Disk Accesses

(c) 10 Locations - DivRel

(d) 10 Locations - Disk Accesses

(e) 20 Locations - DivRel

(f) 20 Locations - Disk Accesses

(g) Legend

Figure 7.23: DivRel and disk access for changing skew of synthetic dataset

Chapter 8

Conclusions and Future Work

In this thesis, we investigate location recommendation services for mobile users in LBSNs. The goal is to provide recommendation results that are interesting for the user (effectiveness), close in proximity to the user's current location, and spatially diverse in relation to the user's current location.

First, we introduce a location recommendation service that considers both in-town and out-of-town scenarios for increasing effectiveness of the service. We propose a new collaborative recommendation framework, namely, *User Preference, Proximity and Social-Based Collaborative Filtering* (UPS-CF), which incorporates user preference, social connections, and geographical proximity to give location recommendations for mobile users. Using datasets from Foursquare and Gowalla, we conduct extensive experiments to evaluate our proposal and compare with collaborative filtering variants as well as baseline algorithms. We show that UPS-CF outperforms all other comparing algorithms and the effectiveness does not degrade for out-of-town users. In addition, we find that for in-town users, similar users are important while social friends become more important for out-of-town users. Knowing this, our proposed solution can provide location recommendations for users whether they are in town or out of town by adjusting a weight parameter between similar users and friends.

Next, we explore the k-nearest diverse neighbor problem to recommend results that are both close in proximity to the user's current location as well as spatially diverse. We introduce the Modified Index-Based Diverse Browsing framework (Mod-IBDB), which fixes deficiencies that exist in the Index-Based Diverse Browsing framework. In addition, we propose the Distance-Based Diverse Browsing framework (DBDB), which uses the intuition of search algorithms to improve an initial solution by considering swapping locations into the solution in increasing order of distance. For the initial set of locations, we create the Distance-First Distance-Based Diverse Browsing framework that chooses the closest locations as the initial solution for DBDB and the Diversity-First Distance-Based Diverse Browsing framework that uses a heuristic to select a diverse set of locations. Through extensive experimentation, we show that Mod-IBDB

performance increases when proximity to the query point is emphasized more than spatial diversity. In addition, we show that the DBDB algorithms outperform the other state-of-the-art algorithms in terms of performance.

For future work, we plan to investigate the efficiency problems for online location recommendation algorithms detailed in this paper. Also, we could enhance the proposed recommendation techniques by using location semantic tags (e.g., "restaurant" or "museum"). Since this information about locations gives insight into the types of locations a user visits, we could use this information to facilitate location recommendations and provide additional diversity in location recommendation results. Lastly, we could investigate different techniques for choosing the initial diverse solution for the Diversity-First Distance-Based Diverse Browsing framework.

Bibliography

[1] Diversity. `http://www.merriam-webster.com/dictionary/diversity`, Jan. 2013.

[2] P. Adamopoulos and A. Tuzhilin. On unexpectedness in recommender systems: Or how to expect the unexpected. In *Proceedings of Workshop on Novelty and Diversity in Recommender Systems*, RecSys '11, pages 11–18. ACM, 2011.

[3] G. Adomavicius and Y. Kwon. Maximizing aggregate recommendation diversity: A graph-theoretic approach. In *Proceedings of Workshop on Novelty and Diversity in Recommender Systems*, RecSys '11, pages 3–10. ACM, 2011.

[4] G. Adomavicius and Y. Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *Knowledge and Data Engineering, IEEE Transactions on*, 24(5):896–911, 2012.

[5] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[6] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 5–14, New York, NY, USA, 2009. ACM.

[7] A. Angel and N. Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 781–792, New York, NY, USA, 2011. ACM.

[8] J. Bao, Y. Zheng, and M. F. Mokbel. Recommendations in location-based social networks: A survey. *ACM*.

[9] A. L. Barabsi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A*, 311(cond-mat/0104162):3–4. 14 p, Apr 2001.

[10] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD '90, pages 322–331, New York, NY, USA, 1990. ACM.

[11] B. Berjani and T. Strufe. A recommendation system for spots in location-based online social networks. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11, pages 4:1–4:6, New York, NY, USA, 2011. ACM.

[12] K. Bradley and B. Smyth. Improving recommendation diversity. pages 85–94, 2001.

[13] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient diversification of web search results. *Proc. VLDB Endow.*, 4(7):451–459, April 2011.

[14] Z. Chen and T. Li. Addressing diverse user preferences in sql-query-result navigation. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 641–652, New York, NY, USA, 2007. ACM.

[15] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090, New York, NY, USA, 2011. ACM.

[16] C.-Y. Chow, J. Bao, and M. Mokbel. Towards location-based social networking services. In *The 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, 2010.

[17] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 659–666, New York, NY, USA, 2008. ACM.

[18] P. Clough, M. Sanderson, M. Abouammoh, S. Navarro, and M. Paramita. Multiple approaches to analysing query diversity. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 734–735, New York, NY, USA, 2009. ACM.

[19] N. Eagle, A. S. Pentland, and D. Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences of the United States of America*, 106:15274–15278, 2009.

[20] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 421–432, New York, NY, USA, 2012. ACM.

[21] J. Golbeck and D. Hansen. A framework for recommending collections. In *Proceedings of Workshop on Novelty and Diversity in Recommender Systems*, RecSys '11, pages 35–42. ACM, 2011.

[22] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 381–390, New York, NY, USA, 2009. ACM.

[23] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, Boston, Massachusetts, June 18-21 1984.

[24] J. R. Haritsa. The kndn problem: A quest for unity in diversity. *IEEE Data Eng. Bull.*, 32(4):15–22, 2009.

[25] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, June 1999.

[26] T. Horozov, N. Narasimhan, and V. Vasudevan. Using location for personalized poi recommendations in mobile environments. In *SAINT*, pages 124–129, 2006.

[27] R. Hu and P. Pu. Helping users perceive recommendation diversity. In *Proceedings of Workshop on Novelty and Diversity in Recommender Systems*, RecSys '11, pages 43–50. ACM, 2011.

[28] A. Jain, P. Sarda, and J. R. Haritsa. Providing diversity in k-nearest neighbor query results. *CoRR*, cs.DB/0310028, 2003.

[29] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, WebKDD/SNA-KDD '07, 2007.

[30] I. Kamel and C. Faloutsos. On packing r-trees. In *Proceedings of the second international conference on Information and knowledge management*, CIKM '93, pages 490–499, New York, NY, USA, 1993. ACM.

[31] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Boston, Massachusetts, July 19-23 2009.

[32] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[33] G. Kossinets and D. Watts. Empirical analysis of an evolving social network. *Science*, 311(5757):88–90, 2006.

[34] O. Kucuktunc and H. Ferhatosmanoglu. Diverse browsing for spatial data. Technical report, Ohio State University, February 2011.

[35] O. Kucuktunc and H. Ferhatosmanoglu. Lambda-diverse nearest neighbors browsing for multi-dimensional data. *IEEE Transactions on Knowledge and Data Engineering*, PP(99), 2011.

[36] K. C. K. Lee, W.-C. Lee, and H. V. Leong. Nearest surrounder queries. *IEEE Trans. Knowl. Data Eng.*, 22(10):1444–1458, 2010.

[37] S. Leutenegger, M. Lopez, and J. Edgington. Str: a simple and efficient algorithm for r-tree packing. In *Data Engineering, 1997. Proceedings. 13th International Conference on*, pages 497–506, Apr. 1997.

[38] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system. In *Proceedings of the ICDE conference*. ACM, 2012.

[39] N. Li and G. Chen. Analysis of a location-based social network. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04*, pages 263–270, Washington, DC, USA, 2009. IEEE Computer Society.

[40] B. Liu and H. V. Jagadish. Using trees to depict a forest. *Proc. VLDB Endow.*, 2(1):133–144, August 2009.

[41] P. J. Ludford, R. Priedhorsky, K. Reily, and L. Terveen. Capturing, sharing, and using local place information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1235–1244, New York, NY, USA, 2007. ACM.

[42] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27:415–444, August 2001.

[43] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[44] K. Oku and F. Hattori. Fusion-based recommender system for improving serendipity. In *Proceedings of Workshop on Novelty and Diversity in Recommender Systems*, RecSys '11, pages 19–26. ACM, 2011.

[45] M. L. Paramita, J. Tang, and M. Sanderson. Generic and spatial approaches to image search results diversification. In *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval*, ECIR '09, pages 603–610, Berlin, Heidelberg, 2009. Springer-Verlag.

[46] O. Pivert, A. Hadjali, and G. Smits. Searching for a compromise between satisfaction and diversity in database fuzzy querying. In *Proceedings of the 6th Conference of the European Society of Fuzzy Systems and Technology*, EUSFLAT'11, 2011.

[47] F. Radlinski and S. Dumais. Improving personalized web search using result diversification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 691–692, New York, NY, USA, 2006. ACM.

[48] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. *SIGMOD Rec.*, 24(2):71–79, May 1995.

[49] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed r-trees. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, SIGMOD '85, pages 17–31, New York, NY, USA, 1985. ACM.

[50] S. Santini and P. Castells. An evaluation of novelty and diversity based on fuzzy logic. In *Proceedings of Workshop on Novelty and Diversity in Recommender Systems*, RecSys '11, pages 51–58. ACM, 2011.

[51] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong, Hong Kong, May 2001.

[52] S. Scellato and C. Mascolo. Measuring user activity on an online location-based social network. In *Proceedings of Third International Workshop on Network Science for Communication Networks*, April 2011.

[53] S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo. Socio-spatial properties of online location-based social networks. In *ICWSM*, 2011.

[54] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 507–518, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[55] S. Sheth, J. Bell, N. Arora, and G. Kaiser. Towards diversity in recommendations using social networks.

[56] Y. Takeuchi and M. Sugimoto. Cityvoyager: An outdoor recommendation system based on user location history. *Ubiquitous Intelligence and Computing*, pages 625–636, 2006.

[57] J. Tang and M. Sanderson. Evaluation and user preference study on spatial diversity. In *Proceedings of the 32nd European conference on Advances in Information Retrieval*, ECIR'2010, pages 179–190, Berlin, Heidelberg, 2010. Springer-Verlag.

[58] M. Van Kreveld, I. Reinbacher, A. Arampatzis, and R. Van Zwol. Distributed ranking methods for geographic information retrieva. In *Developments in Spatial Data Handling (11th International Symposium on Spatial Data Handling)*, pages 231–243, Berlin, 2004. Springer.

[59] M. Van Kreveld, I. Reinbacher, A. Arampatzis, and R. Van Zwol. Multi-dimensional scattered ranking methods for geographic information retrieval*. *Geoinformatica*, 9(1):61–84, March 2005.

[60] R. H. van Leuken, L. Garcia, X. Olivares, and R. van Zwol. Visual diversification of image search results. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 341–350, New York, NY, USA, 2009. ACM.

[61] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. A. Yahia. Efficient computation of diverse query results. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 228–236, Washington, DC, USA, 2008. IEEE Computer Society.

[62] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. Divdb: A system for diversifying query results. *PVLDB*, pages 1395–1398, 2011.

[63] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras. On query result diversification. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1163–1174, Washington, DC, USA, 2011. IEEE Computer Society.

[64] M. J. Welch, J. Cho, and C. Olston. Search result diversity for informational queries. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 237–246, New York, NY, USA, 2011. ACM.

[65] M. Ye, X. Liu, and W.-C. Lee. Exploring social influence for recommendation - a probabilistic generative model approach. *SIGIR*, August 12-16 2012.

[66] M. Ye, P. Yin, and W.-C. Lee. Location recommendation for location-based social networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, San Jose, California, November 02-05 2010.

[67] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Beijing, China, July 24-28 2011.

[68] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 368–378, New York, NY, USA, 2009. ACM.

[69] Q. Yuan, S. Zhao, L. Chen, S. Ding, X. Zhang, and W. Zheng. Augmenting collaborative recommender by fusing explicit social relationships. In *ACM RecSys-Workshop*, pages 49–56, 2009.

[70] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 123–130, New York, NY, USA, 2008. ACM.

[71] M. Zhang and N. Hurley. Statistical modeling of diversity in top-n recommender systems. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 1:490–497, 2009.

[72] F. Zhao, X. Zhang, A. K. H. Tung, and G. Chen. Broad: Diversified keyword search in databases. *PVLDB*, pages 1355–1358, 2011.

[73] V. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang. Collaborative filtering meets mobile recommendation. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 236–241, 2010.

[74] V. Zheng, Y. Zheng, X. Xie, and Q. Yang. Collaborative location and activity recommendations with gps history data. In *Proceedings of the 19th International Conference on World Wide Web*, pages 1029–1038, 2010.

[75] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM.

[76] G. Zipf. Human behaviour and the principle of least-effort. Addison-Wesley, Cambridge, MA, 1949.

ACADEMIC VITA: GREGORY DAVID FERENCE

205 Greenbriar Drive • Cranberry Township, Pennsylvania 16066 • (724) 316-6413 • gdference@gmail.com

EDUCATION

The Pennsylvania State University — AUGUST 2008 – SPRING 2013
Schreyer Honors College — *University Park, PA*

Master of Science in Computer Science and Engineering
Bachelor of Science in Computer Science (with honors)
Bachelor of Science in Mathematics

RESEARCH

GRADUATE RESEARCHER — SPRING 2010 – SPRING 2013
The Pennsylvania State University — *University Park, PA*

- Developed next-generation location recommendation services by incorporating social networks, spatial databases, data mining, information retrieval, and mobile computing techniques

RELEVANT EMPLOYMENT

SOFTWARE ENGINEERING INTERN — SUMMER 2012
Google Inc. — *Pittsburgh, PA*

- Implemented components of client/server software and an Android application using Java
- Improved usability of Android application by changing user interface of preference pages
- Designed and implemented web pages using Google Web Toolkit

SOFTWARE ENGINEERING INTERN — SUMMER 2011
Cisco Systems, Inc. — *Research Triangle Park, NC*

- Created software that collects and displays performance statistic information for networks
- Developed features using Java, JavaScript and JSP in a Linux environment
- Enhanced user interface by replacing and modifying new graphing package
- Created reports that polled data from devices and displayed data in graphs, tables and CSV files

SOFTWARE ENGINEERING INTERN — SUMMER 2010
Westinghouse Electric Company — *Cranberry Township, PA*

- Programmed software for the safety system of a nuclear power plant using the C programming language
- Built shell scripts to control multiple software applications on a UNIX platform
- Designed and implemented custom screens to allow users to retrieve safety-related information
- Collaborated with coworkers to design and develop large, technically complex software

SOFTWARE ENGINEERING INTERN — SUMMER 2009
Compunetix, Inc. — *Monroeville, PA*

- Performed system analysis and software development with video conferencing software
- Developed software in Java using techniques such as multithreading and hash tables
- Accomplished many steps in the software engineering lifecycle, from requirement analysis to implementing, testing and documenting

SCHOLARSHIPS AND AWARDS

- Schreyer Honors College Academic Excellence Scholarship
- George and Terry Selembo Trustee Scholarship in the Schreyer Honors College
- Chris Mader Memorial Scholarship, Penn State Department of Computer Science and Engineering
- Evan Johnson Memorial Award, Penn State Department of Mathematics
- O. Richard Bundy, Jr. Blue Band Endowed Scholarship

- Ruth Varner Otto Memorial Scholarship
- Evan Pugh Scholar Award (Senior)
- Evan Pugh Scholar Award (Junior)

VOLUNTEER AND EXTRACURRICULAR ACTIVITIES

- Association for Computing Machinery (2009 – 2012)
- Robotics Club (2009 – 2010)
- Marching Blue Band (2008 – 2012)
  - Guide (2011 – 2012)
  - Squad Leader (2010 – 2012)
- Pride of the Lions Basketball Pep Band (2009 – 2011)
- Concert Band (2009 – 2013)
- Penn State IFC/Panhellenic Dance Marathon (2009 – 2010)