Location Analytics for Location-Based Social Networks

Saleem, Muhammad Aamir

DOI (link to publication from Publisher):
10.5278/vbn.phd.tech.00038

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Saleem, M. A. (2018). *Location Analytics for Location-Based Social Networks*. Aalborg Universitetsforlag. https://doi.org/10.5278/vbn.phd.tech.00038

LOCATION ANALYTICS FOR LOCATION-BASED SOCIAL NETWORKS

BY
MUHAMMAD AAMIR SALEEM

DISSERTATION SUBMITTED 2018

AALBORG UNIVERSITY
DENMARK

Location Analytics for Location-Based Social Networks

Ph.D. Dissertation

Muhammad Aamir Saleem

Dissertation submitted April, 2018

A thesis submitted to the Technical Faculty of IT and Design at Aalborg University (AAU) and the Faculty of Engineering at Université Libre De Bruxelles (ULB), in partial fulfillment for the joint Ph.D. degree in computer science. The thesis is not submitted to any other organization at the same time.

Dissertation submitted:	April, 2018
PhD Supervisor (AAU):	Prof. Torben Bach Pedersen Aalborg University, Denmark
PhD Supervisor (ULB):	Prof. Toon Calders Université Libre De Bruxelles, Belgium
Assistant PhD supervisors:	Assoc. Prof. Panagiotis Karras Aalborg University, Denmark Asst. Prof. Xike Xike Aalborg University, Denmark
PhD committee:	Associate Professor Bin Yang (chairman) Aalborg University, Denmark Professor Kjetil Nørvåg NTNU Trondheim, Norway Research Director Franseco Bonchi ISI Foundation, Italy
PhD Series:	Technical Faculty of IT and Design, Aalborg University
Department:	Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-185-9

Published by:
Aalborg University Press
Langagervej 2
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2018

Abstract

The popularity of location empowered devices such as GPS enabled smartphones has immensely amplified the use of location-based services in social networks. This happened by allowing users to share Geo-tagged contents such as current locations/check-ins with their social network friends. These location-aware social networks are called Location-based Social Networks (LBSN), and examples include Foursquare and Gowalla. The data of LBSNs are being used for providing different kinds of services such as the recommendation of locations, friends, activities, and media contents, and the prediction of user's locations. To provide such services, different queries are utilized that exploit activity/check-in data of users. Usually, LBSN data is divided into two parts, a social graph that encapsulates the friendships of users and an activity graph that maintains the visit history of users at locations. Such a data separation is scalable enough for processing queries that directly utilize friendship information and visit history of users. These queries are called user and activity analytic queries. The visits of users at locations create relationships between those locations. Such relationships can be built on different features such as common visitors, geographical distance, and mutual location categories between them. The process of analysing such relationships for optimizing location-based services is termed *Location Analytics*. In location analytics, we expose the subjective nature of locations that can further be used for applications in the domain of prediction of visitors, traffic management, route planning, and targeted marketing.

In this thesis, we provide a general LBSN data model which can support storage and processing of queries required for different applications, called location analytics queries. The LBSN data model we introduce, segregates the LBSN data into three graphs: the social graph, the activity graph, and the location graph. The location graph maintains the interactions of locations among each other. We define primitive queries for each of these graphs. In order to process an advanced query, we express it as a combination of these primitive queries and process them on corresponding graphs in parallel. We further provide a distributed data processing framework called GeoSocial-GraphX (GSG). GSG implements the aforementioned LBSN data model for efficient and scalable processing of the queries. We further exploit the location graph for providing novel location analytics queries in the domain of influence maximization and visitor prediction. We introduce a notion of location influence. Such influence can capture the interactions of locations based on their visitors and can be used for propagation of information between them. The applications of such a query lie in the domain of outdoor marketing, and simulation of virus and news propagation. We also provide a unified system IMaxer that can evaluate and compare different information propagation mechanisms. We further exploit the subjective nature of locations by analysing the mobility behaviour of their visitors. We use such information to predict the individual visitors as well as the groups of visitors (cohorts) in future for those locations. The prediction of visitors can be used for better event planning, traffic management, targeted marketing, and ride-sharing services.

In order to evaluate the proposed frameworks and approaches, we utilize data from four real-life LBSNs: Foursquare, Brightkite, Gowalla, and Wee Places. The detailed LBSN data mining and statistically significant experimental evaluation results show the effectiveness, efficiency, and scalability of our proposed methods. Our proposed approaches can be employed in real systems for providing life-care services.

Resumé

Populariteten af apparater med indbygget lokationsteknologi som f.eks. GPS i smartphones har forøget brugen af lokationsbaserede tjenester i sociale netværk. Det kommer som en konsekvens af at brugere kan dele geo-tagged indhold såsom nuværende lokation med deres sociale netværk, såkaldte Location-based Social Networks (LSBN). Foursquare og Gowalla er eksempler på to LSBNs. LSBN data danner grundlag for en række forskellige tjenester, som f.eks. anbefalinger af steder, venner, aktiviteter og medier. LSBN data er almindeligvis delt op i en venskabsgraf der beskriver venskab mellem brugere i et socialt netværk og en aktivitetsgraf der beskriver brugernes besøgshistorik ved diverse lokationer. En sådan dataseparation tillader at processere forespørgsler der inkorporerer information om venskaber og besøgshistorik direkte i forespørgslen på selv store datasæt. Brugeres besøgshistorik kan bruges til at skabe relationer mellem lokationer, f.eks. baseret på fælles besøgende, geografisk afstand, og fælles lokationskategori. En analyse af sådanne relationer med henblik på at optimere lokationsbaserede tjenester kaldes en lokationsanalyse. En lokationsanalyse betragter verden fra et lokationsperspektiv fremfor et brugerperspektiv og kan bruges til bl.a. at forudsige besøgende, trafikstyring, ruteplanlægning og målrettet marketing.

I denne afhandling præsenterer vi en generel LSBN datamodel der understøtter lagring og processering af forespørgsler til lokationsanalyse. Datamodellen separerer LSBN data i en venskabsgraf, en aktivitetsgraf og en lokationsgraf. Lokationsgrafen modellerer lokationers indbyrdes interaktioner. Vi definerer simple forespørgsler for hver af disse grafer. Avancerede forespørgsler defineres som en kombination af de simple forespørgsler, og de simple forespørgsler processeres i parallel i deres tilhørende grafer. Vi præsenterer GeoSocialGraphX (GSG), en distribueret model til processering af avancerede forespørgsler. GSG bruger den 3-delte datamodel til at processere forespørgsler effektivt og skalérbart. Yderligere udnytter vi lokationsgrafen til at udføre nye former for forespørgsler til influence maximization i sociale netværk og forudsigelse af besøgende. Vi introducerer begrebet lokationsindflydelse til at beskrive interaktionerne mellem lokationer baseret på deres besøgende og kan bruges til at propagere information mellem noder i lokationsgrafen. Lokationsindflydelse kan f.eks. bruges til outdoor marketing, samt simulering af vira- og nyhedsspredning. Ydermere præsenterer vi IMaxer, et samlet system der kan evaluere og sammenligne forskellige spredningsmekanismer. Vi analyserer også brugeres bevægelsesmønstre, og bruger informationen til både at forudsige de enkelte besøgende og grupper af besøgende. Forudsigelse af besøgende kan forbedre trafikstyring, målrettet marketing, ride-sharing tjenester og planlægning af begivenheder.

Vi evaluerer vores tilgange og modeller eksperimentelt ved at benytte data fra fire LSBN: Foursquare, Brightkite, Gowalla og Wee Places. Vi demonstrerer med statistisk signifikans at de foreslåede metoder er effektive, skalérbare og producerer resultater af høj kvalitet. Vores foreslåede metoder kan forbedre kvaliteten af mange forskellige tjenester.

Acknowledgements

My first and foremost gratitude is for Allah Almighty Who gave me patience and strength to complete this thesis. Without His guidance and help, I would never have been able to reach this point.

I would like to express my sincere gratitude to my Ph.D. advisor, Professor Torben Bach Pedersen. He has been always been a great help for me throughout my Ph.D. journey. I must have to express that his guidance, patience, and advice contributed a significant role in this thesis. More specifically, I would like to admire his professionalism, outstanding analytical skills of understanding both technical and personal issues of his students, and prompt replies to the queries. His concrete and constructive comments always helped me to take my work in the right direction.

I am grateful to Professor Toon Calders, Associate Professor Panagiotis Karras, and Assistant Professor Xike Xie, who co-supervised my Ph.D. studies. I must admit that I enjoyed each and every moment of working with them. The things I learned from them, not only helped me to bring the thesis in this condition but will also assist me in my future endeavours. I am very thankful to them for their contribution to my Ph.D. studies.

I would like to thank all my colleagues in the Computer Science department at Aalborg University. Whenever needed, they happened to be a huge help for me. I would like to pay my gratitude to Tobias and Simon for helping me in writing the Danish summary. My special thanks to the colleagues who played table football with me. The games and discussions during them always refreshed me and helped me to get back to the work with a new energy. Thanks to Helle Schroll and Helle Westmark for their support regarding the administrative issues. I would also express my thanks to my colleagues at my host university, Université Libre De Bruxelles who helped me to spend a wonderful time there.

Finally, I would like to thank my family for always being with me. Whenever I got disappointed and depressed, they give me the courage, motivation and believe that I can do this. This thesis would never have been same without their continuous help.

Contents

Abstract iii

Dansk Resumé (Summary in Danish) v

Acknowledgements vii

Thesis Details xv

I Thesis Summary 1

1 Introduction . . . 3
- 1.1 Background and Motivation . . . 3
- 1.2 Location Analytics . . . 4

2 General and Scalable LBSN Data Management . . . 6
- 2.1 Motivation and Problem Statement . . . 6
- 2.2 LBSN Data Storage and Processing Model . . . 6
- 2.3 GeoSocial-GraphX . . . 9
- 2.4 Discussion . . . 10

3 Location Influence . . . 11
- 3.1 Motivation and Problem Statement . . . 11
- 3.2 Location-Based Influence . . . 12
- 3.3 Influence Oracle . . . 13
- 3.4 Discussion . . . 14

4 Effective and Efficient Location Influence Mining . . . 15
- 4.1 Motivation and Problem Statement . . . 15
- 4.2 Effective Influence Models . . . 16
- 4.3 Efficient Location Influence Models . . . 17
- 4.4 Discussion . . . 18

5 Unified System for Influence Propagations in LBSNs . . . 19
- 5.1 Motivation and Problem Statement . . . 19
- 5.2 Unified Influence Propagation Model in LBSNs . . . 20
- 5.3 IMaxer . . . 21

6 Prediction of Visitors 23
6.1 Motivation and Problem Statement 23
6.2 Collective Matrix Factorization-based Visitor Prediction Model (CMViP) 23
6.3 Discussion 26
7 Prediction of Geo-Social Cohorts 26
7.1 Motivation and Problem Statement 26
7.2 Cohort Discovery in LBSNs (COVER) 27
7.3 Discussion 28
8 Summary of Contributions 30
9 Future Work Directions 32
References 33

II Papers 35

A Scalable Processing of Location-Based Social Networking Queries 37
1 Introduction 39
2 LBSN Model 40
2.1 Preliminaries 40
2.2 LBSN Graphs 41
3 LBSN Queries 43
3.1 Query Primitives 43
3.2 Advanced LBSN Queries 45
4 GSG Architecture 46
4.1 Storage layer 46
4.2 Operation layer 48
4.3 Index layer 50
4.4 Query Engine 52
5 Experiments 53
5.1 Setup 53
5.2 Results 54
6 Related Work 61
7 Conclusion and Future Work 63
References 63

B Effective and Efficient Location Influence Mining in Location-Based Social Networks 67
1 Introduction 69
2 Related Work 72
3 Location-Based Influence 73
3.1 Bridging Visitors for Location Influence 74

3.2 Methods for determining Location Influence 76
3.3 Combined Location Influence 78
3.4 Problem Formulation 79
4 Influence Oracle 79
4.1 Exact Influence Oracle 80
4.2 Approximate Influence Oracle for M_D and M_F 86
4.3 Single-Influencer based Influence Oracle 87
5 Location Influence Maximization 90
6 LBSN Data Analysis 92
6.1 Setting parameters for the influence models 93
6.2 Mobility analysis of friends 95
7 Evaluation 97
7.1 Qualitative significance of the Location Influence 99
7.2 Comparison of Influence Models 99
7.3 Approximations for M_{DA} and M_{FA} 101
7.4 Effects of parameters: ω and τ 102
7.5 Single-Influencer based Location Influence 104
7.6 Influence Maximization 105
8 Conclusion and Future Work 106
References 107

C IMaxer: A Unified System for Evaluating Influence Maximization in Location-based Social Networks 111
1 Introduction 113
2 IMaxer System Overview 115
3 IMaxer's technical overview 117
3.1 Modeling IM using *Unfied IM Model* 117
3.2 System Implementation 120
4 Demonstration 120
References 121

D Predicting Visitors Using Location-Based Social Networks 123
1 Introduction 125
2 Related Works 127
2.1 Recommendations to Users 127
2.2 Visitors Prediction 128
3 Problem Formulation 128
3.1 Preliminaries 128
3.2 Users' mobility analysis in LBSNs 129
3.3 CMViP 131
3.4 Problem Statement 135
4 Solution Framework 135
4.1 Construction of Matrices 135

4.2 Initialization 135
4.3 Non-Negative Collective Matrix Factorization 136
4.4 Optimization 138
4.5 Prediction of Visitors 139
5 Experimental Evaluation 139
5.1 Datasets 140
5.2 Evaluation Measures 141
5.3 Parameter Analysis 141
5.4 Competitors 142
5.5 Results 142
6 Conclusion 148
References 148

E Mining Geo-Social Cohorts in Location-Based Social Networks 153
1 Introduction 155
2 Related Work 157
2.1 Finding Cliques 157
2.2 Dense Subgraph Discovery 157
2.3 Community Detection 158
3 Problem Formulation 159
3.1 Preliminary Concepts 159
3.2 Objective 160
3.3 Maximizing Activity Density 161
4 COVER Algorithm 163
5 Experimental Study 167
5.1 Datasets 167
5.2 Brute-Force Cohort Discovery 167
5.3 Holdout Approach 168
5.4 Revalidating the Clique Constraint 169
5.5 Prediction without Input Categories 170
5.6 Prediction with Input Categories 172
5.7 Group size vs. surplus α 173
5.8 Scalability 173
5.9 Comparative Evaluation 174
5.10 Analysis on Predicted Cohorts 176
6 Conclusion 176
References 177

F Location Influence in Location-Based Social Networks 181
1 Introduction 183
2 Related Work 185
3 Location-based Influence 186
3.1 Location-based Social Network 186

3.2 Models of Location-based Influence 186
3.3 Friendship-based Location Influence 188
3.4 Combined Location Influence 188
3.5 Problem Formulation 189
4 Solution Framework 189
4.1 Influence Oracle 190
4.2 Approximate Influence Oracle 192
4.3 Influence Maximization 193
5 LBSN Data Analysis 195
5.1 Mobility analysis of friends 195
5.2 Setting ω and τ 196
6 Evaluation 197
6.1 Approximate vs. Exact Oracle 199
6.2 Influence of ω and τ 199
6.3 Influence Maximization 202
6.4 Qualitative Experiment 203
7 Conclusion and Future Work 204
References 204

G Activity-Driven Influence Maximization in Social Networks 207
1 Introduction 209
2 Data-Driven Information Maximization 210
3 Finding Influential Locations 211
4 Conclusion 212
References 212

Thesis Details

Thesis Title:	Location Analytics for Location-Based Social Networks
Ph.D. Student:	Muhammad Aamir Saleem
Supervisors:	Prof. Torben Bach Pedersen, Aalborg University
	Prof. Toon Calders, Universite Libre De Bruxelles

The main body of this thesis consist of the following accepted and submitted papers.

[A] Muhammad Aamir Saleem, Xike Xie, Torben Bach Pedersen, "Scalable Processing of Location-Based Social Networking Queries". *In Proceedings of the IEEE 17th International Conference on Mobile Data Management (MDM 2016), Porto, Portugal, pages 132-141, 2016*

[B] Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Torben Bach Pedersen, "Effective and Efficient Location Influence Mining in Location-Based Social Networks" *Under revision for a Journal publication, submitted on September, 2017*

[C] Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Xike Xie, Torben Bach Pedersen, "IMaxer: A Unified System for Evaluating Influence Maximization in Location-based Social Networks" *In the Proceedings of the ACM Conference on Information and Knowledge Management, CIKM 2017, Singapore, pages 2523-2526, 2017.*

[D] Muhammad Aamir Saleem, Felipe Costa, Peter Dolog, Panagiotis Karras, Toon Calders, Torben Bach Pedersen, "Predicting Visitors Using Location-Based Social Networks" *Under revision for a Conference publication, submitted on January, 2018*

[E] Muhammad Aamir Saleem, Panagiotis Karras, Toon Calders, Torben Bach Pedersen, "Mining Geo-Social Cohorts in Location-Based Social Networks" *Under revision for a Conference publication, submitted on February, 2018*

In addition to the main papers, Paper F and Paper G have also been included. Paper G is an high level abstract version of Paper F and Paper B. Paper F is extended by Paper B, thus it can be considered as a subset of Paper B.

[F] Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Xike Xie, Torben Bach Pedersen, "Location Influence in Location-based Social Networks" *In the Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM 2017), Cambridge, United Kingdom, pages 621-630, 2017*

[G] Rohit Kumar, Muhammad Aamir Saleem, Toon Calders, Xike Xie, Torben Bach Pedersen, "Activity-Driven Influence Maximization in Social Networks" *In the Proceedings of the European Conference on Machine Learning and Practice of Knowledge Discovery in Databases (ECML/PKDD 2017), Skopje, Macedonia, pages 345-348, 2017.*

This thesis has been submitted for assessment in partial fulfillment of the joint Ph.D. degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Technical Faculty of IT and Design at Aalborg University and the Faculty of Engineering at Université Libre De Bruxelles. The permission for using the published and accepted articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited, DOI pointers and/or copyright/credits are placed prominently in the references.

Part I

Thesis Summary

1 Introduction

1.1 Background and Motivation

In recent years, social networks have undergone a rapid growth. These networks provide the opportunity to build relationships, share multimedia contents, and provide recommendations to users. Social network analysis is being used in a wide range of applications such as on-line marketing, product promotions, recommendations, and health-care [1]. The recent pervasiveness of location-aware devices has managed to reduce the gap between the virtual world of the social networks and the real world. This happened by allowing social network users to share Geo-tagged content such as GPS coordinates of their current location and pictures with geographical tags. Such social networks which are enriched with location information are denoted by the term Location-Based Social Networks (LBSN). Examples of LBSNs are Foursquare, Brightkite, Gowalla, Facebook, and Twitter. The data analysis of these LBSNs is not only being used to improve existing applications that were built using data from traditional social networks but are also a source of novel applications and services. These services usually analyse mobility behaviours of users and utilize them for applications such as product promotion, outdoor marketing, route planning, traffic management, and recommendation services like next location of a user, activities to perform in a particular region, or friends who have similar mobility behavior [2].

LBSN data is composed of two types of entities, user and location. A person who uses the LBSN is termed as a user. It is represented by u. A location in an LBSN is a point of interest (POI) that is represented by a quadruple $(l, lat, long, C)$, where l is the location identifier, lat and $long$ are the GPS coordinates of the geographical space where the POI is situated, and C is the set of categories of the POI. The categories typically show the context of the location and are usually assigned by visitors of the locations. For instance, a restaurant may have categories: "Food", "Restaurant", and "Danish cuisine". Based on these entity types, LBSN data is traditionally divided into two parts based on the interactions of the entities. The first part is a social network containing connections between users called friendship/social graph. Usually, the social graph is an undirected and non-weighted graph where the edges express the friendship relation. The second part of LBSNs is composed of visits/check-ins of users at locations. A check-in of an user at a location is represented by a triplet (u, l, t) where u is the user/visitor, l is the location, and t is the visiting time. In LBSNs, such information is represented by a bipartite graph called activity graph. The activity graph consists of directed edges from users to their corresponding locations and maintains visiting time as an edge attribute. An example of an LBSN is given in Fig-

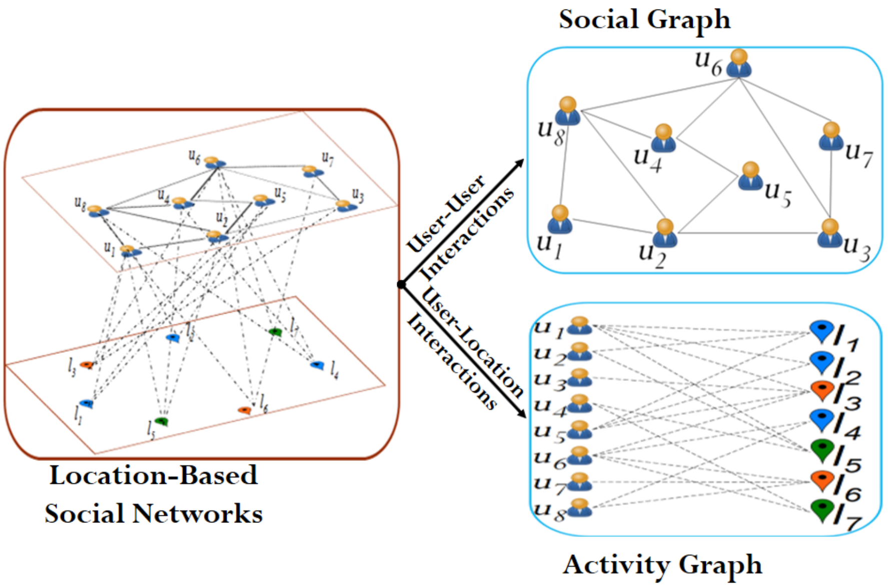

Social Graph
Location-Based Social Networks
User-User Interactions
User-Location Interactions
Activity Graph
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8
l_1 l_2 l_3 l_4 l_5 l_6 l_7

Fig. 2: Example of a traditional way of dividing LBSN data into a social graph and activity graph.

ure 2. Here, $\{u_1, u_2, ..., u_8\}$ and $\{l_1, l_2, ...l_7\}$ represent users and locations, respectively.

The LBSN data along with the rich geographical information inherit several challenges. The first problem is the volume of the data. The LBSN data is huge in size. For instance, the most popular LBSN, Foursquare, has 8M daily check-ins on average, 55M monthly active users, and 100M POIs. Moreover, LBSN data are temporal, spatial and graph nature. These diverse features and the huge size of LBSN data impose several computational and structural challenges in processing. Thus, in order to better exploit LBSNs for improving existing and providing novel applications, we require mechanisms that support efficient and scalable processing of data of such a diverse nature.

1.2 Location Analytics

Usually, the social graphs and the activity graphs of LBSNs are utilized for providing user and activity-based services such as user, activity, and location recommendations [3]. However, the subjective nature of locations was overlooked in the past which can be utilized for diverse applications. For instance, interactions of locations can be used for finding the top-k most famous locations that are visited together, and the top-k most frequent common visitors between two locations. Similarly, in order to find potential visitors for targeted marketing for a trip consisting of multiple locations, we can find the groups of users who frequently have been visiting such locations together in the past. Further, by finding locations that are commonly visited by the users, we can provide some joint incentives/discount offers on the activities that can be performed there or use such information for traffic management, and route planning. Exploiting the activity graph for finding such information is computationally expensive. The reason is that the activity graph only maintains the relationship of users with locations and requires intensive computation to find relationships between locations. Thus, it is important to maintain information of interactions of locations. This information can then directly be used for processing of queries that requires such interactions of locations, termed *Location Analytics queries*.

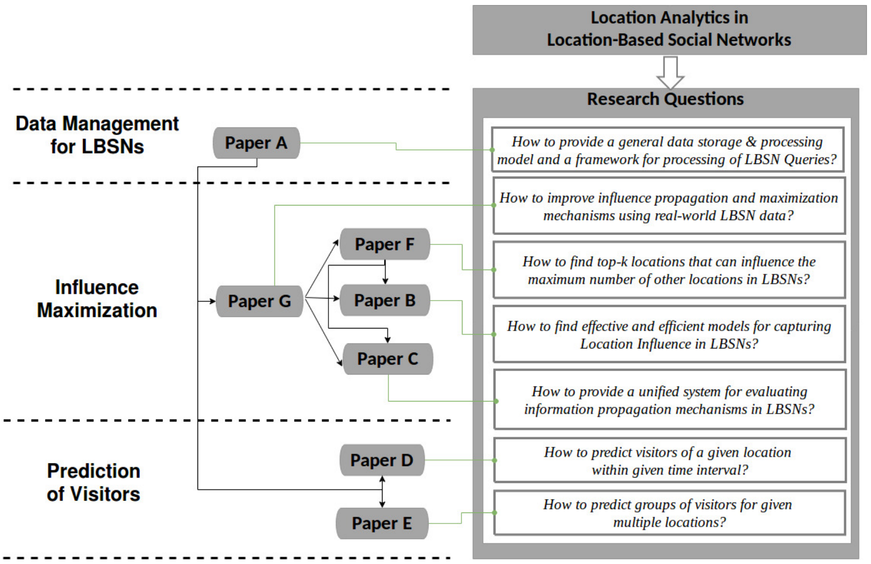

Location Analytics in Location-Based Social Networks
Research Questions
Data Management for LBSNs
Paper A
How to provide a general data storage & processing model and a framework for processing of LBSN Queries?
How to improve influence propagation and maximization mechanisms using real-world LBSN data?
Influence Maximization
Paper G
Paper F
Paper B
Paper C
How to find top-k locations that can influence the maximum number of other locations in LBSNs?
How to find effective and efficient models for capturing Location Influence in LBSNs?
How to provide a unified system for evaluating information propagation mechanisms in LBSNs?
Prediction of Visitors
Paper D
Paper E
How to predict visitors of a given location within given time interval?
How to predict groups of visitors for given multiple locations?

Fig. 3: The structure of the thesis: Paper A introduces the LBSN data model for Location Analytics that is further exploited by rest of the papers. Papers F, B, and C cover location-based influence propagation and maximization methods, where Paper G is an overview paper that covers the idea at an abstract level. Paper F formally introduces the notion of location-based influence. Paper B extends the Paper F by incorporating all of its contents and further providing optimized models for capturing the location influence. Paper F and Paper G, may thus be skipped when reading the thesis cover to cover. In Paper C, a unified system is provided that supports a wide range of information propagation models for LBSNs. Paper D and Paper E provide mechanisms for predicting individuals and groups of visitors at locations, respectively. A logical, optimized, and precise way to go through the thesis is: *Paper A=>Paper B=>Paper C=>Paper D=>Paper E.* Reading Paper F and Paper G is optional.

In this thesis, we provide a general LBSN data storage and processing model with a focus on novel location analytics queries. The overall structure of the thesis is given in Figure 3. The figure shows that the thesis is basically divided into three parts, where we first provide a general data management model for LBSN data and then, based on this, provide novel location analytics queries in the domain of influence maximization and visitors prediction. We further show how the papers are built upon each other with their corresponding research questions. Next, we discuss the topics covered in this thesis, in detail.

2 General and Scalable LBSN Data Management

2.1 Motivation and Problem Statement

The check-in information and social interactions of users in LBSN data are being used for different kinds of services such as recommendations of locations, users, and activities [4–7]. These services require LBSN queries of diverse nature such as finding visitors of a location, range friends and top-k visited locations. The processing of these diverse LBSN queries requires data to be collected, maintained, and processed in ways that require huge storage and computation costs. For instance, finding visitors of a location only requires information about users' activities. However, a range query that aims to find friends of a user within a given radius from his or her location, requires the social network attributes of the user to find friends and spatial measures to calculate the distance among them. In order to tackle such challenges, a general framework [3] for the processing of LBSN queries has been proposed. The authors focus on queries that require social interactions and activities of users. However, their framework overlooked location analytics queries that require interactions of locations. For instance, finding top-k locations that are visited together. The efficient processing of such queries demands an LBSN data model that is generalized enough to support location analytics as well as user and activity analytic queries. Moreover, the enormous amount of LBSN data leads to another challenge of scalability of processing of these queries. This highlights the importance of a data processing system for LBSN data. Several distributed data and graph processing systems have been proposed in past such as Spark [8], SpatialHadoop [9], and GraphLab [10]. However, none of them is capable of efficiently dealing with all the features of LBSN data which are of graph, temporal, and spatial nature. The Paper A addresses aforementioned two challenges that require: 1) a generalized LBSN data storage and query processing model for diverse types of queries. 2) a scalable framework that can process LBSN data efficiently.

2.2 LBSN Data Storage and Processing Model

In order to provide the generalized LBSN data model, we exploit the entities of LBSNs: users and locations, and their interactions with each other such as *user-user*, *user-location*, and *location-location* interactions. Based on these

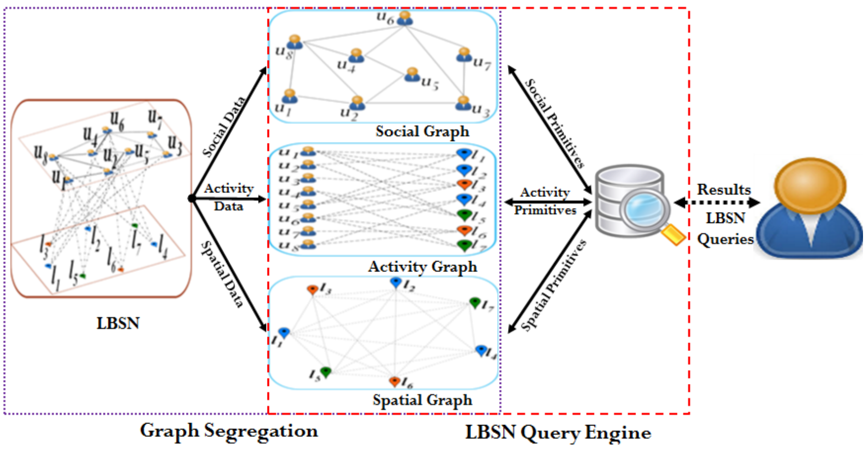


Fig. 4: A segregated graphs (the social graph, the activity graph, and the spatial/Location graph) based LBSN data storage and query processing model [11].

interactions, we segregate the LBSN data into three graphs, the *Social graph* (G_S), the *Activity graph* (G_A), and the *Spatial/Location graph* (G_L), respectively, as shown by the *Graph Segregation* in the Figure 4. These graphs are defined below. The following definitions are reproduced from [11].

Definition 0.1. *$SocialGraph(G_S)$ is an undirected graph, where vertices represent users and edges represent relationships/ friendship between them. It is given by, $G_S = (U, F)$, where U is the set of vertices that represents users, and F is the set of edges representing connections between them, i.e., $F \subseteq U \times U$.*

The activity graph records the visits/check-ins of users at locations (user-location interactions) with the corresponding visiting time in a bipartite graph.

Definition 0.2. *$ActivityGraph(G_A)$ consists of users and their Activities. It is given by $G_A = (U, L, A)$, where U and L are the two sets of vertices: users and locations, respectively. A is the set of edges which represents activities of users U at locations L at times T. It is given by $A \subseteq U \times L \times T$.*

The location graph captures the interactions of locations. In this paper, we consider the edges between locations based on following definition of common visitors of the locations. Our data model can support other various definitions of the locations' interactions such as based on geographical distance, trajectories, and mutual location categories.

Definition 0.3. *$CommonVisitors(U_c(l_m, l_n))$ is the set of users who visited the locations l_m and l_n. It is given by: $U_c(l_m, l_n) = \{u \in U | \exists (u, l_m, t_a), (u, l_n, t_b) \in A\}$.*

Graphs	Categories	Primitives	Notations
G_S	Selection	FindFriends	FF
	Structural	FindSocialSeparationDegree FindCliqueFriends	FSoSD FCF
	Aggregate	Top-k ConnectedUsers	TCU
G_A	Selection	FindUserLocation FindLocationVisitors	FUL FLV
	Structural	FindRangeLocations FindSpatialSeparationDegree	FRL FSpSD
	Aggregate	Top-k NearestLocations Top-k Visitors Top-k VisitedLocations	TNL TV TVL
G_L	Selection	FindCommonVisitors FindLinkedLocations	FCV FLL
	Structural	Find LocationSeparationDegree	FLSD
	Aggregate	Top-k VisitedLinkedLocations Top-k ConnectedLinkedLocations	TVLL TCLL

Table 1: Query Primitives for social graph, activity graph and spatial graph [11].

The location graph based on the common visitors is defined as:

Definition 0.4. *The SpatialGraph (G_L) graph represents the locations as vertices and the relationships as edges. It is given by $G_L = (L, E)$, where L is the set of vertices and E represents the set of edges, i.e., $E \subseteq L \times L$. An edge is a pair of locations having common visitors.*

Next, for scalable processing, we exploit the parallel nature of these segregated graphs. We define query primitives for each of these graphs. To process an advanced query, we divide it into these primitives and run them on corresponding graphs in parallel as shown in the *LBSN Query Engine* of Figure 4. The query primitives for all graphs are given in Table 1. The query primitives are further divided into three types based on their functionalities: selection, structural, and aggregate primitives as shown in the table. For instance, the selection queries such as finding friends, locations of users, and common visitors of locations are considered selection primitives. The queries that require structural processing of graphs such as finding clique friends and separation degrees are considered structural primitives. The top-k related aggregated queries such as finding top-k visitors and nearest locations are termed aggregate primitives.

Example 2.1
For the sake of example, we show the processing of an advanced query: *RangeFriends* as given in Figure 5. We divide the query into three query primitives, *FindFriends, FindLocationVisitors* and *FindRangeLocations*. We first process *FindFriends* and *FindRangeLocations* in parallel on the social

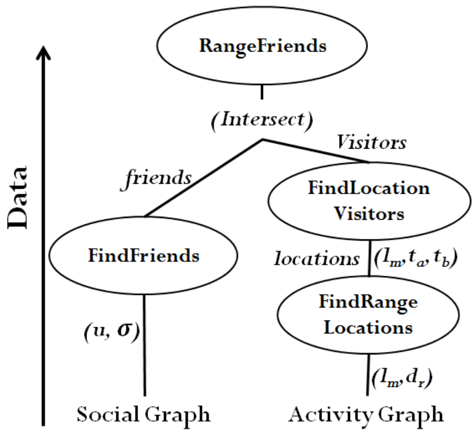


Fig. 5: Query Plan for Range Friends [11]

graph and the activity graph, respectively. Then, based on the results of *FindRangeLocations*, we further compute *FindLocationVisitors*. Finally, we combine the results with the output of *FindFriends* to get the *RangeFriends*.

2.3 GeoSocial-GraphX

In order to find an underlying tool for implementing the proposed LBSN data model and processing of query primitives, we evaluate different open source distributed tabular data, graph, and spatial data processing systems. We choose GraphX, an Apache Spark based graph-parallel system for building our proposed framework called *GeoSocial-GraphX* (GSG) based on its capabilities of efficient processing of both tabular and graph data. GSG inherits the properties of tabular and graph data processing systems from Spark and GraphX, respectively. GSG further provides support for efficient spatial data processing. The architecture of the GSG is divided into four parts based on their functionalities as shown in Figure 6. 1) The *Storage layer* manages the appropriate data structures for storing the LBSN graphs. To do that it utilizes the extension of Spark's Resilient distributed dataset (RDD) by GraphX: VertexRDD and EdgeRDD for storing vertices and edges, respectively. 2) The *Operation layer* is responsible for providing operators for processing of the query primitives. The operators are divided into two types, core operators which are baselines such as *mapEdges*, and *triplets* and advanced operators that utilize core operators for processing such as *Pregel* and *ConnectedComponents*. 3) The *Index layer's* job is the efficient processing of query primitives that involve spatial attributes. We provide a two-level indexing mechanism that consists of a *Global Index* and a *Local Index*. The global index is responsible for distribution of data across nodes of the cluster where the local index handles the data partitioning within a node. To do that we provide standard indexing mechanisms, such as Quadtree, Octree and k-d tree. 4) The *Query Engine* provides the implementation of the query primitives that can further be extended for implementing advanced LBSN queries.

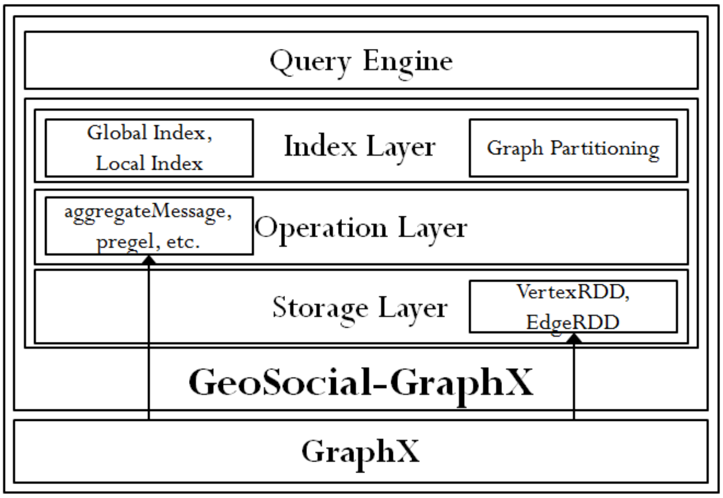


Fig. 6: Architecture of GeoSocial-GraphX [11].

2.4 Discussion

For the experimental evaluation, we use three datasets taken from real-world LBSNs, Foursquare, Brightkite and Gowalla. We first evaluate the effectiveness of the LBSN data storage and query processing model by implementing a set of social, activity and location analytics queries. We show that the location graph of our proposed model can be used to process the location analytics queries up to 4 orders of magnitude faster than a traditional LBSN data model as shown in Table 2. Although the computation of the location graph is more time consuming, we show that it is offset after a single query. Next, we analyse GSG's scalability and optimization based on spatial indexing and show that it outperforms GraphX up to 7x for location analytics queries. We also compare the query performance of GSG with a Hadoop based spatial data processing framework, SpatialHadoop (SH). The results show that GSG outperforms SH in terms of runtime and scalability up to 20x and 7x, respectively. The overall evaluation results validate the significance of the proposed LBSN data model and show that GSG outperforms the competitors in terms of efficiency and scalability.

Data set	Graph	Time	Query	Activity Graph				Spatial Graph			
				1	5	10	15	1	5	10	15
Brighkite	G_A	43.7	FCV	7.5	35.4	71.3	109.1	0.3	1.6	3.1	3.9
			FLL	31125.6	155625.9	311254.9	466881.9	2.4	11.9	21.5	32.6
			TCLL	31127.5	155620.9	311253.4	466880.5	2.4	7.7	14.7	21.3
	G_L	31127.12	TVLL	31125.4	155622.8	311249.7	466871.5	1.7	6.3	12.2	17.8
			FLSD	31127.2	155632.6	311264.6	466863.8	3.4	11.6	22.3	30.5

Table 2: Comparison of computation time (in seconds) of spatial primitives on activity graph and location/spatial graph [11]

3 Location Influence

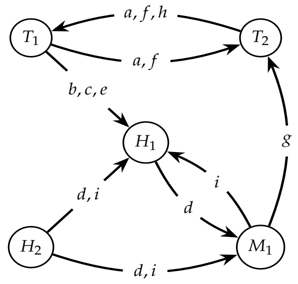
3.1 Motivation and Problem Statement

One of the applications of the social network analysis that gained significant importance in recent past is viral marketing using influence maximization. In these studies, influence of a user is measured by the number of followers of his activities. For instance, if a user u posts a status on social media, the number of his friends who re-shared his status within a particular time defines the influence of u. In a viral marketing approach, these influential users are utilized for product promotion. They are usually asked to share promotional messages for a product to maximize its visibility among his followers. Existing influence maximization studies focus on finding such top-k influential users for product promotion via word of mouth marketing. However, the geo-tagged contents of LBSNs allow us to further exploit it for other marketing approaches which require displaying ads on billboards and banners such as out-of-home/outdoor marketing. Outdoor marketing takes approximately 6% of the total marketing budget [12] which shows the significance of such approaches. Paper G shows how the activity data of users from social networks and LBSNs can be utilized to optimize the information propagation and influence maximization mechanism for diverse marketing approaches.

We exploit the location graph and utilize the locations' interactions for proposing an information propagation mechanism that can be used for such marketing approaches. On the basis of the proposed mechanism, we compute the influence of locations on each other and find top-k influential locations. For example, consider a marketer who aims to promote his product to the maximum regions of a city by giving free promotional items such as a cap with a promotional message on it. In order to do that he needs to find locations where he should distribute those caps to visitors so that the message is spread to the maximum regions. The paper F focuses on finding such locations. We formally define our problem statement [13]: *Given an LBSN and a parameter k, find a set of k locations such that their combined location influence is maximal.*

Check-ins

loc	Users		
	t=1	t=2	t=3
T_1	b, c, e, f	a, h	f
T_2	a, h	f, g	a
M_1	g	i	d
H_1	–	b, c, d, e	i
H_2	d, i	–	–



Friendships

User	Friends
a	c, h, i
b	d, f, i
c	a, f
d	b, e
e	d, g
f	c, b
g	e
h	a
i	a, b

Fig. 7: Running example of an LBSN: Check-ins (L) shows the visits of users (represented by small letters) at locations (represented by capital letters) at time stamps t=1,2 and 3. Graph(C) depicts the movement of users between consecutive locations. Friendships(R) show the friends of each user in the social network. [13]

3.2 Location-Based Influence

We first introduce a notion for capturing such influence among locations, called *Location Influence*. We define the location influence of a location s by its capacity to spread its visitors to another location d within a particular given time ω. The intuition behind such an influence is that users who are exposed to a message at a location promote it indirectly by talking to their friends and colleagues at the locations they visit afterwards. Such visitors/information carriers are called *Bridging Visitors*.

Example 3.1
Consider the running example, given in Figure 7. The bridging visitors $B_D(T_1, T_2)$ from location T_1 to location T_2 within time window $\omega = 2$ are a and f.

Next, in order to measure the location influence between two locations s and d, we propose following two location influence models. 1) The *Absolute Influence model* captures the influence based on the number of bridging visitors. The intuition behind this is that a compelling activity is usually repeated by a significant number of people. The absolute location influence $I_A(s, d)$ of s on d is given by [13].

$$I_A(s, d) := \begin{cases} 1, & \text{if } |B(s, d)| \geq \tau_A \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $B(s, d)$ is a set of bridging visitors from s to d and τ_{DA} is absolute influence model threshold.

2) The *Relative Influence Model* measures the relative influence $I_{DR}(s, d)$ from s to d based on the ratio of the number of bridging visitors from s to d and the number of visitors at d ($V_D(d)$). The motivation behind finding such an influence is to avoid bias towards considering all popular locations with many visitors as influential locations. The relative influence of s on d is given by [13]:

$$I_R(s, d) := \begin{cases} 1, & \text{if } \dfrac{|B(s, d)|}{|V(d)|} \geq \tau_R \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $V(d)$ is the set of users who visited location d and τ_R is relative influence model threshold.

Example 3.2
In the running example given in Figure 7. Suppose $\tau = 2, \omega = 2$ and $\tau_R = 0.4$, then $I_A(T_1, H_1) = 1$ and $I_R(T_1, H_1) = 1$.

Usually, users like to follow the activities of their friends. We find this observation by analysing the LBSN datasets. Based on this observation, we propose a friendship based location influence model. In this model, we incorporate friends of bridging visitors as potential information carriers. To determine the friendship based influence, we replace $B(s, d)$ in Equation 1 and Equation 2 by a set of bridging visitors from s to d together with their friends. Further, in Equation 2, we replace $V(d)$ with a set of visitors of d and their friends.

A location s may not influence another location d alone but together with other locations may influence it. For instance in Figure 7, T_1 and M_1 alone cannot influence T_2 for $\omega = 2$ and $\tau_A = 3$. However, together they influence T_1. We called such an influence, *Combined Location Influence (ϕ_I)*. We aim to find the locations that maximize such combined influence.

3.3 Influence Oracle

Next, in order to compute the location influence, we first provide a data structure called *Influence Oracle* and then we use it for finding top-k influential locations. In influence oracle, for each location, we maintain a list of locations which have at least one bridging visitor from the location, called *Location Summary*. We provide an on-line incremental algorithm for finding such location summaries. To do that, we maintain sets of user histories that record the visits of users with corresponding locations and latest visiting time. As our input check-in data is sorted in increasing order of time, we update user histories on every new check-in/visit. For a check-in (u, l, t), we update the location summary of a location l by adding the set of locations that are visited by a user u within time $[t - \omega, t]$. For the relative model, we further maintain total visitors of every location.

Maintaining the location summaries requires a lot of space and processing time. In order to improve the space and time efficiency, we provide a probabilistic data-structure, modified Hyperloglog (*mhll*) to store the location summaries. Based on this, we improve both space and time efficiency as given in Table 4. Once we computed such data-structure, we utilize standard greedy algorithms for computing top-k influential locations.

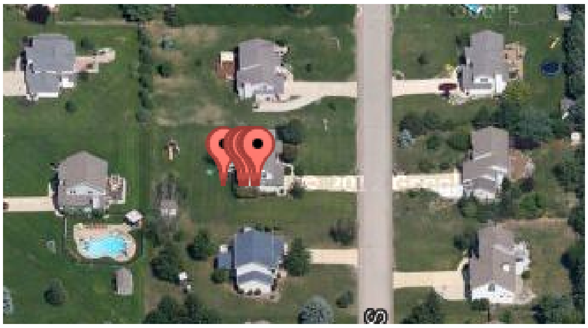
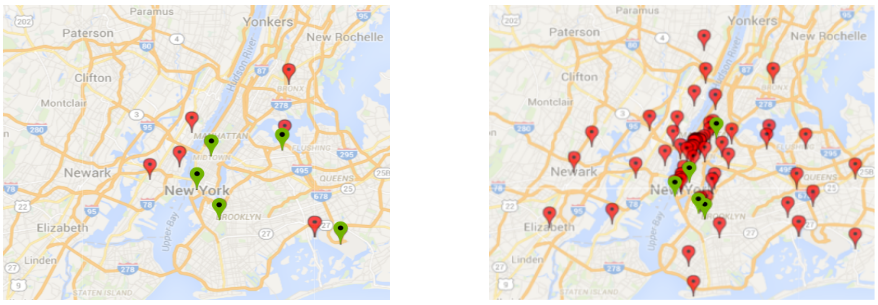


Fig. 8: GPS coordinates of 13 location-ids on GoogleMaps [13].

3.4 Discussion

In order to evaluate the proposed concepts and algorithms, we utilized three datasets, Foursquare, Brightkite, and Gowalla. In these datasets for some of the GPS coordinates, multiple location ids are provided. For instance, in



(a) Naive `BrightKite` (16 locations) **(b)** Our `BrightKite` (72 locations)

Fig. 9: Comparison of top- 5 influential locations (green) and their spread (red) between naive and our approach [13].

Figure 8, 13 GPS coordinates with different location ids are associated with the same house. To avoid such issues, we cluster the GPS coordinates to get POIs using density-based spatial clustering. We use such POIs for all the experiments. We performed detailed experimental analysis and provide the suitable values for the thresholds of the location influence models. We also show the effect of these thresholds on the influence spread and computational resources. In order to evaluate the effectiveness of our proposed approach, we compare the influence spread of top-k locations fetched by our proposed approach and a naive greedy approach. Our proposed approach outperforms the naive approach in influence spread by 5 times as shown in Figure 9. We further analyse the effect of approximate algorithms on the accuracy. To do that for each location, we compute the relative error for the influence set and compare it with the exact approach. We show that the approximate algorithms require 5 times less computation time and 4 times less memory as compared to the exact algorithms.

4 Effective and Efficient Location Influence Mining

4.1 Motivation and Problem Statement

The *Location Influence* introduced in Paper F [13] can be used for applications like outdoor marketing, monitoring and controlling of a virus and fake news propagations. Several models for capturing such location influence are provided. However, no quantitative comparison of influence spread of these models is provided. Furthermore, in the friendship-based influence model, all the friends are assumed to follow the activities of bridging visitors. However, usually, in real-life, only a subset of friends follow the activities. Thus, it is important to identify and incorporate significantly influenced friends to optimize the accuracy of the influence spread.

Moreover, in real life, there exist several cases in which a single influencer may carry the influence. For instance a virus spread or information propagation in specialized information networks. LBSN data is also often sparse which leads to very few visitors to locations. For such cases, we can more efficiently compute the location influence. To do that, in Paper B, which is an extension of Paper F, we first focus on an effective influence model that incorporates the influenced friends of bridging visitors for finding location influence. Then, we further focus on efficient algorithms for finding single influencer based location influence. Finally, we emphasize on an extensive experimental evaluation to show the effectiveness of the proposed model in comparison with a baseline naive approach and as well as state-of-the-art influence models.

	Location-Based Influence Models					
	Absolute Influence			**Relative Influence**		
	Direct Bridging Visitors	**Friends-Based Bridging Visitors**	**Influenced Friends-based Bridging Visitors**	**Direct Bridging Visitors**	**Friends-Based Bridging Visitors**	**Influenced Friends-based Bridging Visitors**
Label	Direct Absolute Model (M_{DA})	Friends Absolute Model (M_{FA})	Influenced Friends Absolute Model (M_{IA})	Direct Relative Model (M_{DR})	Friends Relative Model (M_{FR})	Influenced Friends Relative Model (M_{IR})
Parameters	τ_{DA}, ω	τ_{FA}, ω	$\tau_{IA}, \omega, \theta$	τ_{DR}, ω	τ_{FR}, ω	$\tau_{IR}, \omega, \theta$
Location Influence	I_{DA}	I_{FA}	I_{IA}	I_{DR}	I_{FR}	I_{IR}

Table 3: Summary of different types of Location-Based Influence Models with corresponding notations and threshold [14].

4.2 Effective Influence Models

In order to find friends that most likely follow activities of users, we first find the influence of users among each other based on their visiting activities. We consider influence of a user u on a user v if v visits a location l after u within a particular time. To compute such influence of u on v, we provide an algorithm that computes the influence probabilities using Bernoulli distribution that computes this by the fraction of the number of times v followed the activities of u and the total number of activities of v. If v is influenced by multiple other users, we divide the influence credit among all such influencer users using a partial credit distribution model. Furthermore, considering the observation that usually influence of an activity remains active within a particular time, we incorporate a discrete-time based model and consider the influence if it is spread within the given time window. We incorporate the activities that are performed at the same locations for measuring influence. We further introduce a threshold θ and consider the influence between two users if the influence probability is greater than the threshold. Furthermore, it is worth noting that a user u alone may not influence v but along with other users may influence him. Thus, we consider the complete bridging visitor set for a pair of locations (influential and influenced) for finding their influenced users. Next, for finding absolute location influence based on these influenced friends of the bridging visitors, we replace the bridging visitor set with a set of users containing the bridging visitors and their corresponding influenced friends called Influenced Friends-based bridging visitors. It is given by [14]

Definition 0.5. *Influenced Friends-Based Bridging Visitors: Given an LBSN(G_S, A), time window ω, locations s and d, $B_D(s,d)$, and a threshold of the influence probability between users θ, a set of* Influenced Friends-Based bridging visitors *between s and d is denoted by $B_I(s,d)$:*

$$B_I(s,d) = B_D(s,d) \cup \{u \in U | \sum_{v \in B_D(s,d)} p_{v,u} \geq \theta\} \tag{3}$$

	Exact			**Approx**		
	Memory	Oracle Time	Query Time	Memory	Oracle Time	Query Time
M_{DA}	$O(\lvert U\rvert(\lvert L\rvert^2+\omega))$	$O(\omega\log(\lvert U\rvert)\lvert A\rvert)$	$O(\lvert S\rvert\lvert L\rvert\lvert U\rvert\lvert U\rvert)$	$\mathcal{O}(\lvert L\rvert^2 b+\lvert U\rvert\omega)$	$O(\omega\lvert A\rvert)$	$O(\lvert S\rvert\lvert L\rvert b)$
M_{DR}				$\mathcal{O}(\lvert L\rvert^2 b$ $+\lvert U\rvert\omega+\lvert L\rvert\lvert U\rvert)$		
M_{FA}	$O(\lvert U\rvert(\lvert L\rvert^2+\omega$ $+\lvert U\rvert))$	$O(\omega\lvert U\rvert\log(\lvert U\rvert)\lvert A\rvert)$		$\mathcal{O}(\lvert L\rvert^2 b$ $+\lvert U\rvert\omega+\lvert U\rvert^2)$	$O(\omega\lvert U\rvert\lvert A\rvert)$	
M_{FR}				$\mathcal{O}(\lvert L\rvert^2 b$ $+\lvert U\rvert\omega+\lvert U\rvert^2$ $+\lvert L\rvert\lvert U\rvert)$		
M_{IA}		$O(\omega\lvert U\rvert\log(\lvert U\rvert)\lvert A\rvert$ $+\lvert L\rvert^2\lvert U\rvert^2$ $+\lvert A\rvert(\lvert A\rvert+\lvert U\rvert))$		N/A		
M_{IR}						

Table 4: Summary of time and space complexities for the influence models [14].

where $p_{u,v}$ show the influence probability of user u on user v. For the relative location influence model, we further incorporate the influenced friends of the visitors of the destination location. Moreover, we introduce minimum influence thresholds for both the absolute and the relative influenced friends-based influence models. All the proposed influence models with corresponding parameters and notations are given in Table 3.

In order to find top-k locations using influenced friends-based influence model, we first compute the influence oracle which contains the location summaries of all the locations and then used it for finding top-k locations. To do that, we first compute the influence probabilities of users among each other using the aforementioned algorithm. Next, we consider the bridging visitors of each influential and influenced pair of locations and incorporate their influenced visitors having influence probability greater than the influenced visitor based influence threshold θ. It is important to note that this is an off-line algorithm. The is reason is that we need to first find all the bridging visitor of the locations which requires a complete traversal of the data and then find their combined influenced visitors. We follow the same procedure for incorporating the influenced friends for the visitors of the destination locations in the relative influence model. After computing the influence oracle, we use the standard greedy algorithm to compute the top-k locations. For this model, we compute the exact influence. The reason is that the probabilistic data structure (mhll), which is used in approximate algorithms for other models provides an approximate count of a set. However, for this model, we need Ids of the bridging visitors for finding the influenced visitors. The time and space complexity for all the influence models is given in Table 4.

4.3 Efficient Location Influence Models

Next, we provide efficient algorithms for single influencer based influence. These algorithms improve the computation time for building an influence oracle by exploiting the following observation. For single influencer, we do not need to maintain the bridging visitor sets for locations as only one visitor is enough to carry the influence. Moreover, considering that we only need cardinality of the influenced locations for finding top-k locations, we replace the actual locations summary sets with a HyperLogLog (HLL) set. We provide two algorithms for computing single influencer based location influence, *On-Sin* which is an on-line algorithm and *Off-Sin* which is an off-line algorithm but a more efficient algorithm. For the On-Sin algorithm, for every visit, we add the current location of the user as an influenced location for all the locations in the user history, i.e., the locations user has visited earlier within time window ω. For the off-line algorithm, we traverse the check-ins in a reverse order. For every visit, we add all the locations within ω in the user's history into the location summary of the visited location. Off-Sin is more efficient than On-Sin. The reason is that for On-Sin, for each visit, the number of location summaries that need to be updated is the total number of locations in the user's history within ω. However, for Off-Sin, for each visit at a location, we only need to update the location summary of that visited location. The time and space memory for these algorithms are given in Table 5.

	Off-Sin			**On-Sin**		
	Memory	**Time Oracle**	**Time Query**	**Memory**	**Time Oracle**	**Time Query**
M_A	$\mathcal{O}(b(\lvert L\rvert + \lvert U\rvert \log \omega))$	$O(b \log(\omega)\lvert A\rvert)$	$O(\lvert S\rvert b)$	$\mathcal{O}(b\lvert L\rvert + \lvert U\rvert\omega)$	$O(\omega\lvert U\rvert\lvert A\rvert)$	$O(\lvert S\rvert b)$
M_R	$\mathcal{O}(b(\lvert L\rvert + \lvert U\rvert \log \omega) + \lvert L\rvert\lvert U\rvert)$			$\mathcal{O}(b(\lvert L\rvert + \lvert U\rvert \log \omega) + \lvert L\rvert\lvert U\rvert)$		

Table 5: Summary of time and space complexities for Single influencer-based Influence [14]

4.4 Discussion

For experimental evaluation, we compare the effectiveness of our proposed models with a PageRank based approach, $PR - LCG$ and two variants of the most relevant state-of-the-art approach, IRS and $IRS - window$. In order to evaluate, we divide the datasets such that each part consists of check-ins of one month. We compute the top-k influential locations using the dataset of one month and evaluate their spread on the dataset of the next month in the sequence. We repeat this for all the parts of the dataset and report the total influence spread. We compute the spread for 5 different values of k, i.e., 1, 3, 5, 10, 15, and 20 to have statistically significant results. The results are given in Table 6. It can be observed that in terms of influence spread, our proposed models outperform all the competitors. Overall the absolute influence models perform better than the relative influence models. More specifically, the spread of the top-k locations fetched by M_{IA} is up to 45%, 700%, and 400% more than that of M_{DA}, M_{FA}, and $IRS - window$, respectively. This shows the significance of considering influenced friends-based bridging visitors for finding location influence. Moreover, we compare the computational resources required by the single influencer based location

Dataset	K	Number of Influenced Nodes								
		Absolute Influence Model			Relative Influence Model			IRS	IRS-window	PR-LCG
		M_{DA}	M_{FA}	M_{IA}	M_{DR}	M_{FR}	M_{IR}			
FourSquare	1	89	37	**98**	66	66	66	79	71	N/A
	3	190	116	**195**	159	160	159	82	105	
	5	230	117	**255**	216	196	216	121	116	
	10	322	159	**361**	273	260	273	124	167	
	15	380	200	**421**	307	288	307	130	174	
	20	432	255	**495**	330	326	330	132	191	
BrightKite	1	662	657	**671**	648	655	648	512	537	657
	3	943	882	**959**	896	882	896	613	743	924
	5	**1,153**	9,67	1,112	1,041	994	1,041	666	835	1,100
	10	1,458	1,140	**1,465**	1,269	1,259	1,269	749	1,027	1,437
	15	**1,717**	1,257	1,686	1,449	1,444	1,449	821	1,171	1,693
	20	1,921	1,381	**1,951**	1,589	1,570	1,589	867	1,275	1,928
Gowalla	1	676	111	**982**	446	405	446	453	613	N/A
	3	**1804**	153	1787	1129	1212	1129	821	1072	
	5	2834	238	**3087**	1772	1716	1772	997	1134	
	10	4875	860	**5197**	3302	3218	3302	1116	1445	
	15	6236	1395	**6767**	4136	3645	4136	1244	1854	
	20	7460	1877	**8165**	4668	4119	4668	1452	2015	

Table 6: Influence spread of top-k influential locations fetched by the proposed influence models, IRS, $IRS - window$ and $PR - LCG$. The check-ins are divided on monthly basis. The influential keys are fetched on one part and spread is computed on the next part in the sequence. The process is iteratively repeated for all the months and the total influence spread for each value of k, for each dataset is depicted [14].

influence models with that of the other models. The results show that the single influencer based algorithms require up to 20 times less computation time and 50 times less memory for finding the location influence.

5 Unified System for Influence Propagations in LBSNs

5.1 Motivation and Problem Statement

LBSNs are being utilized for providing several information propagation mechanisms such as Location influence (Paper F) and users' activity based information propagations methods. These information propagation methods have diverse objectives such as viral marketing, outdoor marketing, and location promotion. Thus, for a particular use-case, there might be several ways to propagate information and maximize the influence.

Example 5.1
For instance, a marketer is interested to promote its product to the maximum regions. To do that, he would like to find a better method between following two options.

- *Case 1:* He can display the advertisement using billboards on locations which are visited by the most frequent travelers who after seeing the message may spread it to the regions they visit.
- *Case 2:* He can give some promotional items such as t-shirts with a promotional message on it to the frequent travelers who spread these messages indirectly to the people of regions they visit.

In order to explore, analyze and compare these diverse information propagation and influence maximization mechanisms, we need a system that can support processing of these methods. Moreover, the process of influence maximization requires several steps: data selection, data pre-processing, influence maximization, and spread estimation. Thus, the system should be able to provide all of these steps. The Paper C focuses on providing such a system that can handle these aforementioned challenges.

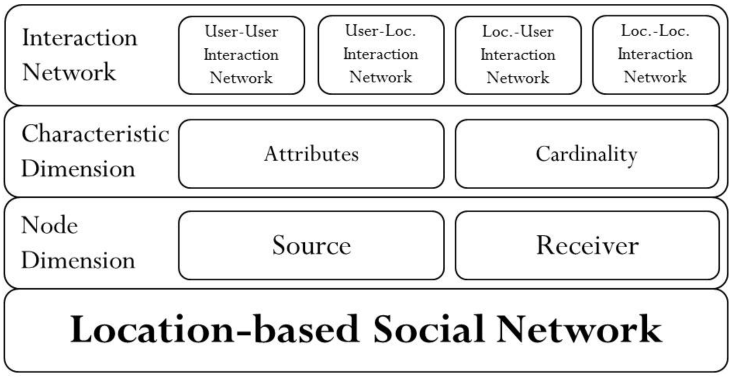

Interaction Network
User-User Interaction Network
User-Loc. Interaction Network
Loc.-User Interaction Network
Loc.-Loc. Interaction Network
Characteristic Dimension
Attributes
Cardinality
Node Dimension
Source
Receiver
Location-based Social Network

Fig. 10: The Unified Influence Maximization Model [15]

5.2 Unified Influence Propagation Model in LBSNs

In order to provide such a system, we identify diverse information propagation mechanisms in the LBNSs. To do that, we exploit the interactions of users and locations. Based on these interactions, we provide a unified model that supports information propagations through different sorts of interactions.

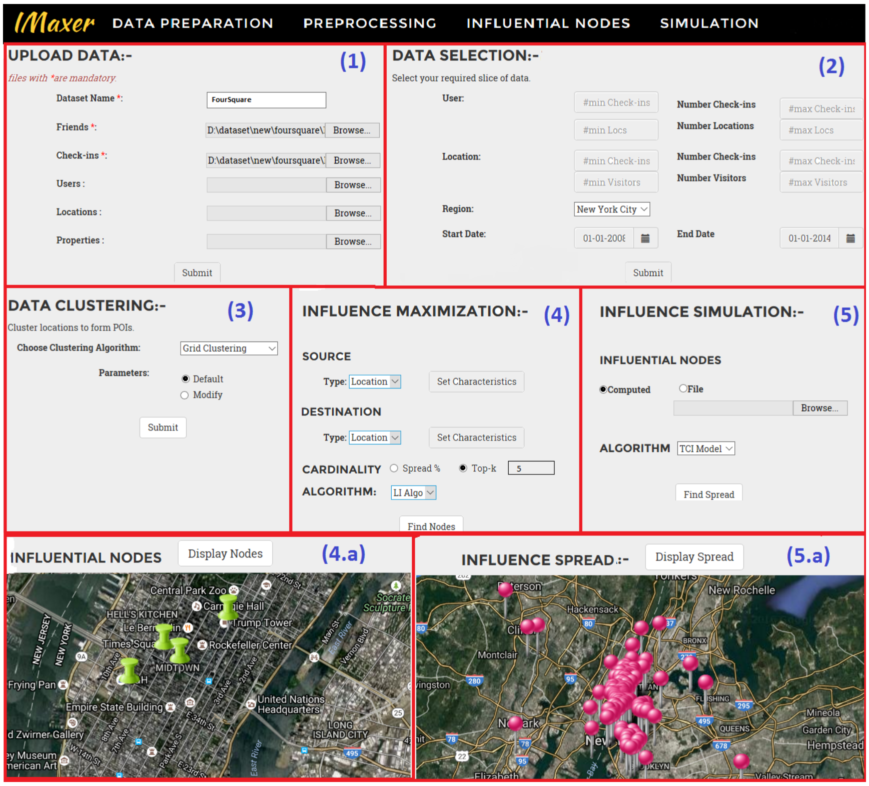
The unified model has three layers as shown in Figure 10. 1) The *Node Dimension* identifies the source (influential) and destination (influenced) vertices based on the type of influence. For instance, in case 1, it is intended to spread the influence from locations (billboards) to the users of the maximum number of other locations. Thus, both the source and the destination vertex will be location in this case. However, in the second case, influence is to be spread by the visitors having promotional items to the users of other locations. In this case, the source vertex is user and the destination (influenced) vertex is location. 2) The *Characteristic Dimension* identifies the features of the source and destination nodes. For instance, if it is intended to spread messages from a specific community or an age group then only corresponding nodes of users will be considered as source vertices. Further, it is also used for estimating potential spread. For example, the cardinality of influential nodes that should be used to spread a message. 3) The *Interaction Network* layer provides a graph of influential and influenced nodes based on the node selection in the node dimension. These networks are further utilized for influence maximization and spread simulation. With this unified model, we can support four different types of information propagation and corresponding influence maximization mechanisms as shown in Table 7.

Source	Dest.	Influential Activity ($\mathcal{I}$)	Application (Maximizes)
User	User	u follows u'	Followers
Location	Location	l spreads visitors to l'	Geographical spread
User	Location	u visits l'	Unique visited locations
Location	User	l visited by u'	Unique visitors

Table 7: Information propagation mechanisms w.r.t. types of the influential and influenced nodes in LBSNs. Here, u and l are the influential user and location, respectively and u' and l' are the influenced user and location, respectively. [15]

5.3 IMaxer

Next, based on the unified influence maximization model, we provide a system called *IMaxer*. We divided the IMaxer into four modules based on traditional data mining steps as shown in Figure 11. 1) *Data Selection:* This module is responsible for data managing tasks such as uploading LBSN data in a given format, slicing it based on input attributes for users, locations, activity time and geographical regions. 2) *Data Preprocessing and transformation:* In this module, we pre-process the data based on user requirements. For instance, by clustering, granularities of locations/POIs can be changed based on the demand. 3) *Influential Nodes Mining:* In this module, we provide several state-of-the-art influence maximization algorithms that can be used for finding influential nodes using the unified influence maximization model. 4) *Spread Simulation:* Finally, the potential spread of the fetched top-k nodes is estimated by this module. We provide a set of algorithms for computing the potential spread. We further visualize the influential and influenced locations using the Google Maps APIs.



UPLOAD DATA:-

*files with *are mandatory.*

Dataset Name *:	FourSquare
Friends *:	D:\dataset\new\foursquare\ Browse...
Check-ins *:	D:\dataset\new\foursquare\ Browse...
Users :	Browse...
Locations :	Browse...
Properties :	Browse...

Submit

DATA SELECTION:-

Select your required slice of data.

User:	#min Check-ins	Number Check-ins	#max Check-ins
	#min Locs	Number Locations	#max Locs
Location:	#min Check-ins	Number Check-ins	#max Check-ins
	#min Visitors	Number Visitors	#max Visitors
Region:	New York City		
Start Date:	01-01-2008	End Date	01-01-2014

Submit

DATA CLUSTERING:-

Cluster locations to form POIs.

Choose Clustering Algorithm: Grid Clustering

Parameters: [✓ Default] Modify

Submit

INFLUENCE MAXIMIZATION:-

SOURCE

Type: Location Set Characteristics

DESTINATION

Type: Location Set Characteristics

CARDINALITY Spread % [✓ Top-k] 5

ALGORITHM: LI Algo

Find Nodes

INFLUENCE SIMULATION:-

INFLUENTIAL NODES

[✓ Computed] File Browse...

ALGORITHM TCI Model

Find Spread

INFLUENTIAL NODES

INFLUENCE SPREAD :-

Fig. 11: Compact snapshots of usage of IMaxer for maximizing geographical spread with locations in New York City [15]

Example 5.2

Figure 11 show the processing of different modules of IMaxer for the use case 1. Here, we first upload the Foursquare dataset and filter based on location, NYC. Then using grid clustering granularities of the locations are raised. Next, a location influence maximization algorithm [13] is used for finding top-5 influential locations and finally, TCIC [13] model for estimating their spread. It can be observed that using top-5 locations the spread is 236. Similarly, the influence spread for the use-case 2 can be computed and compared with the results of use-case 1 for finding the optimal one.

User	Location	Time
u_1	l_1	13:25 11/12/2017
u_1	l_1	14:30 12/12/2017
u_1	l_3	13:05 14/12/2017
u_2	l_1	13:10 11/12/2017
u_2	l_1	15:10 14/12/2017
u_3	l_1	14:20 11/12/2017
u_3	l_2	13:16 12/11/2017
u_3	l_3	16:20 14/12/2017
u_4	l_2	15:15 11/12/2017

Location	Category	Coordinates
l_1	Sports	42.99,-71.46
l_2	Arts	42.98,-71.45
l_3	Sports	42.97,-71.44

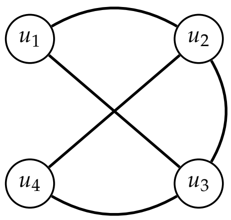


Fig. 12: Toy example: Checkins (left), Location categories (top right) and social graph (bottom right) [16]

6 Prediction of Visitors

6.1 Motivation and Problem Statement

LBSNs have been widely studied for recommendation services such as suggesting potential visiting locations to the users based on their historic activities and interests. However, another perspective of such services which has been often overlooked in the past is the prediction of visitors at locations. Predicting potential visitors for a particular location can be used for applications such as traffic management and event planning. For instance, consider a theatre owner who wants to find out that who will attend the next planned play at his theatre to better organize the event based on the audience. Paper D exploits the concept of finding potential visitors given in Paper B, for such prediction of visitors at the locations. We formally define the problem statement [16]: *Given an LBSN, a location l, and a time interval T, predict the users who will visit l within T.*

6.2 Collective Matrix Factorization-based Visitor Prediction Model (CMViP)

In order to predict visitors, we analyzed the LBSNs data for finding features that affect the mobility of users and provide a potential visitor based prediction model called *CMViP*. CMViP utilizes following five factors that significantly contribute towards visiting a location l for a user u. 1) visit frequency of u at l, 2) visits' frequencies of u at locations having similar categories with l, 3) visits' frequencies of u at a given time interval, 4) distance of l from the current location of u, and 5) influence of u on friends for being followed.

For each user u, we first find its potential for visiting the given location on the basis of the first four aforementioned features. We combine these features using the following linear equation to compute the total potential of a user u for visiting a location l, called *visit score* (Y_S) [16].

$$Y_S(u,l,c,T) = \alpha.Y_U(u,l) + \beta.Y_C(u,c) + \gamma.Y_T(u,T) + \eta.Y_D(u,l) \tag{4}$$

Here, Y_U, Y_C, and Y_T capture the number of activities user u performed at location l, categories of l, and at time T, respectively. Y_D captures the difference between the maximum distance u travelled for the consecutive visits between two locations and the distance of l from the $u's$ current location. Further, α, β, γ, and η are the coefficients for the features showing their corresponding significance for predicting visitors. We provide a minimum visit score threshold θ. The users having visit score greater than θ are termed *Potential Visitors U_P* [16]:

$$U_P(l,c,T) = \{u | Y_S(u,l,c,T) \geq \theta\} \tag{5}$$

Example 6.1
For instance, consider an example given in Figure 12. Here, assume $t_1 = 13:00$, $\alpha, \beta, \gamma, \eta = 0.25$, and $\theta = 0.8$. Then, $Y_S(u_1, l_1, c_1, t_1) = 0.84$. Similarly, $Y_S(u_2, l_1, c_1, t_1) = 0.875, Y_S(u_3, l_1, c_1, t_1) = 0.66$ and $Y_S(u_4, l_1, c_1, t_1) = -\infty$. Thus, $U_P(l_1, c_1, t_1) = \{u_1, u_2\}$.

Next, we further exploit another observation based on the LBSN data analysis that users tend to follow the activities of their friends. Based on this observation, we assume that if a user is visiting a location, his friends may also join him. To find out such friends, we compute the influence of potential visitors, on their friends. Such influenced friends are called *Influenced Potential Visitors ($I(U_P)$)* [16]:

$$I(U_P) = \{v | \sum_{u \in U_P} p(u,v) \geq \xi \wedge (u,v) \in F\} \tag{6}$$

where ξ is the influence threshold that represents the minimum influence probability by which a user should be influenced to be considered a follower. Thus, a set of predicted potential visitors is given by: $U_P \cup I(U_P)$.

Example 6.2
Next, to find the set of predicted potential visitors, we consider the example 6.1 where $U_P = \{u_1, u_2\}$. Further, suppose, $\omega = 1h$ and $\xi = 0.2$. In this case, since, for the location l_1, u_1 and u_2 are followed by u_3 and for the location l_3, u_1 is followed by u_3 within the given time window ω. Thus, the influence probability of U_P on u_3 is $p(U_P, u_3) = 0.3 \geq \xi$ [14]. So, the set of the predicted visitors is $\{u_1, u_2, u_3\}$.

In order to compute the visit score of all the users, we utilize a non-negative collective matrix factorization approach. We compute the four frequency matrices: 1) visitor-location frequency matrix: $Y_U \in \mathbb{R}^{U \times L}$ that captures the number of visits of users U at locations L, 2) visitor-category matrix $Y_C \in \mathbb{R}^{U \times C}$ that captures the number of visits of users for categories C, 3) visitor-time matrix: $Y_T \in \mathbb{R}^{U \times T}$ that shows the number of visits of users for each hour of the day, and 4) visitor-distance matrix: $Y_D \in \mathbb{R}^{U \times L}$ which shows the average distance users U travel for visiting locations L from their last locations. Next, we decompose the frequency matrices into the common latent space and their corresponding feature latent space matrices. For a given location and a time interval, we utilize these matrices for finding the visit scores of the users. To do that, for each user, we compute his potential scores for all the features by combining the common latent space and corresponding feature latent space matrices. These scores are further combined using the Equation 4 for finding the visit scores. Then, based on θ, we find the potential visitors. Next, we find the influenced potential visitors by using a Bernoulli distribution and a partial credit distribution based discrete time constrained model, given in Section 4.2. The predicted set of users for a given query is then computed by combining the potential visitors and the influenced potential visitors.

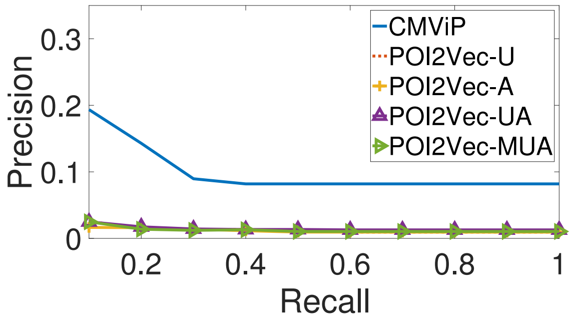


Fig. 13: Precision Recall Curve [16]

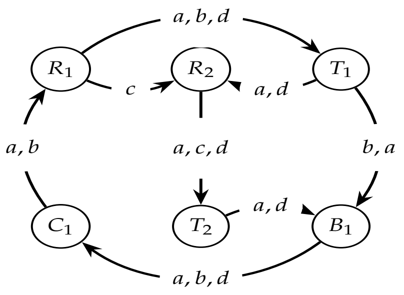
6.3 Discussion

In order to evaluate the effectiveness of CMViP, we utilize two real-world datasets with two different natures, 1) Foursquare that is smaller but denser and 2) Wee which is larger but sparser. We compute the accuracy of predicting the visitors using the precision-recall curve. To do that, we sort the dataset in ascending order of visiting time and divide it into two parts, the training data and the test data. We utilize the training dataset for predicting the visitors and then compare it with the visitors in the test dataset. We compare the results of CMViP with four variants of a state-of-the-art approach POI2Vec [17]. The results show that CMViP outperforms POI2Vec up to 6 times as shown in the Figure 13. We further also show the effectiveness of CMViP for another use-case of finding potential visitors, that is predicting the number of visitors. To do that, we compute mean absolute error (MAE) and root-mean-square error (RMSE) for evaluating the CMViP performance. We show that the error values decreases up to 2 for $\theta = 0.4$ while considering all five factors. This shows the significance of the factors incorporated by CMViP for prediction.

7 Prediction of Geo-Social Cohorts

7.1 Motivation and Problem Statement

A prominent type of recommendations pertains to *groups of users*, as opposed to individuals as given in Paper D. For instance, a discount offer for a skydiving activity may be recommended to a group of friends that have performed similar activities in the past. Similarly, mobility activities of users can be utilized to predict a group that would be interested to join an event. Finding such groups can be utilized for several applications. For instance, consider the following example. A travel agency owner, *Viking Travel*, is interested to promote an offer for a group travel package in the Norwegian Arctic towards groups of users potentially interested in joining such a given travel package together. Thus, he would like to find such potential group of users. Existing research has investigated several aspects of such location-based recommendations [2]. Yet, there has been no attempt to combine information about social ties and mobility to predict potential groups of users (cohorts) who will perform multiple given activities together using LBSN data. Paper E focuses on providing such potential groups of visitors. The formal problem statement is defined as [18]: *Given a set of categories and a parameter k, find the top-k biggest groups of users (cohorts) such that they are friends and their potential to visit the locations of the given categories is maximal.*



Users	Locations	Time
a,b,c,d	R_1	t_1
a,b,d	T_1	t_2
a,c,d	R_2	t_3
a,c,d	T_2	t_5
a,b,d	B_1	t_6
a,b,d	C_1	t_7
a,b	R_1	t_8
a,b	T_1	t_9

Fig. 14: Example location history (L) and check-ins (R) [18]

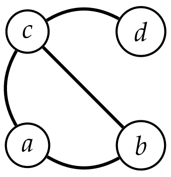


Fig. 15: Social Graph (*undirected*) for running example [18]

7.2 Cohort Discovery in LBSNs (COVER)

In order to find such cohorts, we performed detailed LBSN analysis to find the features that affect the mobility of users in groups. We observe that more than 75% of times, cohorts consist of clique friends, where all users are friends. Usually, users like to join different friends based on the nature/category of the location. Further, users tend to perform multiple activities with different friends than those with whom they like to perform single activities. Based on these observations, we define the following two features of a cohort that its users should: 1) form a clique in the social graph and 2) have been significantly performing activities together in the past.

Example 7.1
For instance, consider the example given in Figure 14 and Figure 15. In this example, we aim to find the group of users who would like to visit "bar" and "cinema" together. In this case, we would choose {a,b} as they have been performing activities together on the given categories as well as on other locations in the past. Thus, it can be induced that they like to perform activities together. Moreover, they are also friends in the social network.

For the first feature, we exploit the social graph $G(V, E)$ where V is the set of users and E is the set of friendship relationships between the users. For the second feature, we need to process the historic visit logs for finding the users who have been performing activities together in the past. To do that, we define an activity graph ($G_A^{\mathcal{L}}$) over the set of users V where the edge weight between a pair of users $w(u, v)$ is given by the following definition.

Definition 0.6. *The* **edge weight** w_{uv} *between the pair of users* (u, v) *in* $G_A^{\mathcal{L}}$ *is defined via the (consecutive) spatiotemporal join* $\bigcup_{cat \in \mathcal{L}} \{\mathcal{A}(u, cat) \bowtie \mathcal{A}(v, cat)\}$*, normalized by dividing by the highest value obtained among all pairs of users, as follows [18]:*

$$w_{uv} = \frac{|\bigcup_{cat \in \mathcal{L}} \{\mathcal{A}(u, cat) \bowtie^{(c)} \mathcal{A}(v, cat)\}|}{max_{x,y \in V} |\bigcup_{cat \in \mathcal{L}} \{\mathcal{A}(x, cat) \bowtie^{(c)} \mathcal{A}(y, cat)\}|} \tag{7}$$

where $\mathcal{A}(v, cat)$ show the set of activities v performed over locations of category cat.

For cohorts, we are interested to find the users who form cliques in the social graph and have high density in the activity graph, i.e., have been performing many activities with each other in the past. Moreover, cohorts with a larger number of users are preferred. We assume that such groups of users, will behave similarly in the future and thus, can be predicted accordingly. In order to find the trade-off between the high density and the size of cohorts, we find the optimal quasi-cliques (OQC) [19] on the activity graph. We show that finding such OQC on the activity graph is NP-hard.

Next, to find such top-k cohorts, we provide an algorithm called *COVER*. COVER, first fetches the maximal cliques from the social graph and then evaluate them for finding the OQC based on their edge weights in the activity graph. COVER provides a heuristic based solution to avoid computational overhead and to prune the maximal user cliques groups which do not have potential to emerge in top-k cohorts.

7.3 Discussion

In order to evaluate our proposed approach, we utilize three real-world LBSN datasets; Foursquare, Wee, and Gowalla. For comparisons, we provide a baseline approach (BF) in which we mine the cohorts using a brute force way and two variants of a state-of-the-art approach, Group Finder [20], i.e., GF-PAV and GF-PLM. We sort the dataset in the ascending order of the visiting time and divide it into the training data and the test data such that both have an equal number of check-ins. A group of users having at least two users who have visited at least two locations together consecutively is considered a cohort. Such cohorts are fetched using the BF approach from the test data and used as ground truth. We use COVER and other competitors to find

	Accuracy					P@K					MAP					NDCG				
K	**BF**	**GF PAV**	**GF PLM**	**OQC**	**COVER**	**BF**	**GF-PAV**	**GF-PLM**	**OQC**	**COVER**	**BF**	**GF-PAV**	**GF-PLM**	**OQC**	**COVER**	**BF**	**GF-PAV**	**GF-PLM**	**OQC**	**COVER**
1	0.5	0.01	0.01	0.25	**0.6**	0	0	0	0	0	0	0	0	0	0	**0.5**	0.09	0.1	0.25	0.35
5	**0.7**	0.03	0.03	0.1	0.4	**0.35**	0	0	0	0.15	**0.18**	0	0	0	0.17	0.7	0.25	0.1	0.5	**0.76**
10	0.6	0.03	0.03	0.05	**0.63**	**0.22**	0	0	0	0.15	**0.22**	0	0	0	0.16	0.79	0.25	0.15	0.5	**0.81**
15	0.5	0.08	0.05	0.03	**0.57**	**0.18**	0	0	0	0.15	**0.22**	0	0	0	0.16	0.81	0.1	0.25	0.5	**0.81**
20	0.47	0.08	0.05	0.025	**0.53**	0.14	0	0	0.01	**0.14**	**0.2**	0	0	0	0.16	0.8	0.25	0.25	0.5	**0.82**

Table 8: Comparative Evaluation on Wee ($\alpha = 0.4$ for COVER, best values for OQC) [18].

cohorts from the training dataset. If a predicted cohort exists in the ground truth, i.e., cohorts fetched by the BF from the test dataset, we consider it a correct prediction. We evaluate the performance based on following four measures: accuracy, P@K, mean average precision (MAP), and normalized discounted cumulative gain (NDCG). We define the accuracy by the following equation [18].

$$Acc = \frac{|C \cup T|}{\min\{|C|, |T|\}} \tag{8}$$

where C and T represents the set of fetched cohorts by COVER and the set of cohorts in the test data, respectively. The results show that BF outperforms COVER for P@K and MAP in few cases, but takes 3 orders of magnitude more computation time. This makes BF an infeasible solution. However, COVER outperforms the variants of group-finder for all the measures and BF for accuracy and NDCG as shown in Table 8.

8 Summary of Contributions

The thesis provides a general LBSN data model with a focus on the efficient and scalable processing of existing and novel location analytics queries. The thesis is composed of 7 papers. A summary of contributions for each paper is given below.

- *Paper A* [11] introduces the concept of location graph based on interactions of locations. Such a graph is used to provide an LBSN data storage and processing model. The proposed model divides the LBSN data into three graphs; the social graph, the activity graph and the location graph. A set of query primitives is defined for each graph for processing of advanced queries. Moreover, a general and scalable framework, GeoSocial-GraphX is proposed for efficient and scalable processing of these queries using the proposed data model. The experimental evaluation shows that the proposed model and framework are effective, efficient and scalable, and outperform the existing approaches and frameworks for location analytics queries.

- *Paper G* [21] exploits a location graph based on visitors of locations and shows that the mobility data can be used to improve the information propagation and influence maximization mechanism. Based on this observation, *Paper F* [13] introduces a notion of location influence that defines the influence of locations among each other by the ability of a location to spread its visitors, called bridging visitors to the other locations. Such influence can be used to spread a message to the maximum geographical regions and is thus useful for applications such as outdoor marketing, simulation of virus/news propagations. The location influence models are defined for determining the location influence. To measure such influence an accurate algorithm and an approximate but efficient algorithm are provided. The experimental evaluation shows the significance of our proposed notion and the efficiency and scalability of our algorithms.

- *Paper B* [14] extends the Paper F to provide efficient and effective location influence models. For effectiveness, influenced friends of the bridging visitors are considered and for efficiency, single influencer based influence models are proposed. Moreover, an extensive quantitative evaluation to compare the proposed models with the existing location influence models and state-of-the-art approaches is provided.

- *Paper C* [15] provides a unified model for evaluating information propagations and influence maximization mechanisms in LBSNs. The model captures four types of information propagations based on corresponding types of interactions of locations and users. A system called IMaxer is provided that implements the unified model. IMaxer provides four modules that support all steps required in influence maximization; Data Selection, Data preparation and transformation, Influential nodes mining, and Spread estimation. IMaxer is further enriched with several state-of-the-art approaches for influence maximization and spread estimation.

- *Paper D* [16] exploits the subjective nature of locations to provide a model called CMViP for predicting visitors at a given location and at a particular time. This paper identifies the features which affect the mobility of users and based on these features computes the potential of a user to visit a particular location. To do that, a non-negative collective matrix factorization approach in combination with an influence maximization method is provided. The experimental evaluation shows that CMViP outperforms the competitor in both precision and recall.

- *Paper E* [18] provides a method for predicting top-k groups of users (cohort) that will visit a set of given locations together. To do that, we proposed a heuristic algorithm called COVER. COVER enumerates cliques from the social graph and finds optimal quasi-cliques from the activity graph which is composed of the mutual historic activities of users. The experimental evaluation shows that COVER outperforms brute force based methods and different variants of state-of-the-art approaches.

9 Future Work Directions

In future work, we aim to improve the expressiveness of the LBSN data model such that it can support any relational algebra query. In order to achieve this, more query primitives for the LBSN segregated graphs: the social graph, the activity graph, and the location graph should be provided such as group by query primitives. Novel query primitives should be introduced to support state-of-the-art queries such as trajectory/flow-based queries. Moreover, interactions of locations based on other features such as common visitors, and trajectories should be considered for the location graphs. Location graphs based on these features should be used to exploit LBSN data for further optimizing LBSN applications.

Other LBSN data models should be considered such as aggregated graphs and graph cubes based LBSN data models. For instance, a heterogeneous graph consisting of user and location nodes and their diverse types of interactions with each other can be used for modelling LBSN data. Such an LBSN data model can be used for finding communities based on the heterogeneous information such as friends of users, their travel history, and relationships of locations they visit. Similarly, in another LBSN data model, we can materialize friendship, visit history, and location interactions based information in temporal and spatial graph based cubes. Such an LBSN data model can be used for providing trend aware queries.

In order to provide an efficient implementation of these LBSN data models, extension of suitable data processing frameworks should be considered. Relational algebra and calculus should be designed for the primitive queries of these LBSN data models and corresponding query languages should be introduced. Further, based on the LBSN data model and underlying frameworks, advanced data processing and query optimization methods should be proposed. For GeoSocial-GraphX (GSG), Spark SQL should be used for providing a query language for processing queries. Furthermore, Spark's latest APIs should be used to further optimize the indexing and partitioning mechanisms.

The location influence maximization studies focus on finding top-k seeds that can be used for maximizing the influence. Nevertheless, in real life, the locations for displaying advertisements or promotional messages, i.e., billboards are fixed and already known in advance. Thus, we should focus on optimizing the propagations methods based on given seed nodes. The locations are usually visited by the visitors of diverse interests. Further, users are attracted by the contents of their interest. Thus, we should consider the interest of visitors of a location for designing advertisements/promotional messages to be displayed at the seed locations for maximizing influence. Based on this corresponding context-aware location influence models should be proposed that can capture such context-aware location influence. Furthermore, different information propagations models should be considered for evaluating location influence such as independent cascade and linear threshold models. Further, distributed solutions for implementing these location influence models should be proposed based on the proposed query primitives.

For the visitor predictions, more features should be considered for optimizing predictions, such as days of week or weekend, and popularity of the locations. The state-of-the-art machine learning methods such as neural network and deep learning based approaches should also be used for improving effectiveness. Moreover, distributed and parallel processing of query primitives based algorithms should be provided for efficient processing.

References

[1] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[2] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "Recommendations in location-based social networks: a survey," *GeoInformatica*, vol. 19, no. 3, pp. 525–565, 2015.

[3] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," in *PVLDB*, 2013, pp. 913–924.

[4] L. Del Prete and L. Capra, "differs: A mobile recommender service," in *MDM*, 2010, pp. 21–26.

[5] Z. Yin, L. Cao, J. Han, C. Zhai, and T. Huang, "Geographical topic discovery and comparison," in *WWW*, 2011, pp. 247–256.

[6] M. Sarwat, A. Eldawy, M. F. Mokbel, and J. Riedl, "Plutus: Leveraging location-based social networks to recommend potential customers to venues," in *MDM*, 2013, pp. 26–35.

[7] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Who, where, when and what: Discover spatio-temporal topics for twitter users," in *KDD*, 2013, pp. 605–613.

[8] "Spark," https://spark.apache.org/.

[9] "Spatialhadoop," http://spatialhadoop.cs.umn.edu/.

[10] "Graphlab," https://graphlab.org.

[11] M. A. Saleem, X. Xie, and T. B. Pedersen, "Scalable processing of location-based social networking queries," in *MDM*, 2016, pp. 132–141.

[12] "Outdoor advertising association of america," https://oaaa.org/StayConnected/NewsArticles/IndustryRevenue.aspx.

[13] M. A. Saleem, R. Kumar, T. Calders, T. B. Pedersen, and X. Xie, "Location influence in location-based social networks," in *WSDM*, 2017, pp. 621–630.

[14] M. A. Saleem, R. Kumar, T. Calders, and T. B. Pedersen, "Effective and efficient location influence mining in location-based social networks," in *In Journal of Knowledge and Information Systems (Submitted)*, 2017.

[15] M. A. Saleem, R. Kumar, T. Calders, X. Xie, and T. B. Pedersen, "Imaxer: A unified system for evaluating influence maximization in location-based social networks," in *CIKM*, 2017, pp. 2523–2526.

[16] M. A. Saleem, F. Costa, P. Dolog, P. Karras, T. Calders, and T. B. Pedersen, "Predicting visitors using location-based social networks," in *MDM (Submitted)*, 2018.

[17] S. Feng, G. Cong, B. An, and Y. M. Chee, "Poi2vec: Geographical latent representation for predicting future visitors," in *AAAI*, 2017, pp. 102–108.

[18] M. A. Saleem, P. Karras, T. Calders, and T. B. Pedersen, "Mining geo-social cohorts in location-based social networks," in *KDD*, 2018.

[19] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli, "Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees," in *KDD*, 2013, pp. 104–112.

[20] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso, "Group finder: An item-driven group formation framework," in *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, vol. 1, 2016, pp. 8–17.

[21] R. Kumar, M. A. Saleem, T. Calders, X. Xie, and T. B. Pedersen, "Activity-driven influence maximization in social networks," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part III*, 2017, pp. 345–348.

Part II

Papers

Paper A

Scalable Processing of Location-Based Social Networking Queries

Muhammad Aamir Saleem, Xike Xie, Torben Bach Pedersen

The paper has been published in the
17th IEEE International Conference on Mobile Data Management (MDM '16),
pp.132–141, 2016.

Abstract

Using GPS-enabled smart phones, social network services are enriched with location information which allows users to share geo-tagged contents with their friends. This so-called location-based social network (LBSN) data has a dual spatial and graph nature. The growing scale and importance of LBSN data necessitate a platform which (i) has both spatial and graph capabilities; (ii) supports a wide range of queries, e.g., selection, structural, and aggregate queries; (iii) supports scalable distributed processing of large data volumes.

In this paper, we propose such a platform, called GeoSocial-GraphX, that segregates the LBSN data into several specific graphs capturing user-user, user-location, and location-location relationships, and enables a wide range of LBSN queries by proposing a comprehensive set of query primitives that can be composed into more advanced queries. We implement the platform based on GraphX, a map-reduce infrastructure for distributed graph computation. We further improve the query performance in several ways. For social-related data, we use vertex-centric messaging operators which better address the recursive nature of graph data than traditional two-stage map-reduce. For spatial-related data, we use effective spatial partitioning and indexing methods. Experiments on both synthetic and real LBSN datasets show that GeoSocial-GraphX can process a variety of LBSN queries efficiently, scales on multicore architectures, and achieves much better performance than the state of the art competing framework, SpatialHadoop.

1 Introduction

With the ubiquity of location-aware mobile terminals, social network users are allowed to share geo-tagged contents with their friends. These social networks with location information are called location-based social networks (LBSN). For example, users make comments on Twitter about an event happening somewhere, upload geo-tagged pictures in Flickr, or report their "check-ins" in Foursquare. LBSN data offers richer interdependency between people and locations thus holds potential for a wide range of services.

For example, the context of a location can be deciphered by its historical visitors; the mobility behaviour of a user can be investigated by his/er historical visits. It is interesting to query for top-k popular places for a given user, and top-k frequent travellers for a set of places. Such information leverages ranking and recommendation in order to facilitate travel and social interactions [1–3]. In another example, one might need to find nearby friends for an activity, or detect a community in a local area. Similarly, one might also be interested to find a set of locations that are famous for visiting together. Such applications require considering structural information in LBSN.

Another challenge arises in the recent explosion of the amounts of LBSN data. As reported in [4], Foursquare has more than 55 million users with 6 billion check-ins. The map-reduce framework, e.g., Hadoop, is generally accepted as a solution for scalable processing of large volume datasets. However, it falls short in fully addressing the challenges in LBSN data which is of both spatial and graph nature. First, the data partitioning method used in HDFS does not consider the recursive nature of graph data and consecutive nature of spatial data. Thus, it does not efficiently support multiple random data accessing which is essential for spatial or graph queries. Second, the two-stage computation framework has limited capabilities in handling iterative algorithms required by graph (or structural) queries in LBSN.

In particular, SpatialHadoop [5] , MongoDB [6], and MD-base are map-reduce frameworks equipped with spatial features. GraphX, which is built on top of *Apache Spark* is a data-parallel computation framework supporting big graph data. None of them have capabilities to support big data of both spatial and graph nature. It is thus desirable to have a platform which embeds both spatial and graph capabilities and is able to process large volume LBSN queries that are diverse in nature.

Motivated by these challenges, this paper makes the following contributions. We consider the two major roles, *user* and *location*, in LBSN. These roles formulate three types of relationships, *user-user*, *user-location*, and *location-location*. In response to such characteristics, we segregate the LBSN data into three graphs, namely *social graph*, *activity graph*, and *spatial graph*. We present a comprehensive set of query primitives defined on these graphs. The LBSN query primitives are classified into three categories: *selection*, *aggregate*, and *structural* queries, such that general-purpose LBSN queries can be answered by the combination of one or more primitives. For example, we can retrieve a local community by combining results of a structural query on the social graph and a selection query on the spatial graph. Furthermore, we present a platform that: (i) is capable of handling large volume and distributed LBSN data; (ii) answers general purpose LBSN queries. The first objective of the platform is reached by building a data-parallel platform, GeoSocial-GraphX (GSG *in short*). Based on that, we incorporate indexing and partitioning techniques for supporting spatial data. The second objective of the platform is reached by implementing a number of query primitives which are essential for LBSN queries on top of the platform. In order to evaluate our system, we perform experiments on three real LBSNs namely, BrightKite, Gowalla and FourSquare and one synthetic dataset. The experiments show that segregation of LBSN into three graphs and usage of the spatial graph for location-centric queries improve the performance up to four orders of magnitude. Furthermore, the partitioning and indexing methods for spatial data enhance the performance by 6 times on average. Experimental results also confirm the scalability of GSG on multiple cores. Moreover, GSG outperforms SpatialHadoop, a leading spatial distributed framework, in terms of efficiency, scalability, and ease of implementation, i.e., lines of code, up to 20x, 7x, and 13x, respectively.

The rest of the paper is organized as follows. Section 2 provides the LBSN model including definitions of the basic roles in LBSN and the segregated graphs. Section 3 presents the query primitives and advanced LBSN queries. Section 4 covers the architecture of the proposed platform, GSG, including storage layer, operation layer, index layer, and query engine. Section 5 presents the experimental evaluation. Section 6 presents related work, and Section 7 concludes the paper and points to future work.

2 LBSN Model

We present basic concepts in Section 2.1 and define the three segregated graphs in Section 2.2.

2.1 Preliminaries

Definition A.1. *Users(U) is the set of persons that are using the LBSN. A user is identified by the unique identifier u_i. In Figure A.1, users are $U = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$.*

Definition A.2. *Locations(L) is the set of locations. We consider a location as a circular region that covers a geographical space. A location is a four-tuple (l_m, x_m, y_m, r_m),*

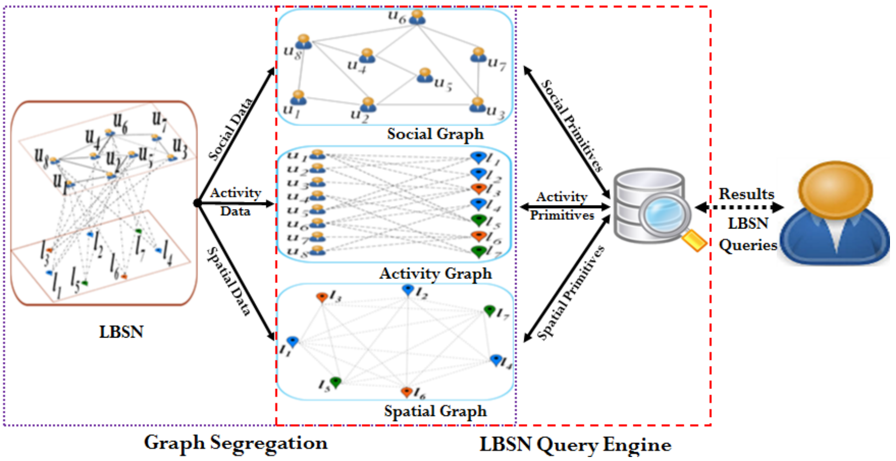


Fig. A.1: LBSN Model

where the l_m is the identifier, x_m and y_m represent the center of the region, and r_m is the radius of the region. In Figure A.1, locations are $L = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7\}$.

Definition A.3. *Activities(A) is the set of activities of the users. An activity is a checkin of a user including both location and time. It is given by (u_i, l_m, t_a), where u_i is the user identifier, l_m is the location identifier, and t_a is the visiting time. In Figure A.1, activities are shown by dotted lines, e.g., u_1 has visited l_1, l_2, l_3, and l_5 at sometime.*

2.2 LBSN Graphs

Traditionally, LBSN data is maintained into two components on the basis of its roles: user and location. The components are termed as social component and activity component. The social component represents relationships among users, i.e., user-user. The activity component deals with users' visits at locations and captures user-location relationships. However, a location is a subject in the LBSN that has attributes such as GPS coordinates. Furthermore, the interactions of LBSN roles in the social and activity component yield the relationships between locations, i.e., location-location, such as common visitors and distances among locations.

For LBSN queries about user-user and user-location relationships, the social component and the activity component can be used, respectively. However, for queries about location-location relationships, it is very expensive to use the social and the activity components. An example of such queries is to find a user's best travel companions based on their common visited places. In order to process such queries in an efficient and a scalable way, it is beneficial to maintain a component that represents location-location relationships in LBSN which directly handle location oriented queries and recommendations. In this work, such a component is called *spatial graph*.

We segregate the LBSN data into three graphs, i.e., social graph, activity graph, and spatial graph as shown in Figure A.1. Details of these graphs are given below.

Social relationships between users are categorized into two types: directed and undirected. For example, if users are friends on Facebook their relationship is undirected. However, Twitter users are either followers or followees, therefore, their relationship is directed. For the sake of simplicity, this study considers undirected relationships between users. However, directed relationships can be treated similarly. The social graph stores the relationships of users in the form of a graph as shown in Figure A.1, where u_1 is a friend of u_2 and u_8.

Definition A.4. *SocialGraph*(G_S) *is an undirected graph, where vertices represent users and edges represent relationships/ friendship between them. It is given by,* $G_S = (U, F)$*, where U is the set of vertices that represents users, and F is the set of edges representing connections between them, i.e.,* $F \subseteq \{U \times U\}$.

Activities of users, i.e., visits/interactions of users at locations, is the essence of LBSN. The activity graph maintains this information in the form of a bipartite graph as shown in Figure A.1, where u_1 visits l_1, l_2, l_3, and l_5.

Definition A.5. *ActivityGraph*(G_A) *consists of users and their Activities. It is given by* $G_A = (U, L, A)$*, where U and L are the two sets of vertices: users and locations, respectively. A is the set of edges which represents activities of users U at locations L at times T. It is given by* $A \subseteq \{U \times L \times T\}$.

Locations are related to each other in several ways on the basis of their properties, e.g., distances, trajectories, and common visitors. In this study, we consider common visitors for building relationships between locations [1]. The definition of common visitors is given below.

Definition A.6. *CommonVisitors*$(U_c(l_m, l_n))$ *is the set of users who visited the locations* l_m *and* l_n*. It is given by:* $U_c(l_m, l_n) = \{u \in U | (u, l_m, t_a), (u, l_n, t_b) \in A\}$*, where u is a user that visited both locations* l_m *and* l_n*, and* t_a *and* t_b *are visit times.*

The locations l_m and l_n that have at least one[2] common visitor are called *LinkedLocations* and are represented as $l_{lk}(l_m, l_n)$. Next, the spatial graph defined on the basis of common visitors is presented.

[1]We are aware of other methods to associate locations to each other, e.g., [7]. Our spatial graph extensions can support their applications with simple alterations.

[2] A threshold describing the minimum number of common visitors among locations to be considered as the LinkedLocations, can be utilized to control the density of the spatial graph. For example, for FourSquare dataset [8] by setting the threshold to 3, the graph density is reduced to $1.02x10^-6$.

Definition A.7. *The spatial graph represents the locations as vertices and the relationships between each other as edges. It is given by $G_L = (L, E)$, where L is the set of vertices and E represents the set of edges, i.e., $E \subseteq \{L \times L\}$. An edge is a pair of LinkedLocations having common visitors.*

3 LBSN Queries

This section presents the LBSN queries. We define query primitives in Section 3.1 and advanced queries in Section 3.2.

3.1 Query Primitives

Query primitives are defined as fundamental operations for processing of LBSN queries. Below, we define a comprehensive set of such query primitives for each type of graph, in a way that each segregated graph is sufficient to independently process its primitive queries. The primitives of each graph are grouped based on the type of processing performed into selection, structural, and aggregate queries, as shown in Table A.1. The primitive queries can further be combined to answer a wide range of general purpose LBSN queries.

The first group contains *social query primitives* that exploit the relationships among users over the social graph. Among them, FF is a selection query; FSoSD and FCF are structural queries; TCU is an aggregate query.

- $FindFriends(u_i, \sigma)$: Given a user u_i and a social separation degree σ, return the set of users that require minimum σ steps to connect u_i.
- $FindSocialSeparationDegree(u_i, u_j)$: Given the users u_i and u_j, return the social separation degree, i.e., minimum number of hops required for connecting them.
- $FindCliqueFriends(u_i)$: Given a user u_i, return the set of users that are friends with u_i and as well as with each other.
- $Top - k\ ConnectedUsers(k, \sigma)$: Given a value of separation degree σ and a parameter k, return $top - k$ users based on their maximum number of connections with users in social separation degree of σ.

In the second group, we define *activity query primitives* that exploit the user and location interactions over the activity graph. Here, FUL and FLV are selection queries; FRL and FSpSD are structural queries; TNL, TV and TVL are aggregate queries.

- $FindUserLocation(u_i, t_a, t_b)$: Given a user u_i and a time interval $[t_a, t_b]$, return the set of locations that u_i visited between t_a and t_b.

Table A.1: LBSN query primitives

Graphs	Categories	Primitives	Notations
G_S	Selection	FindFriends	FF
	Structural	FindSocialSeparationDegree FindCliqueFriends	FSoSD FCF
	Aggregate	Top-k ConnectedUsers	TCU
G_A	Selection	FindUserLocation FindLocationVisitors	FUL FLV
	Structural	FindRangeLocations FindSpatialSeparationDegree	FRL FSpSD
	Aggregate	Top-k NearestLocations Top-k Visitors Top-k VisitedLocations	TNL TV TVL
G_L	Selection	FindCommonVisitors FindLinkedLocations	FCV FLL
	Structural	Find LocationSeparationDegree	FLSD
	Aggregate	Top-k VisitedLinkedLocations Top-k ConnectedLinkedLocations	TVLL TCLL

- $FindLocationVisitors(l_m, t_a, t_b)$: Given a location l_m, and a time interval $[t_a, t_b]$, return the set of users that visited l_m between t_a and t_b.
- $FindRangeLocations(l_m, d_r)$: Given a location l_m and a radius d_r, return the set of locations within the circular region centered at l_m with radius d_r.
- $FindSpatialSeparationDegree(u_i, u_j, t_a, t_b)$: Given the users u_i and u_j, and a time interval $[t_a, t_b]$, return the minimum number of hops required for connecting u_i with u_j based on their visited locations.
- $Top - k\ NearestLocations(l_m, k)$: Given a location l_m, and a parameter k, return the set of k locations in the ascending order of distances from l_m.
- $Top - k\ Visitors(k, t_a, t_b)$: Given a parameter k, and a time interval $[t_a, t_b]$, return $top - k$ users based on their maximum number of activities during t_a and t_b.
- $Top - k\ VisitedLocations(k, t_a, t_b)$: Given a parameter k, and a time interval $[t_a, t_b]$, return $top - k$ locations with maximum number of visits between t_a and t_b.

The third group defines the *spatial query primitives* that exploit the relationships among locations, i.e., spatial graph. Here, FCV and FLL are selection queries; FLSD is a structural query; TVLL and TCLL are aggregate queries.

- $FindCommonVisitors(l_m, l_n)$: Given the locations l_m and l_n, return the common visitors of l_m and l_n, if there are any.
- $FindLinkedLocations(l_m, \lambda)$: Given a location l_m and a degree of separation λ , return the set of locations that require no more than λ steps to connect l_m.
- $FindLocationSeperationDegree(l_m, l_n)$: Given the locations l_m and l_n, return the minimum number of steps required for connecting them.
- $Top-k\ VisitedLinkedLocations(l_m, k)$: Given a location l_m, a set of its $LinkedLocations$ and a parameter k, return $top-k\ LinkedLocations$ of l_m based on their maximum number of $CommonVisitors$.
- $Top-k\ ConnectedLinkedLocations(k)$: Given a parameter k, return $top-k$ locations based on their maximum vertex degree, i.e., connections with other locations.

3.2 Advanced LBSN Queries

Advanced LBSN queries utilize the combinations of the query primitives for processing. These queries are divided into four groups on the basis of these primitives, i.e., the queries that are composed of social and activity, social and spatial, activity and spatial, and social, activity and spatial query primitives, respectively. Due to limited space, we provide only two groups of these queries.

In the first group, we define the queries that combine the social and the activity query primitives.

- $RangeFriends(u_i, \sigma, l_m, t_a, t_b, d_r)$: Given a user u_i, a social separation degree σ, a location l_m, a time interval $[t_a, t_b]$ and a radius d_r, return the set of friends of u_i in social separation degree of σ whose locations during $[t_a, t_b]$ are within the circular region centered at l_m with radius d_r.
- $Top-k\ NearestFriends(u_i, \sigma, l_m, t_a, t_b, k, U, A,)$: Given a user u_i, a social separation degree σ, a location l_m, a time interval $[t_a, t_b]$ and a parameter k, return the set of k users that are friends of u_i in social separation degree of σ, in the ascending order of distances between their current locations and l_m during $[t_a, t_b]$.

- $Top - k\ NearestStarGroup(\omega, l_m, t_a, t_b, k)$: Given a number of users ω, a location l_m, a time interval $[t_a, t_b]$ and a parameter k, return the k groups of ω users each, in ascending order of aggregate distance of the user's locations among each other and with l_m during $[t_a, t_b]$ such that users of each group are connected by at least one common friend [9].

In the second group, we define the advanced LBSN queries that combine the social and the spatial query primitives.

- $Top - k\ EndorsedLinkedLocation(u_i, l_m)$: Given a user u_i, a location l_m and parameter k, return the k $LinkedLocations$ of l_m that have maximum number of the user's friends as $Common\ Visitors$ in between.

- $Top - k\ CommonVisitorFriends(U)$: Given a user u_i, return the k friends of u_i based on their maximum number of appearances as $CommonVisitors$ with u_i.

4 GSG Architecture

GSG is a graph-parallel platform that supports distributed processing of LBSN queries on large volume data. The architecture of GSG is composed of four parts, i.e., storage layer, operation layer, index layer and query engine, as shown in Figure A.2.

4.1 Storage layer

This layer is responsible for providing the data structures and mechanisms for storage of LBSN graphs in a distributed way.

The underlying framework of GSG, GraphX extends the Spark's [10] distributed data scheme, RDD [11]. RDD is an immutable, persistent and parallel data structure that supports distributed operations on it. In GraphX, VertexRDD and EdgeRDD [12] extend RDD to support augmented attributes for vertices and edges, respectively, and are abstractly combined to construct the graphs. GSG utilizes this data scheme for storage of the LBSN graphs.

In LBSN graphs, there are two types of vertices: users and locations. A user vertex maintains a user identifier and a vertex type. A location vertex keeps the location id as the identifier and corresponding location information (e.g., latitude, longitude, and radius) as attributes. Furthermore, the LBSN graphs posses three types of edges with their corresponding unique attributes. An edge of the social graph maintains the ids of users as source and destination vertices, and their corresponding relationship, i.e., friends as an attribute. An edge of the activity graph is composed of ids of a user as a source vertex, a location as a destination vertex, and visit time of user at the location as an edge attribute. An edge of the spatial graph consists of

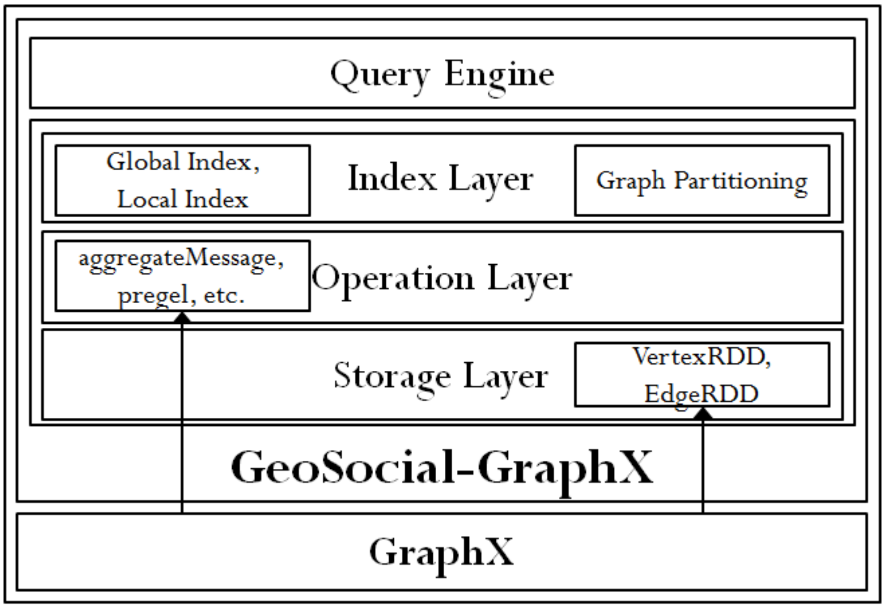
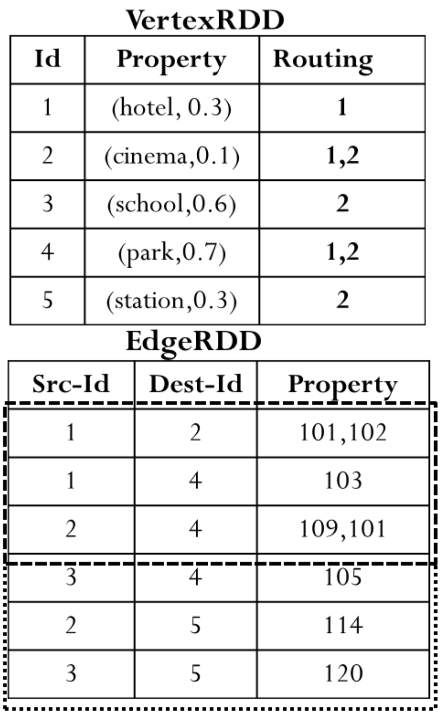


Fig. A.2: Architecture of GeoSocial-GrapX

location ids as source and destination vertices, and ids of common visitors as edge attribute. Figure A.3 (a) shows an example of the storage of the spatial graph.

In order to store the graphs, both VertexRDDs and EdgeRDDs are partitioned and distributed among nodes, where a node represents a computational unit (e.g., a computer). The vertex cut approach is utilized to partition the graph. This approach helps to reduce more communication and storage overhead as compared to the edge cutting approach [13]. Vertices are kept on the same node with their connecting edges. Figure A.3 (b) shows an example of the distributed spatial graph. The graph is split into two partitions, each of which corresponds to a node. Here, vertices 2 and 4 are cut in order to split the graph. These vertices are replicated and stored on both nodes. Next, we provide the algorithm for the construction of the spatial graph. The construction for the social and activity graphs are similar. They are thus omitted due to page limits.

In Algorithm A.1, users' check-ins data are utilized to extract two kinds of information, i.e., check-ins that contain the visits of users at their corresponding locations (line 6) and vertices, i.e., locations (line 8). The users are grouped on the basis of their visited locations to extract edge attributes, i.e., common visitors (line 12). These groups are then used to create the list of edges (lines 14-21). The extraction of check-ins, vertices, and grouping of



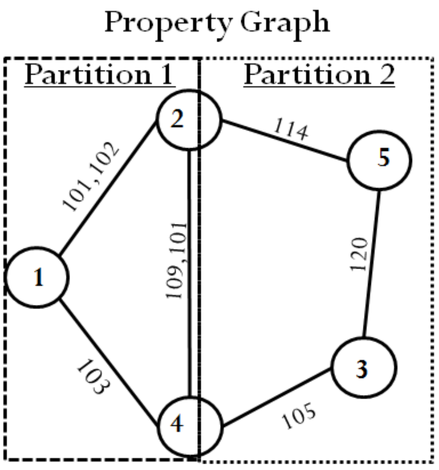
VertexRDD

Id	Property	Routing
1	(hotel, 0.3)	**1**
2	(cinema,0.1)	**1,2**
3	(school,0.6)	**2**
4	(park,0.7)	**1,2**
5	(station,0.3)	**2**

EdgeRDD

Src-Id	Dest-Id	Property
1	2	101,102
1	4	103
2	4	109,101
3	4	105
2	5	114
3	5	120

(a) vertexRDD and EdgeRDD



(b) property graph

Fig. A.3: Data storage for the spatial graph

users are performed by map-reduce operations. However, the construction of edges utilizes concurrent programming to exploit maximum computational resources in the cluster. We use scala based concurrent programming model: akka [14] for parallel operations, i.e., nested for loops (lines 14-21 and 16-20). Further, the graph is constructed by utilizing vertices and edges (line 22).

4.2 Operation layer

The role of this component is to provide the operators that are applied on the graphs for processing of LBSN queries. These operators are grouped into two types, i.e., core operators such as *mapEdges, triplets, and aggregateMessage*, and advanced operators such as *pregel, and connectedComponents*. The core operators transform the structures and properties of input graphs on the basis of user-defined functions. The advanced operators are the optimized variants of core operators that provide advanced graph algorithms. We selectively discuss two core and one advanced operators which we believe best reflect the nature of the operators of corresponding groups.

Algorithm A.1: ConstructSpatialGraph(UserCheck-ins): G_L

```
1  begin
2      $Check - ins = \emptyset$
3      $edges = \emptyset$
4      $vertices = \emptyset$
5      foreach $(u_1, loc_1) \in UserCheck - ins$ do
                                   // $loc_1$ contains the attributes of location
6          $Check - ins+ = (u_1, loc_1)$
7          if $l_1 \notin Location$ then
8              $vertices+ = new\ Location(loc_1, "location")$
9          end
10     end
11     foreach $loc_1 \in vertices$ do
12         $UsersGroupByLocation+ = Check - ins.groupBy(loc_1)$
             // provide visitors of each location
13     end
14     parallel foreach $(loc_1, U_1) \in UsersGroupByLocation\ AND$
       $count <= size(UsersGroupsByLocations)/2$ do
15         $count++$
16         parallel foreach $(loc_2, U_2) \in UsersGroupByLocation$ do
17             if $loc_1 \neq loc_2$ then
18                 $edges+ = Edge(loc_1, loc_2, (U_1 \cap U_2))$
19             end
20         end
21     end
22     $G_L = createGraph(vertices, edges)$
23     return spatial graph
24 end
```

- **triplet** is a core graph operator that combines edges along with attributes of their neighbouring vertices.
- **aggregateMessage** is a core aggregate operator that aggregates values from neighbouring edges and vertices of each vertex.
- **pregel** is a bulk-synchronous parallel graph operator that recursively use a vertex-centric approach for traversing the graph.

triplet logically combines VertexRDD and EdgeRDD, and provides an augmented EdgeRDD. The augmented EdgeRDD contains the attributes of edges and well as attributes of their source and destination vertices. In order to avoid confusion, we call each record of augmented EdgeRDD as *triplet tuple*. *triplet* is utilized when attributes of both vertices and edges are needed. Furthermore, it is utilized for other operations such as *aggregateMessage* and *pregel*.

aggregateMessage performs aggregate operations using local communication, i.e., messaging among neighbouring vertices. This operation is considered suitable for LBSN queries that require aggregated information of neighbouring vertices, e.g., FF and FUL. The traditional map-reduce structure falls short in addressing such local interactions that make it expensive to establish communication among vertices. On the other hand, *aggregateMessage* establishes this communication by two sub-functions: *sendMsg* and *mergeMsg*. Following example provides the usage of *aggregateMessage* for processing of a LBSN query.

Consider the query FF for all vertices of the social graph. In order to process it, first, the augmented EdgeRDD is created by using *triplet*. Then, *sendMsg* concurrently sends the ids of source vertices to their destination vertices for each triplet tuple. *mergeMsg* runs at every user vertex and combine all the messages received. Thus, every user vertex ends up with the ids of directly connected vertices, i.e., friends.

pregel utilizes vertex-centric approach similar to *aggregateMessage* to communicate among neighbouring vertices. However, it operates recursively in a series of iterative steps, called super-steps. Compared with map-reduce, the parallel communication among neighbouring vertices, and optimized storage, and un-persistence of intermediate results are utilized to efficiently process the query by the *pregel* operator.

For example, consider a query that requires to find degrees of a given user with all other users in the social graph. we use *pregel* and implement the single source shortest path algorithm to process this query in the super-step fashion. In order to process this query, initially, every vertex maintains its degree value from the source vertex that is 1 for directly connected edges and infinity for non-directly connected edges. In each super-step, vertices communicate and receive the updated values of their neighbouring vertices from the previous super-step. On the basis of this information, vertices compute the minimum value to connect the source vertex. Vertices update their values only if the newly computed value is smaller than their current values. These intermediate results are maintained in the memory in an optimized way and used for next super-step. The iterations are continued until there remain no vertices to be updated. At this moment, the values of vertices represent their degrees from the source vertex.

4.3 Index layer

Query primitives that involve spatial attributes (i.e., those on the spatial graph and the activity graph) can be further optimized by indexing techniques. With an index, the mapping phase of map-reduce operators benefits from the efficient locating of data. The indexing scheme for GSG is composed of two parts: *global index* and *local index*.

Global index partitions the data across cluster nodes. It is maintained on the master node. The purpose of the global index is to locate the edges of nearby locations on the same node.

We provide two types of global indexes: grid and k-d tree. Grid based indexes divide the geographical space into equal-sized partitions and dis-

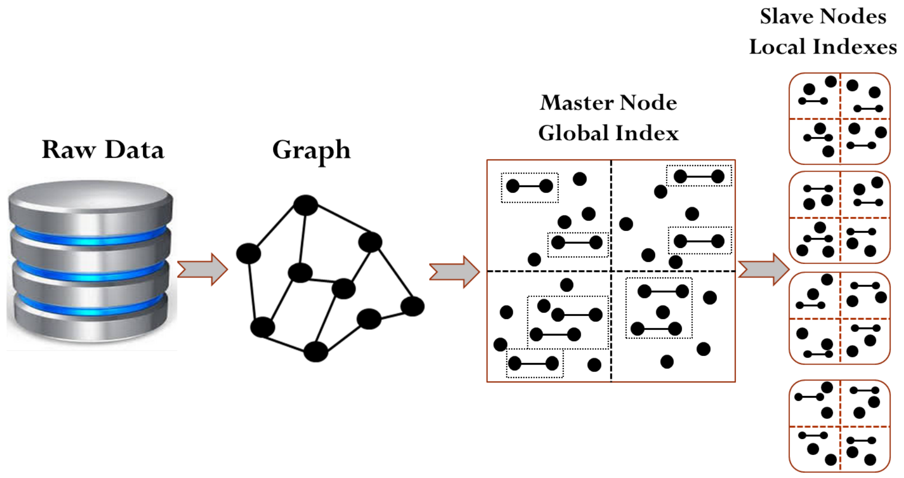


Fig. A.4: Indexing mechanism for the spatial graph: k-d tree and quadtree are used for the global and local indexes, respectively.

tribute data accordingly. k-d tree partitions the space in a more adaptive manner. Thus, it is preferred for skewed data.

The computation of the global index is performed by the following steps. (1) Default non-partitioned graph is created and augmented EdgeRDD using *triplet* is obtained. (2) Minimum bounding rectangles (MBRs) are constructed for triplet tuples of the augmented EdgeRDD based on user-defined partitioning strategy, i.e., grid or k-d tree. On the basis of these MBRs, the triplet tuples are mapped and distributed among nodes. (3) Edges are extracted from the augmented EdgeRDD on corresponding nodes. The vertices are also transferred to the nodes that store their connecting edges. (4) The memories used for storing augmented EdgeRDD and non-partitioned graph are released. (5) The k-d tree or grid is maintained at the master node as the global index.

Local index For each node, a local index to efficiently retrieve the local data is stored. We provide two indexing methods: a two-dimensional method, i.e., quadtree and a three-dimensional method, i.e., octree, for the spatial graph and the activity graph, respectively. The quadtree is used for the spatial graph because its triplet tuples have only spatial attributes: latitude and longitude. The octree is deployed for the activity graph because triplet tuples in the activity graph are of both spatial and temporal attributes. The maximum number of triplet tuples in a tree node is controlled by a given capacity. When the capacity of a tree node is reached, it splits into sub-tree nodes. It splits into 4 sub-tree nodes for a quadtree and 8 sub-tree nodes for

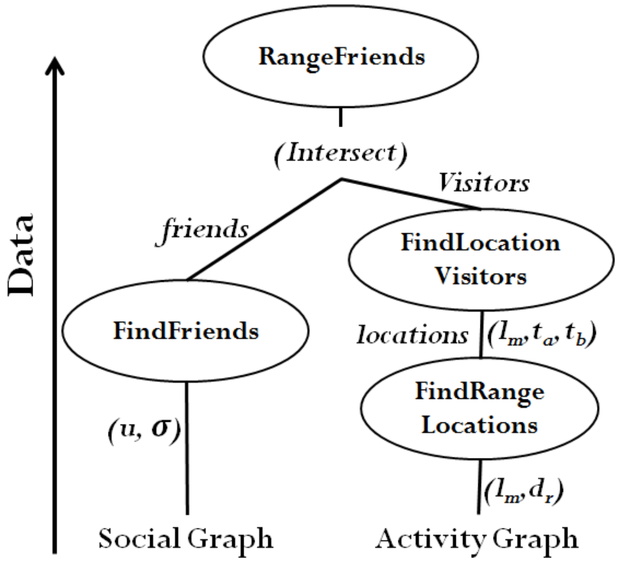


Fig. A.5: Query plan for RangeFriends$(u, \sigma, l_m, t_a, t_b, d_r)$

an octree. Thus, the height of a quadtree is more than an octree. The overall process of computation of index is shown in Figure A.4.

In order to retrieve a data item, the global index is used to find the node on which the required data resides. Then, the local index is used to efficiently fetch the required data item.

4.4 Query Engine

This component provides the implementation of all the query primitives and advanced LBSN queries mentioned in Section 3. It exposes a programmable API that provides the opportunity for building general purpose LBSN queries. In this section, we provide the mechanisms to process advanced LBSN queries with the help of a sample query plan.

Advanced queries are segregated into query primitives and processed on corresponding LBSN graphs. There are two kinds of benefits that are achieved by opting this way of query processing. First, data based optimizations are performed on corresponding graphs and utilized for efficient processing of data that can be difficult for hybrid graph structures, e.g., spatial data oriented indexing or graph partitioning strategies for the activity and spatial graph. Second, efficiency is achieved by executing basic primitives in parallel on their respective graphs. Distributed processing of a job may not require all the resources of clusters, i.e., mappers and reducers. Thus, concurrent threads utilize the available resources to efficiently process the queries.

Table A.2: Statistics of Datasets (in Million)

LBSN	Vertices		Edges		
	Users	Locations	G_S	G_A	G_L
BrightKite	0.06	0.77	0.2	4.49	0.65
Gowalla	0.2	1.28	0.95	6.44	0.15
FourSquare	0.02	0.85	0.12	2.07	0.39
Synthetic	31.18	50.97	114.64	147.34	172.02

The following example provides the processing of an advanced LBSN query: RangeFriends that is defined in Section 3.2. The overall query plan is shown in Figure A.5. The query is segregated into three primitives: FF, FRL, and FLV. FF is processed on the social graph. Simultaneously, FRL and FLV are processed on the activity graph. We exploit akka programming model [14] for concurrent programming. An independent thread is run for processing of each primitive, i.e., FF and FRL. The output of FRL is utilized for FLV. Then, the intermediate results of FF and FLV are combined by utilizing distributed operations on the cluster to get the final output.

5 Experiments

5.1 Setup

We conduct our experiments on a multi-processor machine with 4 AMD Opteron 6376 processors with 2.3GH, having 8 cores each. The machine has 512 GB RAM. To simulate the distributed environment, we create five virtual nodes with 4 cores and 100GB of RAM each. Among them, one is master and four are slaves/workers. This is used as a default setting for the cluster of GSG and SH. The OS is Ubuntu 14.04 LTS. GSG is implemented based on GraphX 1.3.1. Moreover, SpatialHadoop v2.2 with Apache Hadoop 1.2.1 is used for comparison. For concurrent programming, we exploit Akka 2.3.6. The experiments are conducted in Scala 2.10.4.

Datasets. We utilize both real and synthetic datasets. Three real LBSN datasets are used: Brightkite [15], Gowalla [15] and FourSquare [8]. The number of edges in the spatial graph is determined by the minimum number of *CommonVisitors*. Here, in order to capture all the common visitors among locations, we set its value to 1. To test the scalability of our proposals, we create a synthetic dataset of size 25GB by iteratively scaling up the graphs of the Brightkite dataset. In order to create this synthetic dataset, in each iteration, first nodes and edges are replicated then offset with new ids thus a new dataset is fetched. The vertices in both the original and replicated datasets are connected by random connections. At each iteration, 1% of the vertices in the original graph are randomly chosen to connect with those in the replicated graph. The iterations are continued until required dataset is retrieved. The details about datasets are shown in Table A.2.

5.2 Results

Scalability: The first experiment tests the scalability of query processing. Few queries from each primitive groups of the graphs, i.e., selection, aggregate, and structural query primitives, that best reflect the behavior of the corresponding groups are considered for evaluation. We evaluate the scalability of GSG by performing two types of experiments.

First, we analyze the query execution time by keeping the number of cores constant and varying the size of the dataset. The purpose is to analyze the performance of GSG when the dataset grows. Figure A.6 presents the results. Here, it is observed that the scalability for all types of queries is very close to linear. This shows that GSG scales well when data grows. All the queries show the same pattern, however, aggregate queries scale a bit less than the rest, since they require the processing of the whole graph.

Second, we fix the dataset size and vary the number of cores. The purpose of this experiment is to test how GSG scales with the amount of computational resources. The results are shown in Figure A.7. Here, as the number of cores grows from 1 to 16 cores, we can see the maximum speed-up is 9.8x. The savings in execution time are better for fewer nodes, i.e., we see a speed-up up to 3.4x for 1 to 4 cores. However, afterwords the gain in speed-up decreases gradually, i.e., from 4 to 16 cores the speed-ups are 6.2x, 8.3x, and 9.8x, respectively. The reasons for the smaller gains when adding many cores is that the ratio of communication to computation cost grows. This is consistent with the observations in [16]. Further, it can also be observed that aggregate queries show better speed-up. Because they require more processing and thus decrease the effect of communication.

SpatialHadoop Comparison: We compare the efficiency, scalability, and ease of implementation of GSG with the leading Big Data system, SpatialHadoop (SH). SH is a map-reduce based system, however, GSG's underlying framework uses memory based computation. Thus, to make a fair comparison, we do not count the I/O cost. We consider queries from all types of query primitives, i.e., FF from the selection, FCF, FLSD and FRL from the structural, and TNL from the aggregate query primitives. We use FF to evaluate the vertex-centric approach of the operator *aggregateMessage (AM)* in GSG. Furthermore, it is implemented with the different number of degrees, i.e., σ (1-3) to reflect the performance with respect to different iterations. To simulate a real workload, we execute multiple queries concurrently, with random input parameters. The results of all the datasets provide similar trends, however, due to page limits, the results of Gowalla and FourSquare are omit-

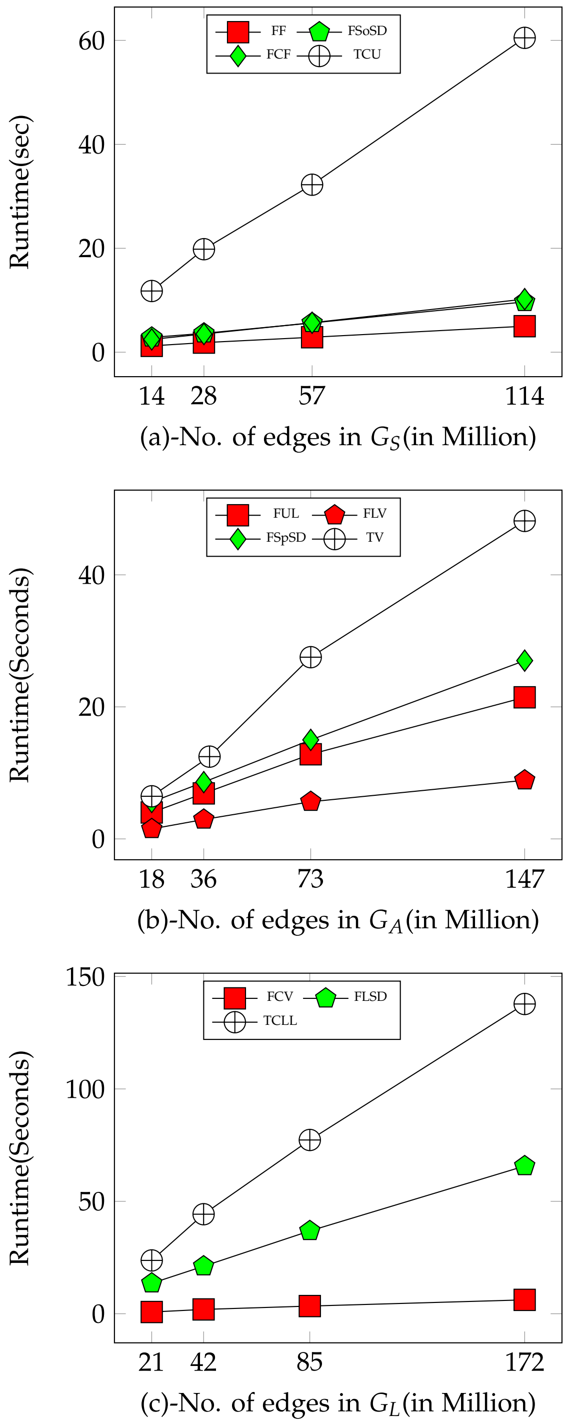


Fig. A.6: Scalability of the GSG with respect to data size

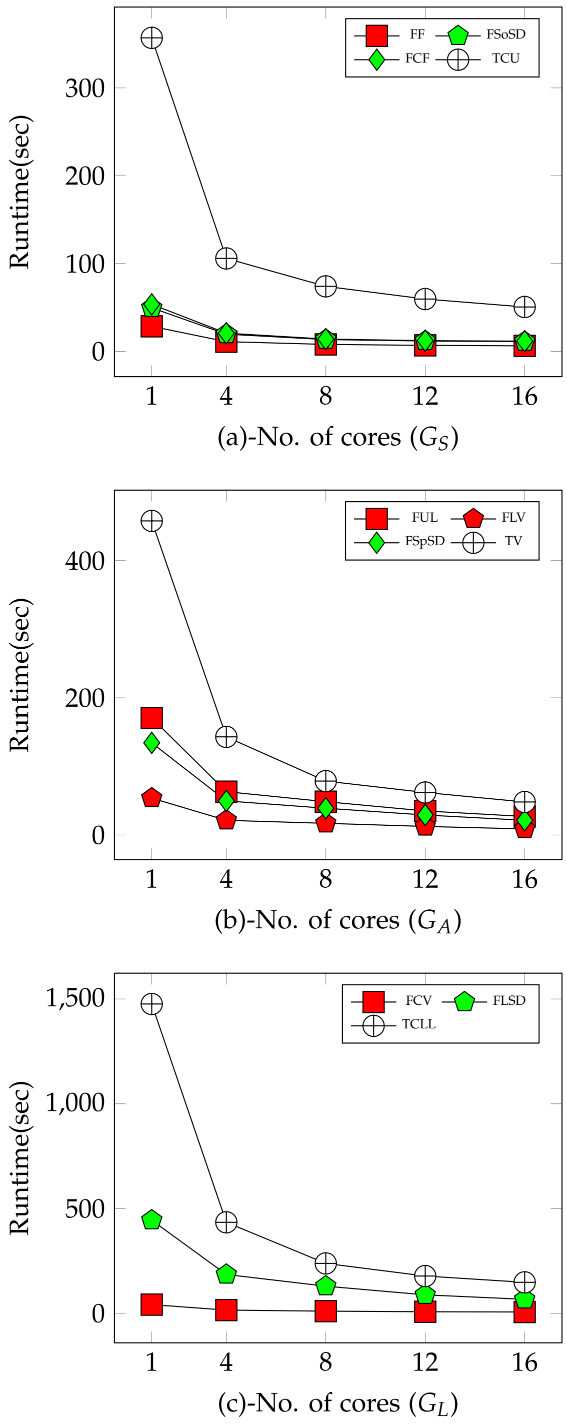
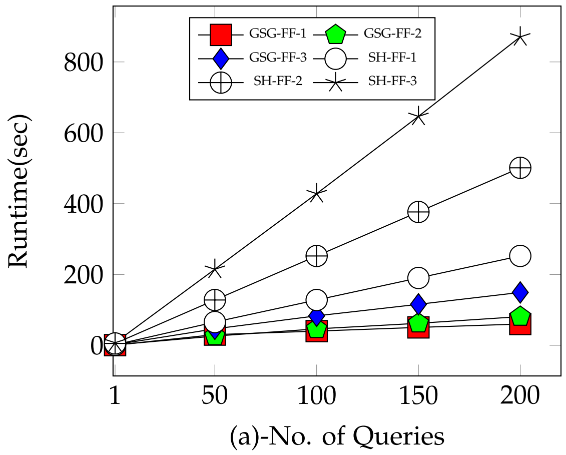
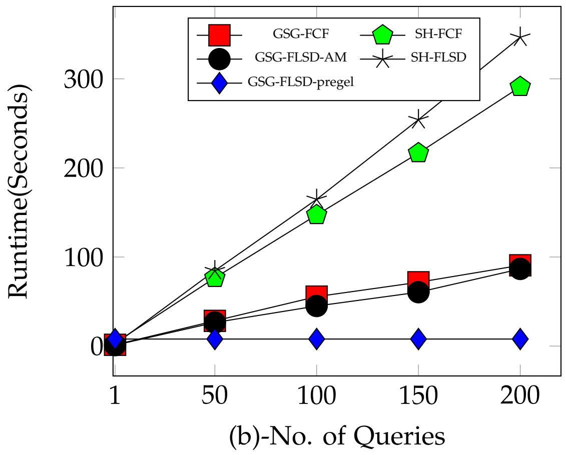


Fig. A.7: Scalability of the GSG with respect to number of cores





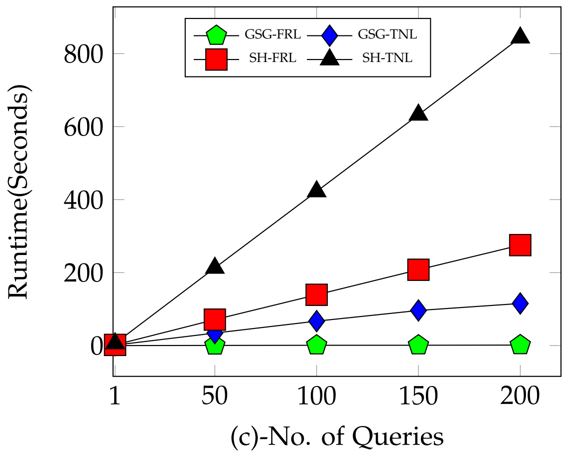


Fig. A.8: Comparison of computation time of GeoSocial-GraphX and SpatialHadoop (BrightKite)

ted. Figure A.8 (a) shows the results for the Brightkite dataset where GSG outperforms SH on average by 2.5 times for a single query and this time difference increases up to 4 times for 3 degrees. Moreover, when going from 1 to 200 concurrent queries, GSG scales 1.5 times better than SH for 1-degree and 3.5 times better for 3-degrees. GSG's better performance is due to its vertex-centric approach as compared to SH's map-reduce based methods.

Figure A.8 (b) shows the results of structural queries, i.e., FCF and FLSD. GSG outperforms SH by 2.5x and scales 1.3 times better than SH from 1 to 200 queries. GSG uses the operators *AM* and *pregel* while SH utilizes traditional map-reduce methods for the implementation of FLSD. Vertex-centric approach by *AM* shows a significant improvement as compared to SH and outperforms it by 2x. The computation time of *pregel* is 4 times slower as compared to *AM* for a single query. The reason is that *pregel* computes the social separation degree of a given vertex with all vertices of the social graph. However, it retains the query time for multiple queries and outperforms *AM* by 9x for 50 queries. The speed-up increases with the number of queries. Thus, we conclude that *pregel* is beneficial for the queries that require a traversal of the whole graph or multiple concurrent queries. Furthermore, it also shows better performance in processing higher degrees operations. However, *AM* is effective for a single query that requires processing up-to few degrees of vertices.

SH is designed for efficient processing of spatial data. Thus, we utilize its corresponding features to compare the results with GSG. We implement aggregate spatial queries, i.e., FRL and TNL to compare the results. These queries are implemented by considering the locations as *Point* in SH. Furthermore, indexes for both SH and GSG are used to compare the computation time. The corresponding results are depicted in Figure A.8 (c). GSG computes FRL 20 times faster and scales 7 times better up to 200 queries as compared to SH. Similarly, for FNL, GSG outperforms SH by 4x and scales 1.5 times better. Results for the Gowalla and FourSquare datasets are similar and not shown due to space constraints.

GSG exposes a programmable API for building general purpose queries. We analyze the ease of implementation of queries in GSG as compared to SH. We calculate and compare the Lines of Code (LoC), required to implement queries for both systems. The queries, FF, FCF, FLSD, FRL and TNL take 3 LoC with GSG, but requires 31, 39, 34, 15, and 12 LoC with SH, respectively, i.e., 4-13 times more LoC as compared to GSG.

Spatial Graph: This set of experiments shows the significance of GSG's materialized spatial graph in processing of the spatial primitives. These primitives can also be processed on the activity graph, i.e., by traversing the user-location data. However, the activity graph based solution has to repeatedly retrieve a large number of check-ins for query processing. The spatial graph based solution alleviates the problem by reusing the intensive computation

Table A.3: Comparison of computation time (in seconds) of spatial primitives on activity graph and spatial graph

Data set	Graph	Time	Query	Activity Graph				Spatial Graph			
				1	5	10	15	1	5	10	15
	G_A	43.7	FCV	7.5	35.4	71.3	109.1	0.3	1.6	3.1	3.9
			FLL	31125.6	155625.9	311254.9	466881.9	2.4	11.9	21.5	32.6
Brighkite			TCLL	31127.5	155620.9	311253.4	466880.5	2.4	7.7	14.7	21.3
	G_L	31127.12	TVLL	31125.4	155622.8	311249.7	466871.5	1.7	6.3	12.2	17.8
			FLSD	31127.2	155632.6	311264.6	466863.8	3.4	11.6	22.3	30.5
	G_A	59.4	FCV	12.8	48.1	95.9	127.1	0.06	0.2	3.1	4.9
			FLL	52180.2	104360.5	156541	208721.5	0.4	0.9	2.4	3.7
Gowalla			TCLL	52180.2	104360.7	156541.2	208721.7	0.3	1.02	2.3	3.8
	G_L	52183.1	TVLL	52180.2	104360.7	156541.2	208721.6	0.2	0.6	1.52	2.67
			FLSD	52184.2	104372.1	156563.6	208757.4	0.39	1.0	2.8	4.1
	G_A	52.19	FCV	8.7	37.8	75.9	108.9	0.5	1.1	2.1	2.9
			FLL	24078.6	48157.3	72235.9	96314.7	2.9	12.3	21.9	30.4
FourSquare			TCLL	24078.6	48157.3	72236.1	96314.8	2.6	8.4	17.1	25.1
	G_L	24079.2	TVLL	24078.5	48157.1	72235.8	96314.4	1.3	5.5	10.6	16.1
			FLSD	24080.6	48163.1	72247.8	96333.7	2.3	10.4	20.1	30.9

during the graph construction. We implement the query primitives as listed in row G_L of Table A.1 to compare the results. These primitives are implemented on the activity graph as well as on the spatial graph. We compare the evaluation time for the query processing on corresponding graphs as shown in Table A.3.

It can be observed that the query execution time of a single query(FLL) on the spatial graph for Brightkite is 13,000 times less when using the activity graph. This time difference increases up to 14,000 times for 15 queries. Among the queries examined, FCV is an outlier. For FCV the speed-up increases from 25 times to 28 times for one to 15 queries. The reason is that its processing requires less traversal of the activity graph as compared to the other queries. The speed-up is more than four orders of magnitude for Gowalla and three orders of magnitude for the FourSquare. The reason for less speed-up for FourSquare is high number of edges in the spatial graph as compared to the Gowalla. These results confirm the importance of the spatial graph for processing of the spatial primitives. Moreover, it can be observed from the Table A.3 that the construction time of the spatial graph is significantly higher than that of the activity graph, i.e., 7000 times in the Brightkite. Furthermore, materialization costs for Brightkite, Gowalla, and FourSquare datasets are 10 MB, 5 MB, and 8 MB, respectively, which is negligible in comparison to other storage. Taking this in combination, we observe that the seemingly high cost of computing the spatial graph is offset already *after a single query*

GSG Optimization: We evaluate the optimizations of GSG by using partitioning and indexing methods. We select a query from the activity graph primitives: FRL that best reflects the spatial nature of LBSN queries. We compare its execution time in three conditions, i.e., non-indexed, partitioned only,

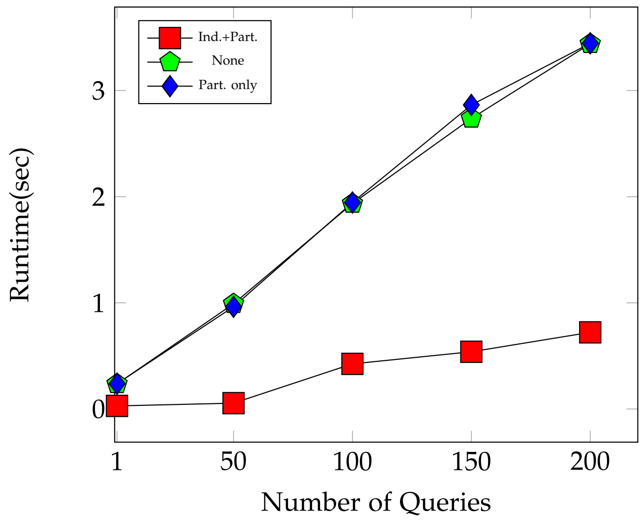


Fig. A.9: Improvement of the GSG with partitioning and indexing for the query: FRL. (FourSquare)

and combined partitioned and indexed. We use the k-d tree for partitioning of data among nodes. Furthermore, we use the k-d tree for the global index and octree as the local index. Figure A.9 provides the results for this experiment. Here, it can be observed that the query execution time for partitioned data is similar to the time of non-partitioned and non-indexed data. The reason is that by partitioning the data, we locate the data items of nearby places on the same node. However, for FRL, the whole dataset is traversed to find the results because the locations of required data points are not determined. Thus, only partitioning is not enough to improve the query execution time. On the other hand, by providing indexes with partitioned data, the nodes that contain the required data are identified and only relevant data is traversed. Thus, for combined partitioning and indexing method, a speed-up of 7x is achieved for FourSquare. Here, the speed-up remains approximately same for an increasing number of concurrent queries. In the case of BrightKite and Gowalla, we get a speed-up of around 5x and 6x, respectively. However, due to limited space, results are not shown here. Furthermore, the computation time for multiple queries decreases significantly. The reason is that the usage of concurrent programming maximizes the computational resources.

We further evaluate the optimization for processing of advanced LBSN queries. We select an advanced query: RangeFriends for the evaluation. The overall query plan is shown in Figure A.5. The query is decomposed into 3 query primitives, i.e., FF, FRL, and FLV. The query is implemented by two ways. First, we compute the query primitives in a sequential way. Sec-

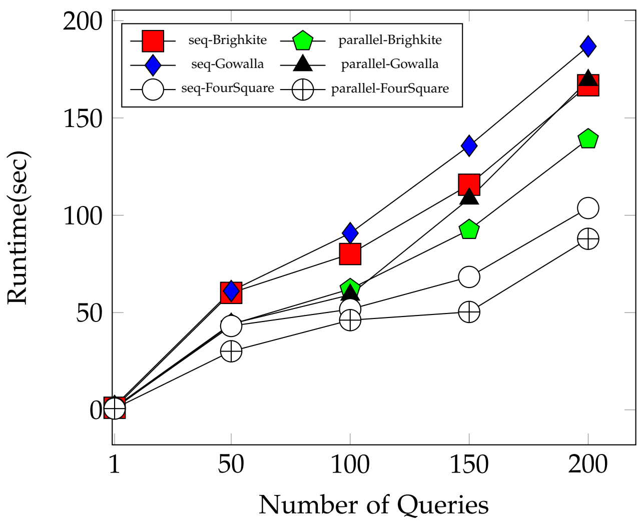


Fig. A.10: Optimization for an advanced LBSN query: RangeFriends. (BrightKite)

ond, the query primitives are computed by using the Akka based concurrent programming model. We further run the experiments for multiple queries. These queries are also processed in parallel. Figure A.10 presents the results on BrightKite dataset, where seq represents sequential traversal. Here, it can be observed that we achieve a speed-up of 1.4x for a single query. The dependency of FLV on FRL causes the decrease in parallelism and therefore, the speed-up. On increasing the number of concurrent queries, the speed-up reaches to 1.2x. The reason is that concurrent threads of multiple queries acquire computational resources that cause the decrease in speed-up for parallel computation of query primitives. Similar trends are observed from results on Gowalla and FourSquare datasets. The results are omitted due to page limits.

6 Related Work

In this section, we present the existing work in two parts, i.e., data processing systems in terms of their appropriateness for processing of LBSN data (graph and spatial data), and data management strategies for LBSN data.

Big Data Systems: First, We provide the analysis of well known open source graph database systems. Neo4j [17] is a single tier graph database. It maintains data in graph formats thus, efficiently process corresponding operations. Apache Giraph [18] and GraphLab [19] are general purposed distributed graph processing systems. Giraph is an analogous system of the proprietary Pregel. Pregel and Giraph exploit Bulk Synchronous Parallel model while GraphLab implements an asynchronous method. These systems work on a vertex-centric approach, thus suitable for iterative graph algorithms and graph analytics. There are several other graph database systems that lies in same category such as VertexDB [20], HypergraphDB [21], flockdb [22], and Trinity [23]. GraphX [24] is a graph parallel platform that is built on top of the distributed tabular data framework, Spark [10] with the optimization for graph processing. Thus, it efficiently supports both tabular and graph based data processing. However, none of these systems explicitly contribute towards spatial operations.

Next, we analyze well known open source spatial data processing systems. PostGIS [25] is a spatial database extension for PostgreSQL. It is a RDBMS based system with the support of processing at a single machine. Neo4j-Spatial [26] is a single tier graph database system with the support of spatial operations. Esri provides tools and APIs [27] for spatial data processing with distributed computing framework Hadoop, i.e., spatial framework for Hadoop and geoprocessing tools for Hadoop. Further, SpatialHadoop [5] is also a distributed system, especially designed for spatial data that optimizes the spatial queries by using map-reduce based spatial indexes. However, none of them can handle large volume spatial graph data efficiently.

We choose GraphX as the underlying framework for building our GSG platform because it has better support for efficient and scalable graph processing. The GraphX data structures and operators are given in Section 4 along with the corresponding changes and extensions in GSG.

LBSN Data Management: This section narrates the overview of existing work in perspective of LBSN data management. In [28], authors maintain the LBSN data in an adjacency matrix. That may cause storage and computation overhead in large LBSNs. In [29], authors aggregate the data and exploit hybrid indexes for data access that may require extensive cost due to the rapid increase of check-ins. In [9], authors segregate the LBSN data into social and geographical (activity) components. This framework works efficiently for the social and activity-centric queries. However, for location-centric queries they need to recursively traverse geographical component that causes the huge computation overhead. Thus, a data management mechanism that could efficiently maintain all the types of queries is required. Further, it should be scalable enough to deal with huge data of LBSN.

Our proposed GSG framework provides a data management mechanism to deal with these issues. We segregate the LBSN data into three graphs namely, social graph, activity graph and spatial graph. Furthermore, it is capable of dealing with LBSN queries with graph and spatial nature on large volumes of data.

7 Conclusion and Future Work

In this paper, we present a platform, GeoSocial-GraphX (GSG) for distributed processing of LBSN queries on large data volumes. GSG segregates the LBSN data into three graphs, namely social graph, activity graph, and spatial graph, to support scalable and efficient processing. A comprehensive set of query primitives is defined on these graphs. The primitives can be combined to answer a wide range of general purpose LBSN queries. GSG also features distributed storage mechanisms, vertex-centric operators, and spatial indexing and partitioning techniques that are used to further optimize the processing and scalability of LBSN queries.

The experiments on real and synthetic datasets show that GSG scales well on multicore architecture and for increasing dataset sizes. The partitioning and indexing methods improve the performance up to 6 times. Furthermore, segregation of LBSN and usage of the spatial graph improves the processing of location-centric queries upto four orders of magnitude. GSG also outperforms the competing system, SH in terms of efficiency, scalability and LoC by up to 20X, 7X and 13x, respectively.

In the future, we plan to consider updates in LBSN graphs. The graphs will be updated using batch updates to optimize the update process. Furthermore, GSG will be enriched with more LBSN queries with corresponding optimization methods for query specific data management and processing techniques.

References

[1] M. Ye, P. Yin, and W.-C. Lee, "Location recommendation for location-based social networks," in *GIS*, 2010, pp. 374–383.

[2] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," in *GIS*, 2012, pp. 199–208.

[3] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *WWW*, 2009, pp. 791–800.

[4] "Foursquare," https://foursquare.com/about.

[5] "Spatialhadoop," http://spatialhadoop.cs.umn.edu/.

[6] "mongodb," https://www.mongodb.org/.

[7] J. Shi, N. Mamoulis, D. Wu, and D. W. Cheung, "Density-based place clustering in geo-social networks," in *SIGMOD*, 2014, pp. 99–110.

[8] H. Gao, J. Tang, and H. Liu, "Exploring social-historical ties on location-based social networks," in *AAAI*, 2012.

[9] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," in *PVLDB*, 2013, pp. 913–924.

[10] "Spark," https://spark.apache.org/.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012, pp. 2–2.

[12] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *OSDI*, 2014, pp. 599–613.

[13] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *OSDI*, 2012, pp. 17–30.

[14] "akka," http://akka.io/.

[15] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *KDD*, 2011, pp. 1082–1090.

[16] R. S. Xin, D. Crankshaw, A. Dave, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: Unifying data-parallel and graph-parallel analytics," *CoRR*, 2014.

[17] "Neo4j,," http://neo4j.com/.

[18] "Apache giraph," http://giraph.apache.org/.

[19] "Graphlab," https://graphlab.org.

[20] "Vertexdb," http://www.dekorte.com/projects/opensource/vertexdb/.

[21] "Hypergraphdb," http://www.hypergraphdb.org/.

[22] flockdb. https://github.com/twitter/flockdb.

[23] "Trinity," http://research.microsoft.com/en-us/projects/trinity/.

[24] "Graphx," http://spark.apache.org/graphx/.

[25] “Postgis,” http://postgis.net/.

[26] “Neo4j-spatial,” https://github.com/neo4j-contrib/spatial.

[27] “Gis tools for hadoop,” http://esri.github.io/gis-tools-for-hadoop/.

[28] W. Liu, W. Sun, C. Chen, Y. Huang, Y. Jing, and K. Chen, “Circle of friend query in geo-social networks,” in *DASFAA*, 2012, pp. 126–137.

[29] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler, “Buddy tracking - efficient proximity detection among mobile friends,” in *INFOCOM*, 2004, pp. 298–309.

Paper B

Effective and Efficient Location Influence Mining in Location-Based Social Networks

Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Torben Bach Pedersen

The paper is under revision for
a Journal publication

Abstract

Location-based social networks (LBSN) are social networks complemented with location data such as geo-tagged activity data of its users. In this paper, we study how users of an LBSN are navigating between locations and based on this information we select the most influential locations. In contrast to existing works on influence maximization, we are not per se interested in selecting the users with the largest set of friends or the set of locations visited by the most users; instead, we introduce a notion of location influence *that captures the ability of a set of locations to reach out* geographically *by utilizing their visitors as message carriers. We further capture the influence of these visitors on their friends in LBSNs and utilize them to predict the potential future location influence more accurately. We provide exact on-line algorithms and more memory-efficient but approximate variants based on the HyperLogLog and the modified-HyperLogLog sketch to maintain a data structure called* Influence Oracle *that allows to efficiently find a top-k set of influential locations. Experiments show that our new location influence notion favors diverse sets of locations with a large geographical spread and that our algorithms are efficient, scalable and allow to capture future location influence.*

The layout has been revised.

1 Introduction

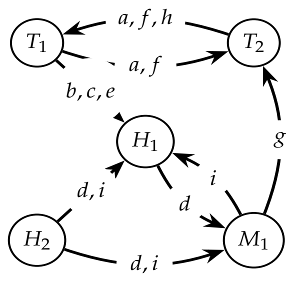
One of the domains in social network analysis [1–4] that received ample attention over the past years is *influence maximization* [5], which aims at finding influential users based on their social activity. Applications like viral marketing utilize these influential users to maximize the information spread for advertising purposes [6]. With the pervasiveness of location-aware devices, nowadays, social network data is often complemented with geographical information. For instance, users of a social network share geo-tagged content such as locations they are currently visiting with their friends. These social networks with location information are called location-based social networks (LBSNs). In LBSNs, the location information offers a new perspective to view users' social activities. In this paper, we study navigation patterns of users based on LBSN data to determine *influential locations*. Where other works concentrate on finding *influential users* [7], *popular events* [8], or *popular locations* [9], we are interested in identifying sets of locations that have a large *geographical* impact. Although often overlooked, the geographical aspect is of great importance in many applications. This geographical information can be utilized to provide more targeted marketing strategies. For example, unlike viral marketing which focuses on finding influential users and spreading the message via word of mouth marketing (WOMM), influential locations can be found and information can be spread using out-of-home/ outdoor marketing (OOH) e.g., by putting advertisements on billboards and distributing promotional items on such locations. For instance, consider the following example.

Example 1.1
A marketer is interested in creating visibility for her products to the maximum regions in a city by offering free promotional items such as T-shirts with a printed promotional message. To do that she has to choose locations to distribute the promotional items to visitors.

In order to choose the most suitable locations for offering these items, not only the popularity of the places is important, but also the geographical reach. By visiting other locations, people that were exposed to the advertisement, especially the receivers of the promotional items, may indirectly promote the products. For example, by wearing the shirt they expose the T-shirt's message to the people at the places they go to and they talk about it with their friends and relatives. Thus, when the goal is to create awareness of the product name, it may be preferable to have a moderate presence in many locations throughout the whole city rather than a high impact in only

Check-ins

loc	Users		
	t=1	t=2	t=3
T_1	b, c, e, f	a, h	f
T_2	a, h	f, g	a
M_1	g	i	d
H_1	–	b, c, d, e	i
H_2	d, i	–	–



Friendships

User	Friends
a	c, h, i
b	d, f, i
c	a, f
d	b, e
e	d, g
f	c, b
g	e
h	a
i	a, b

Fig. B.1: [10] Running example of an LBSN: Check-ins(L) shows the visits of users (represented by lower-case letters; a, b, etc.) at locations (represented by upper-case letters; T_1, H_1, etc.) at time stamps t=1, 2 and 3. Graph (C) depicts the movement of users between consecutive locations. Friendships (R) show the friends of each user in the social network.

a few locations. An illustration of this example is given in Figure B.1. Nodes represent popular locations of different categories, such as tourist attractions (T_1, T_2), a metro station (M_1), and hotels (H_1 and H_2). Lowercase letters represent users. For each user, her friends in the social network and check-ins have been given. The top-2 locations with the maximal number of unique visitors are T_1 and M_1. The geographical impact of these locations, however, is not optimal; visitors of these locations reach only T_2 and H_1. On the other hand, the visitors of T_1 and H_2 visit all locations, i.e., users a, f and b, c, e visit T_2 and H_1 after visiting T_1, respectively, and users d, i visit H_1 and M_1 after H_2.

To capture geographical spread and influence, in Section 3, we introduce the notion of a *bridging visitor* between two locations as a user that visits both locations within a limited time span. If there is a significant number of bridging visitors from one location to another, we say that there is an influence. We introduce different models that capture when the number of bridging visitors is considered to be sufficient to claim influence between locations. One model is based on the absolute number of visitors, and one on the relative number. For each of these two models, we further present a direct bridging visitor based location influence model and two friendship-based location influence models that take the social graph of the LBSN into account. The friendship-based location influence models are based on the following observations obtained by detailed analysis of three real-life LBSNs. The first observation is that users tend to follow their friends and perform the same activities; e.g., in Figure B.1, users i and f visited the same locations T_1 and H_1 after their friend b did. The second observation is that sometimes the number of visits/activities to some locations can be rather low because of data sparsity, especially when the time window used in the algorithms is small. The data sparsity may affect the location influence models and capture less influential seeds. Considering these observations, the friendship-graph-based models allow to compute potential future bridging visitors. By incorporating such visitors, the models overcome the data sparsity problem and capture location influence more accurately. The first friendship-based influence model considers all friends of bridging visitors, while the second model computes the influence of bridging visitors on their friends in LBSNs and only incorporates the strongly influenced friends as potential future bridging visitors. Based on these models, we define influence for sets of locations and the *location influence maximization problem*: *Given an LBSN and a parameter k, find a set of k locations such that their combined location influence on other locations is maximal.*

To solve this problem, in Section 4 a data structure, called *Influence Oracle*, is presented that maintains a summary of the LSBN data that allows to determine the influence of any set of locations at any time. Based on this data structure, we can easily solve the location influence maximization problem using a greedy algorithm. As for large LBSNs with lots of activities the memory requirements of our algorithm can become prohibitively large, we also develop a more memory-friendly version based upon the well-known HyperLogLog sketch [11]. This algorithm gets further refined in Section 4.3 where we introduce a single-carrier-based influence maximization mechanism for capturing influence in information propagation scenarios where even a single carrier can carry the influence such as propagation of infections, and confidential information in specialized information networks. We provide off-line and on-line memory- and time-efficient algorithms for the single-carrier-based-influence maximization. Next, in Section 5 we provide a greedy algorithm for finding top-k influential locations.

In Section 6 we analyze several LBSNs to select reasonable threshold values for our models and to verify our claims. In Section 7 we evaluate the proposed notions and algorithms using the real-world datasets in term of effectiveness and efficiency.

In summary, the main contributions of this paper are (i) the introduction and motivation of a new location influence notion based on LBSN data, (ii) the development of an efficient Influence Oracle, and (iii) the demonstration of the usefulness of the location influence maximization problem in real-life LBSNs.

This paper is an extended version of the conference paper [10]. As an extension, we present a novel mechanism for spreading location influence that incorporates the influential users based on their geographical activities and social friends. The mechanism is given in Section 3.1. On the basis of the mechanism, we further propose two variants of absolute and relative influence models (given in Section 3.2). The new algorithms for finding such location influence are provided in Section 4.1. Furthermore, the single-carrier-based influence maximization mechanisms (given in Section 4.3) and the two algorithms for capturing such influence also constitute previously unpublished work. The methods for finding suitable values of the thresholds for new models are given in Section 6. We further present a set of new experiments for validating the proposed approaches in terms of effectiveness (Section 7.2) and efficiency (Section 7.5) as well to evaluate their significance in comparison with existing methods.

2 Related Work

Influence maximization in the context of traditional social networks and LBSNs has been studied in much detail. We divide the existing studies in the domain into three groups. The first group consists of approaches for finding influential users in traditional social networks. The second group covers studies that use check-ins as an additional source of data to identify influential users in LBSNs, whereas the third group utilizes the check-ins for finding influential locations in LBSNs.

Influential Users in Social Networks. The influence maximization approaches in social networks are generally divided into two main groups. The first group of studies [5, 12–14] operates on static graphs and assumes that the influence relationships among nodes are already known. They compute the influence probability of a node using probabilistic simulations and use them for determining influence among nodes. These approaches do not capture the temporal and dynamic nature of real networks such as social media. On the other hand, the second group of studies in this category [15–18] is data-driven and requires interactions of users and their activities. They compute influence probabilities based on relationships and historic activities of nodes such as common actions among two friends within a specified time. Thus, these studies are more suitable for dynamic networks such as LBSNs. Goyal et al. [18], propose the first data-based approach for finding influential users in social networks by considering the temporal aspect in the cascade of common activities of users. In [17], they further introduce a time window based approach to determine the true leaders in social networks. In [19], they present several models to compute influence probabilities. They provide static models based on likelihood estimation, as well as continuous and discrete time models for capturing the dynamic behavior of users in social networks. However, the limitations of these approaches are their assumptions that information propagation is non-cyclic and thus users can perform an action only once. In order to find the influence of users, we provide an extension of [19] as part of our influential friends-based location influence model. Our algorithm identifies influential nodes without any constraints on the number of times a user performs an action. It further allows cyclic propagation of information.

Influential users and events in LBSNs. Zhang et al. [8] use the social and the geographical correlation of users to find influential users and popular events. Users with many social connections are considered influential and events visited by them are considered important. Similarly, Wu et al. [7] identify influential users in LBSNs on the basis of the number of followers of their activities (check-ins). Li et al. [20] and Bouros et al. [21] on the other hand, identify regionally influential users on the basis of their activities. The focus of the work by Wen et al. [22] and Zhou et al. [23] is to find and utilize the influential users for product marketing strategies such as word-of-mouth. Our focus, however, is to find influential *locations* that could be used, e.g., for outdoor marketing, hence, none of the previous works applies directly to our problem.

Location Promotion in LBSNs. Zhu et al. [9], Hai [24], and Wang et al. [25] study location promotion. Given a target location, their aim is to find the users that should be advertised to attract more visitors to this location. Doan et al. [26] compute the popularity ranks of locations based on the number of visitors. Zhou et al. [23] study the product promotion in O2O (on-line to off-line) model using LBSNs. Their model combines the on-line features, i.e., network topology (social network) and off-line user properties such as daily activity area and location preferences of users. Based on these features they find top-k users that can maximize the number of influenced users for a given location (product). These studies have different objectives as compared to our problem statement as they focus on finding top-k users that can attract the maximum visitor for a given location.

Novelty. Our work is different from all of the above as we focus on finding a *set of influential locations* where influence is defined using visitors as a mean to spread influence to other locations. Applications include outdoor marketing by selecting locations with the maximal geographical spread.

3 Location-Based Influence

In this section, we first provide preliminary definitions, then present location influence and different models to capture it, and finally we formally define the *Location Influence Maximization* and *Oracle* problems.

Let a set of users U and a set of locations L be given.

Definition B.1. *An* activity *[10] is a visit of a user to a location. It is a triplet (u, l, t), where $u \in U$ is a user, $l \in L$ a location and t is the time of visit of u to l. The set of all activities over U and L is denoted $\mathcal{A}(U, L)$.*

Definition B.2. *A* Location-based Social Network (LBSN) *[10] over U and L consists of a graph $G_S(U, F)$, called the* social graph, *where $F \subseteq \{\{u, v\} | u, v \in U\}$ represents friendships between users, and a set of activities $A \subseteq \mathcal{A}(U, L)$. It is denoted $LBSN(G_S, A)$.*

3.1 Bridging Visitors for Location Influence

We define the influence of a location by its capacity to spread its visitors to other locations. The intuition behind this is that the visitors exposed to a message at a location will spread the message to other locations they visit. Thus, the location influence (indirectly) captures the capability of a location to spread a message to other geographical regions by using common visitors as message carriers. The effect of an activity in a location, however, usually remains effective only for a limited time. We capture this time with the *influence window* threshold ω. Such visitors that spread messages among locations based on their activities in LBSNs are called *Bridging Visitors (B)*. Next, we provide three types of bridging visitors for spreading location influence in LBSNs.

Direct Bridging Visitors

The first type of bridging visitors is called *Direct Bridging Visitors*:

Definition B.3. *Direct Bridging Visitor [10]: Given an $LBSN(G_S, A)$ and time window ω, a user u is said to be a* direct bridging visitor *from location s to location d if there exist activities $(u, s, t_s), (u, d, t_d) \in A$ such that $0 < t_d - t_s \leq \omega$. We denote the set of all direct bridging visitors from s to d by $B_D(s, d)$.*

Example 3.1
Consider the running example of Figure B.1. Let $\omega = 2$. Then, $B_D(T_1, H_1) = \{b, c, e\}, B_D(H_2, H_1) = \{d, i\}$ and $B_D(M_1, H_1) = \{i\}$.

Friends-Based Bridging Visitors

Activity data in LBSNs is often sparse in the sense that the number of check-ins per location is low. In Section 7, we see that the real-world datasets have only up to 6 check-ins per location on average. This sparsity of data affects the computation of location influence as usually there are very few bridging visitors among locations. In order to deal with this issue, we use the observation that users tend to perform similar activities as their friends (this claim is verified and confirmed in Section 6). Thus, the friends of bridging visitors have potential to carry the same message as the bridging visitors do. Based, on this observation we define *Friends-Based Bridging Visitors:*

Definition B.4. *Friends-Based Bridging Visitors: Given an* $LBSN(G_S, A)$*, time window* ω*, locations* s *and* d*, and direct bridging visitors from* s *to* d $B_D(s,d)$*, the set of* Friends-Based bridging visitors *between* s *and* d *is denoted by* $B_F(s,d)$*:*

$$B_F(s,d) = B_D(s,d) \bigcup_{u \in B_D(s,d)} F_u \tag{B.1}$$

where F_u *is the set of friends of* u*, i.e.,* $F_u = \{v | (u,v) \in F\}$

Example 3.2
Consider again the running example of Figure B.1. Let $\omega = 2$. Then, $B_F(T_1, H_1) = \{a,b,c,d,e,f,g,i\}, B_F(H_2, H_1) = \{a,b,d,e,i\}$ and $B_F(M_1, H_1) = \{a,b,i\}$.

Influenced Friends-Based Bridging Visitors

Next, we further improve the notion of bridging visitors based on the following observation. Not all friends of bridging visitors may follow them, thus considering all of them as potential future visitors may bring high inaccuracy in predicting the number of bridging visitors and so in capturing the location influence. To improve the accuracy, we evaluate the ability of each bridging visitor to persuade their friends to follow them. The friends that are significantly influenced by the bridging visitors are called *Influenced Friends-Based Bridging visitors*.

A user v is considered to be influenced by a user u, if u visits a location l and v visits the same location after u within a particular time. In order to find such influence, we present an extended version of an existing algorithm given in [19] that computes the influence probabilities using a Bernoulli distribution based on partial credit distribution and discrete time constraint models [19]. According to the model, the influence probability is measured by the ratio of the number of successful attempts to persuade the influenced user to follow the influential user's activities over the total number of trials. Considering that a user can be influenced by multiple sources for an activity, the influential credit for each following activity is distributed among all such influential parents using the Partial Credit Distribution model. Furthermore, as influence probability is dependent on time, a discrete time constraint model is incorporated, which ensures that a user can influence other users only within the given time window. It is worth noting that such a time window can be different than the ω given in Definition B.3. However, for our experiments, we consider the same ω for such a time window because we consider ω as a maximal time between two activities to still consider them connected. Goyal et al. in [19] do not capture repeating activities of users considering that traditional social network users are unlikely to repeat their actions such as re-posting the same contents. However, in LBSNs, users may visit the same locations again. This implies that, if an influential user visits a location after her influenced user, she is considered to be influenced by her influenced user. Our proposed algorithm (given in Algorithm B.2) captures multiple activities of users at the same locations and thus, such aforementioned relationships. We ensure however that a user is influenced maximally once for an activity, i.e., a visit to a location, regardless of the number of times she visits the same location within ω. However, if the influential user visits the same location at another time, she may influence the same influenced user again. We denoted the influence probability of a user u on v as $p_{u,v}$. Next, we utilize such influence probabilities to define the influenced friends-based bridging visitors.

Definition B.5. *Influenced Friends-Based Bridging Visitors: Given $LBSN(G_S, A)$, time window ω, locations s and d, direct bridging visitors from s to d $B_D(s,d)$, and a threshold of the influence probability between users θ, a set of* Influenced Friends-Based bridging visitors *between s and d is denoted by $B_I(s,d)$:*

$$B_I(s,d) = B_D(s,d) \cup \{u \in U | \sum_{v \in B_D(s,d)} p_{v,u} \geq \theta\} \tag{B.2}$$

Example 3.3
Let $\tau_{IA} = 2$, $\omega = 2$ and $\theta = 0.2$. In the running example, a is the influenced user of h. a followed one out of two activities of h, i.e., for visiting T_2 and there is no other friends influencing a for this activity, thus, $p_{h,a} = 1/2 = 0.5 \geq \theta$. Similarly, the influenced users of b and c are $\{i, f\}$, and $\{a, f\}$, respectively. The other users do not have any influenced visitors as their influence probability is less than θ.

3.2 Methods for determining Location Influence

Next to the selection of message carriers, a second dimension is when we consider influence to be present and to what extent. For this purpose, we introduce two influence models (M).

Absolute Influence Model (M_A)

In practice, if a significant number of people perform an activity, then it is considered compelling. Thus, in order to avoid insignificant influences among locations, we use a threshold τ_A. The influence of a location s on a location d is considered only if the number of bridging visitors from s to d is greater than τ_A. This model is referred as the *Absolute Influence Model (M_A)*. The influence of a location s on d under M_A is represented by $I_A(s,d)$:

$$I_A(s,d) := \begin{cases} 1, & \text{if } |B(s,d)| \geq \tau_A \\ 0, & \text{otherwise} \end{cases} \tag{B.3}$$

We instantiate B in Equation B.3, with B_D, B_F or B_I, and τ_A with τ_{DA}, τ_{FA} or τ_{IA} to compute direct absolute influence (I_{DA}), friends absolute influence (I_{FA}) or influenced-friends absolute influence (I_{IA}), respectively.

Example 3.4
Consider the running example of Figure B.1. Let the information carriers be the direct bridging visitors B_D, $\tau_A = 2$, and $\omega = 2$. Then, $I_{DA}(T_1, H_1) = 1$ because $|B_D(T_1, H_1)| = 3$. Similarly, $I_{DA}(H_2, H_1) = 1$. However, $I_A(M_1, H_1) = 0$ because $|B_D(M_1, H_1)| = 1$.

The influence between two locations may change with the value of τ_A and ω. For example, if we update the value of τ_A to 3 and ω to 2, $I_{DA}(T_1, H_1) = 1$, but, $I_{DA}(H_2, H_1)$ becomes 0.

Relative Influence Model (M_R)

In M_A, the influences of two pairs of locations are considered equal as long as the number of their bridging visitors is greater than τ_A. Sometimes, however, the relative number of contributed bridging visitors is important. Consider, for example, a popular location s that attracts many visitors and a non-popular location d with few visitors. In such a setting, to capture the influence of s on d, we may have to set the absolute threshold τ_A very low. This low value of τ_A, however, may result in many other popular locations being influenced by s, even if only a very small fraction of their visitors comes from s. Therefore, in such situations, it may be beneficial to use different thresholds for different destinations, relative to the number of visitors in these destination locations. This notion is captured by the *Relative Influence Model (M_R)*. The influence of s on d under M_R is represented by $I_R(s,d)$ and is parameterized by the relative threshold τ_R:

$$I_R(s,d) := \begin{cases} 1, & \text{if } \dfrac{|B(s,d)|}{|V(d)|} \geq \tau_R \\ 0, & \text{otherwise} \end{cases} \tag{B.4}$$

where $V(d)$ is the set of users who visited location d. We instantiate B in Equation B.4, with B_D, B_F or B_I, τ_R with τ_{DR}, τ_{FR} or τ_{IR}, and V with V_D, V_F: a set of V_D and their friends, or V_I: a set of V_D and influenced friends

	Location-Based Influence Models					
	Absolute Influence			Relative Influence		
	Direct Bridging Visitors	**Friends-Based Bridging Visitors**	**Influenced Friends-based Bridging Visitors**	**Direct Bridging Visitors**	**Friends-Based Bridging Visitors**	**Influenced Friends-based Bridging Visitors**
Label	Direct Absolute Model (M_{DA})	Friends Absolute Model (M_{FA})	Influenced Friends Absolute Model (M_{IA})	Direct Relative Model (M_{DR})	Friends Relative Model (M_{FR})	Influenced Friends Relative Model (M_{IR})
Parameters	τ_{DA}, ω	τ_{FA}, ω	τ_{IA}, ω, θ	τ_{DR}, ω	τ_{FR}, ω	τ_{IR}, ω, θ
Location Influence	I_{DA}	I_{FA}	I_{IA}	I_{DR}	I_{FR}	I_{IR}

Table B.1: Types of location-based influence models with labels and the parameters. Further, notations of the location influences captured by these models are also depicted.

of visitors in V_D, to compute direct relative influence (I_{DR}), friends relative influence (I_{FR}) or influenced-friends relative influence (I_{IR}).

Example 3.5
Consider the running example given in Figure B.1. Let the information carriers be the direct bridging visitors B_D, $\tau_I R = 0.4$, and $\omega = 2$. In this example, $I_{DR}(T_1, H_1) = 1$ because $\frac{|B_D(T_1,H_1)|}{|V_D(H_1)|} = \frac{|\{b,c,e\}|}{|\{b,c,d,e,i\}|} = \frac{3}{5} \geq \tau_{IR}$, Similarly, $I_R(H_2, H_1) = 1$ and $I_{DR}(M_1, H_1) = 0$.

In subsection 3.2, we presented two ways to determine the location influence. Each of the ways can utilize any of the three types of bridging visitors (given in subsection 3.1) to spread the location influence. Thus, in total, we have six models for spreading influence in LBSNs as given in Table B.1.

3.3 Combined Location Influence

Based on the influence models, a location can influence multiple other locations. In order to capture such influenced locations, we define the *location influence set*:

Definition B.6. *Given a location s, and an influence model M, the* location Influence Set $\phi_{I_M}(s)$ *is the set of all locations for which the influence of s on that location under M is* 1, *i.e.,* $\phi_{I_M}(s) = \{d \in L \mid I_M(s,d) = 1\}$.

Next, we define *combined location influence* for a set of locations S. To do this, we use the following principled approach: any activity at one of the locations of S is considered an activity from S. In that way we can capture the cumulative effect of the locations in S; even though all locations in S, in isolation may not influence a location d, together they may influence it. The bridging visitors from a set of locations S to d is represented by $B(S,d)$:

$$B(S,d) = \bigcup_{s \in S} B(s,d) \tag{B.5}$$

The influence of a set of locations S on location d under M_A and M_R is defined similarly as for single locations. For instance, the influence of S under M_A is given by

$$I_A(S,d) := \begin{cases} 1, & \text{if } |B(S,d)| \geq \tau_A \\ 0, & \text{otherwise} \end{cases} \tag{B.6}$$

Example 3.6
In Figure B.1, let $\omega = 2$, $\tau_A = 3$ and $S = \{T_1, M_1\}$. Under M_A, $T_2 \notin \phi(T_1)$ and $T_2 \notin \phi(M_1)$. However, $T_2 \in \phi(S)$ as $|B(S,T_2)| = |\{a,f,g\}| \geq \tau_A$.

3.4 Problem Formulation

Based on these influence models, we now formally define the problem statements. We first present a problem statement for finding the most influential locations in LBSNs:

Problem B.1. *(Location Influence Maximization Problem) Given a parameter k, an $LBSN(G_S, A)$, and an influence model M, the location influence maximization problem is to find a subset $S \subseteq L$ of locations, such that $|S| \leq k$ and the number of influenced locations $|\phi_{I_M}(S)|$ is maximum.*

In order to solve the location influence maximization problem efficiently, we first introduce an efficient solution to the following subproblem.

Problem B.2. *(Oracle Problem) Given an $LBSN(G_S, A)$ and an influence model M, construct a data structure that allows to answer: Given a set of locations $S \subseteq L$ and a threshold τ, what is the combined location influence $\phi_{I_M}(S)$.*

4 Influence Oracle

In this section, we provide solutions for the Oracle problem. First, in Section 4.1, we provide a generic algorithm for constructing an influence oracle for any influence model. Then, in Section 4.2, we present an approximate but a more memory- and time-efficient algorithm for constructing the Influence Oracles for the M_D(M_{DA} *and* M_{DR}) and the M_F(M_{FA} *and* M_{FR}) models. After that, in Section 4.3 we present an even more efficient algorithm for the special case of the M_{DA} model with $\tau = 1$.

4.1 Exact Influence Oracle

In this section, we provide a data structure for maintaining exact location summaries for each location which works for the M_D model. Extension of this algorithm for incorporating the M_F and the M_I model are discussed later in this section.

Definition B.7. *The* Complete location summary *for a location $s \in L$ is the set of locations that have at least one bridging visitor from s, together with these bridging visitors; i.e., $\varphi(s) := \{(d, B(s,d)) \mid d \in L \land |B(s,d)| > 0\}$.*

We assume activities arrive continuously and deal with them one by one. For the Oracle, we maintain a summary that consists of the collection of individual summaries $\varphi(s)$ for each location S. We present an on-line algorithm (Algorithm B.1) to incrementally update these summaries.

If a user u visits a location s at time t, then u acts as a bridging visitor between all the locations u visited within the last ω time stamps and s. Therefore, for each user $u \in U$, we maintain a set of locations the user has visited and the corresponding latest visiting time. This is called the *visit history* $\mathcal{H}(u)$ and is defined as $\mathcal{H}(u) := \{(s, t_{max}) | u \in V(s), t_{max} = \max\{t \mid (u,l,t) \in A\}\}$. Suppose that we have the complete location summary for the check-ins so far and the visit history of all users, and a new activity (u,d,t) arrives. We update the complete location summary as follows: the location-time pair (d,t) is added in $\mathcal{H}(u)$ if d does not already appear in the visit history, otherwise the latest visit time of d is updated to t in $\mathcal{H}(u)$ (line 13). Furthermore, for every other location-latest visit time pair (s,t') in the history of u, $\varphi(s)$ is updated by adding user u to the set of bridging visitors from s to d provided that the difference between the time stamps $t - t'$ does not exceed the threshold ω (line $5 - 10$). This procedure is illustrated in Algorithm B.1. The visit history $\mathcal{H}(u)$ is pruned at line 11 to remove those locations which were visited by u more than ω time ago. Pruning of old locations from the visit history can be done at regular interval for all locations.

Example 4.1
We illustrate the algorithm using the running example shown in Figure B.1. For simplicity, we only consider the activities of two users: d and i. We also add a new activity of d at H_2 at time stamp 5. In this example, we consider $\omega = 2$. The activities are processed one by one in increasing order of time. We show how the visit history $\mathcal{H}(i)$, $\mathcal{H}(d)$ and the complete location summaries $\varphi(H_1)$, $\varphi(H_2)$, $\varphi(M_1)$ evolve with different activities at different timestamps in Figure B.2. Note, at time stamp 5 only $\varphi(M_1)$ is updated even though M_1 and H_1 are both in the visit histories of d because $\omega = 2$. The visit history of d is pruned by removing H_1 from the $\mathcal{H}(d)$ as no future

	$t = 1$	$t = 2$	$t = 3$	t=5
Activity:	$(i, H_2, 1)$ $(d, H_2, 1)$	$(i, M_1, 2)$ $(d, H_1, 2)$	$(i, H_1, 3)$ $(d, M_1, 3)$	$(d, H_2, 5)$
$\mathcal{H}(i)$:	$\{(H_2, 1)\}$	$\{(H_2, 1), (M_1, 2)\}$	$\{(H_2, 1), (M_1, 2), (H_1, 3)\}$	$\{(H_1, 3)\}$
$\mathcal{H}(d)$:	$\{(H_2, 1)\}$	$\{(H_2, 1), (H_1, 2)\}$	$\{(H_2, 1), (H_1, 2), (M_1, 3)\}$	$\{(M_1, 3), (H_2, 5)\}$
$\varphi(H_1)$:	$\{\}$	$\{\}$	$\{(M_1, \{d\})\}$	$\{(M_1, \{d\})\}$
$\varphi(H_2)$:	$\{\}$	$\{(H_1, \{d\}), (M_1, \{i\})\}$	$\{(H_1, \{d\}), (M_1, \{i, d\})\}$	$\{(H_1, \{d\}), (M_1, \{i, d\})\}$
$\varphi(M_1)$:	$\{\}$	$\{\}$	$\{(H_1, \{i\})\}$	$\{(H_1, \{i\}), (H_2, \{i\})\}$

Fig. B.2: [10] An example of updating locations summaries for location H_1, H_2 and M_1 and visit histories of users i and d under M_A model for $\omega = 2$ at every time stamp.

activities by d affect $\varphi(H_1)$. The visit time of H_2 is updated to the latest visit time. Similarly, $\mathcal{H}(i)$ is also pruned.

It can be observed from the example that a new activity of a user u only updates the complete location summary of the locations in the recent visit history of u. Notice that, since the activities of a user arrive in strictly increasing order of time, the size of $\mathcal{H}(u)$ is upper bounded by ω, as only locations that are visited within a time window ω are processed and a user can only visit one location at a time.

Proposition B.1. *For the M_{DA} model, the time required to process an activity by Algorithm B.1, is $\mathcal{O}(\omega \log(|U|))$. The complete location summary $\{\varphi(l) | l \in L\}$ can be stored in $\mathcal{O}(|L|^2|U|)$ memory and the visit history$\{\mathcal{H}(u) | u \in U\}$ in $\mathcal{O}(|U|\omega)$ memory.*

Proof. The visit history $\mathcal{H}(u)$ for a user u can at maximum have ω locations hence the for loop in line 4 of the algorithm will run for maximum ω iteration. The maximum set size of the bridging visitors is $|U|$, so adding an element to the set will take maximum $\log(|U|)$ time using an appropriate data structure, such as a balanced tree for storing a set. Thus, the total time for processing an activity in the worst case is $\mathcal{O}(\omega \log(|U|))$. The memory complexity is straightforward as there could be maximally $|L|$ influenced locations and the bridging visitor set size is at most $|U|$, hence, the memory complexity is $\mathcal{O}(|L||U|)$ in the worst case for a location hence for all locations it is $\mathcal{O}(|L|^2|U|)$. □

Algorithm B.1: Exact Oracle: Updating complete location summaries [10]

```
Input: New activity $(u, d, t)$; threshold $\omega$; $\forall l \in L$, $\varphi(l)$ is given; $\mathcal{H}(u)$
Output: Updated $\varphi(.)$ and $\mathcal{H}(.)$
begin
    foreach $(s, t^{'}) \in \mathcal{H}(u)$ do
        if $t - t^{'} \leq \omega$ then
            if $\exists (d, B(s, d)) \in \varphi(s)$ then
                replace $(d, B(s, d)) \in \varphi(s)$ with $(d, B(s, d) \cup \{u\})$
            end
            else
                add $(d, \{u\})$ to $\varphi(s)$
            end
        end
        else
            $\mathcal{H}(u) \leftarrow \mathcal{H}(u) \setminus \{(s, t^{'})\}$ ;  // Too old to be a bridging
             visitor
        end
    end
    if $\exists (d, t') \in \mathcal{H}(u)$ then
        replace $(d, t')$ with $(d, t)$
    end
    else
        add $(d, t)$ to $\mathcal{H}(u)$
    end
end
```

Proposition B.2. *For the M_{DA} model, the time required to produce $\phi(S)$ from $\{\varphi(l) | l \in S\}$ for given threshold τ and set of locations S is $\mathcal{O}(|S||L||U|)$.*

Proof. Every location can have influence on maximally $|L|$ locations with the bridging visitor set size at most $|U|$. Hence, to produce $\phi(S)$, the union of sets of size $|U|$ has to be taken at most $|S||L|$ times, thus, the time complexity is $\mathcal{O}(|S||L||U|)$. □

Extending for Relative models. For the relative models, we additionally have to maintain the total number of unique visitors per location, which can be done in the worst case time $O(log(|U|))$ and space $O(|U|)$ per location and hence does not affect the overall complexity.

Extending for Friends based bridging visitors. For this scenario, we assume the friendship graph is given as an adjacency list $< u, u_{friends} >$

Algorithm B.2: Influence Probabilities among users

```
Input: List of activities $A$, time threshold $\omega$, Friendships $F$
Output: $influenceEdges$
begin
    $influenceEdges = Map()$ , $userActs = Map()$ ,
     $influenceActs = Map()$
    $A_g(l)$= Group activities by location and sort by time
    foreach A(l) $\in A_g(l)$ do
        $currentSet = \phi$
        $followersSet = \phi$
        foreach $(u, l, t_u) \in A(l)$ do
            increment $userActs(u)$
            $parents = \phi$
            foreach $v : (v, l, t_v) \in currentSet$ & $(u, v) \in F$ do
                if $(v, u, t_v) \notin followerMap$ then
                    if $t_u - t_v \geq \omega$ then
                        increment $influenceActs(v \rightarrow u)$
                        add $(v, u, t_v)$ to $followersSet$
                        add $v$ to $parents$
                    end
                end
            end
            foreach $v \in parents$ do
                update $influenceEdges < (v, u), p_{v,u} >$
            end
            add $(u, l, t_u)$ in $currentSet$
        end
    end
end
```

indexed by u. Hence, whenever we add u in the bridging visitor set (line 7 and 9 in Algorithm B.1), we just have to add all the friends of the user u in the set of bridging visitors. As the number of friends is bounded by $|U|$, we get:

Proposition B.3. *For Algorithm B.1, the time required to process an activity for Friends based bridging visitors is $\mathcal{O}(\omega|U|)$. The memory required to maintain the summary is the same as for the M_D, $\mathcal{O}(|L|^2|U|)$.*

Proof. Every user can have maximally $|U|$ friends and hence adding them in the bridging visitor set would take $|U|$ time. There are maximum ω location in the visit history of a user, thus, bridging visitors of ω locations would be updated giving a total time complexity of $\mathcal{O}(\omega|U|)$. $\square$

Extending for Influenced Friends based bridging visitors. In this case, we first compute the influence probabilities of users among each other. The influence probabilities are computed using the algorithm B.2. In this algorithm, first, we initialize $influenceEdges$ that stores the influence probabilities for each pair of influential and influenced users, $userActs$ that maintains the count of activities for users, and $influenceActs$ that tracks the number of influential activities of users among each other. We then group the activities based on locations (Line 5). We iteratively process all the activities that are performed at a location (line 9). In line 13, we ensure that a user is not influenced multiple times by the same activity. We consider the activities influential if they the time difference in following the activities is less than the window threshold (line 14). In lines 18-19, we compute/update the influence probability by which a user is influenced using the Bernoulli equation based on the number of influential activities, all activities and the influential users. The influence probabilities are stored in a hash-map.

Next, we use the influence probabilities for adding the influenced friends of bridging visitors for every pair of locations (influential-influenced locations). The pseudo code for the algorithm is given in Algorithm B.3. It is worth noting that this is an off-line algorithm as we need to process all the activities first using Algorithm B.1. After that, we process a complete bridging visitor set of a location pair at a time(line 6-16 in Algorithm B.3). To do that, for each user in a bridging visitor set, we first fetch her influenced users with corresponding influence probabilities (lines: 10-12). Then, we compute the cumulative influence probability for each influenced user by adding the influence probabilities of the influenced user with her every influential user in the bridging visitor set. The influenced users with cumulative influenced probability greater than the minimum influence threshold (θ) (given in Equation B.2) are added in the set of bridging visitors (lines: 13-16). The same procedure is followed for adding influenced friends for V_D in the case of M_R.

Proposition B.4. *For Algorithm B.1, the space complexity for M_I is the same as for M_F i.e, $\mathcal{O}(|L|^2|U| + \omega + |U|^2)$, and the time required to compute influence oracle for the influenced friends-based model is $\mathcal{O}(\omega|U|\log(|U|)|A| + |L|^2|U|^2 + |A|(|A| + |U|))$.*

Proof. A user can at most influence all of her friends which is equivalent to adding all friends of users in a bridging visitor set, as we do in M_F. Thus, the space complexity for M_I is same as for M_F, i.e., $\mathcal{O}(|L|^2|U| + \omega + |U|^2)$.

For computing influence oracle for M_I, we further need to find the influenced friends of bridging visitors. Thus, we first need to compute the influence probabilities among users. To do that we group the activities on

Algorithm B.3: Exact Oracle for Influenced Friends: Updating complete location summaries

```
1  Input: $A$ all activities, $\omega$ time window, $\theta$ minimum influence
     threshold, $F$ friendships
2  Output: Complete location summary $\varphi(l)$ for all $l \in L$
3  begin
4      $InfluenceEdges$= Influence Probabilities among users($A, \omega, F$) ;
         // Algorithm B.2
5      Run Algorithm B.1 $\forall(u, d, t) \in A$ ;  // This will generate $\varphi(l)$
         for all $l \in L$
6      foreach $l \in L$ do
7          foreach $s \in \varphi(l)$ do
8              $InfluencedFriends \leftarrow \phi$
9              $InfluentialBVs \leftarrow \phi$
10             foreach $(u, v, P_{u,v}) \in InfluenceEdges$ do
11                 if $x \in B(l, s)$ then
12                     add $(u, v, p_{u,v})$ to $InfluentialBVs$
13                 end
14             end
15             foreach $v : (u, v, P_{u,v}) \in InfluentialBVs$ do
16                 if $(Sum(P_{u,v}), \forall u : (u, v, p_{u,v}) \in InfluentialBV) \geq \theta$
                     then
17                     add $v$ to $InfluencedFriends$
18                 end
19             end
20             $B(l, s) = B(l, s) \cup$ influenced friends
21         end
22     end
23 end
```

the basis of location and then sort the activities performed at a location in a chronological order. Then, for each location, we iteratively consider all the activities to evaluate the influence relationship of users who performed them among each other and update their corresponding influence scores. As each activity is evaluated with every other activity thus in total we have $|A| * |A|/2$ iterations. We further traverse all influential users to assign them their credit, which can at most be $|U|$. The influence probabilities are computed once and stored in a hashmap. To add influenced friends in a bridging visitor set, we need to fetch the influenced friends and influence probabilities for every user in the bridging visitor. The time to fetch the influenced visitors is constant. Thus, the time to add the influence visitors for a set of bridging visitors which at most can be $|U|$ is $|U| * |U|$ and thus, for each location pair, it is $|L|^2|U|^2$. This makes the overall time complexity for computing oracle for M_I as $\mathcal{O}(\omega|U|\log(|U|)|A| + |L|^2|U|^2 + |A|(|A| + |U|))$. □

	Exact			Approx		
	Memory	Oracle Time	Query Time	Memory	Oracle Time	Query Time
M_{DA}	$O(\|U\|(\|L\|^2+\omega))$	$O(\omega\log(\|U\|)\|A\|)$	$O(\|S\|\|L\|\|U\|\|U\|)$	$\mathcal{O}(\|L\|^2b+\|U\|\omega)$	$O(\omega\|A\|)$	$O(\|S\|\|L\|b)$
M_{DR}				$\mathcal{O}(\|L\|^2b$ $+\|U\|\omega+\|L\|\|U\|)$		
M_{FA}	$O(\|U\|(\|L\|^2+\omega$ $+\|U\|))$	$O(\omega\|U\|\log(\|U\|)\|A\|)$		$\mathcal{O}(\|L\|^2b$ $+\|U\|\omega+\|U\|^2)$	$O(\omega\|U\|\|A\|)$	
M_{FR}				$\mathcal{O}(\|L\|^2b$ $+\|U\|\omega+\|U\|^2$ $+\|L\|\|U\|)$		
M_{IA}		$O(\omega\|U\|\log(\|U\|)\|A\|$ $+\|L\|^2\|U\|^2$ $+\|A\|(\|A\|+\|U\|))$		N/A		
M_{IR}						

Table B.2: Summary of time and space complexities for the influence models.

4.2 Approximate Influence Oracle for M_D and M_F

In the worst case the memory requirements of the exact algorithm presented in the last section are quite stringent: for every pair of locations (s, d), in $\varphi(s)$ the complete list of bridging visitors from s to d is kept. Therefore, here we present an approximate algorithm for maintaining the complete location summaries in a more compact form. This compact representation leads to a significant saving especially in those cases where the window size ω is large since in that case the number of bridging visitors increases.

We observe that when computing the number of bridging visitors between s and d we do not need the exact set of bridging visitors between s and d, but only the cardinality of that set. For the relative number of bridging visitors, we additionally need only the numbers of visitors $|V(s)|$. Furthermore, as per Equation B.5, in order to find the accumulated complete location summary, we need to combine two complete location summaries; for instance: the complete location summary $\varphi(\{s_1, s_2\})$ is obtained by taking the following pairwise union of $\varphi(s_1)$ and $\varphi(s_2)$: if $\varphi(s_1)$ and $\varphi(s_2)$ respectively contain the pairs $(d, B(s_1, d))$ and $(d, B(s_1, d))$, then $\varphi(\{s_1, s_2\})$ contains $(d, B(s_1, d) \cup B(s_2, d))$. But then again, for further computations, we only need the cardinality of the bridging visitor sets. Hence, if we accept approximate results, we could replace the exact set $B(s, d)$ with a succinct sketch of the set that allows to take unions and get an estimate of the cardinality of the set. Please note that we can approximate the bridging visitor set $B(s, d)$ only for the direct and friends-based bridging visitor sets. For the Influenced Friends based bridging visitor set, we need the exact set as we need to know all the users in $B(s, d)$ to find the set of Influenced friends. Therefore, the approx algorithm is only for the direct and friends-based influence models.

In our approx algorithm, we use the HyperLogLog sketch (HLL) [11] to replace the exact sets $B(s, d)$ and $V(s)$. The HLL sketch is a memory-efficient data structure of size 2^k that can be used to approximate the cardinality of a set by using an array. The constant k is a parameter which determines the accuracy of the approximation and is in our experiments in the order of 6 to 10. Furthermore, the HLL sketch allows unions in the sense that the HLL sketch of the union of two sets can be computed directly from the HLL sketches of the individual sets. For our algorithm, we consider the HLL algorithm as a black box. By using HLL, we not only reduce memory consumption but also improve computation time, because adding an element in an HLL sketch can be done in constant time and taking the union of two HLL sketches takes time $\mathcal{O}(2^k)$; that is: the time to take the union of two sets is independent of the size of the sets.

Proposition B.5. *Let $b = 2^k$ be the size of the HLL sketch. For the M_D and M_F models, the time needed to process an activity using the HLL sketch to maintain $B(s, d)$ is $\mathcal{O}(\omega)$. The memory required to maintain the complete location summary $\{\varphi(l) | l \in L\}$ is $\mathcal{O}(|L|^2 b)$. The memory requirement for the visit history $\{\mathcal{H}(u) | u \in U\}$ will remains $\mathcal{O}(|U|\omega)$ as in the exact algorithm mentioned in Proposition B.1.*

Proof. Adding an element in a HLL set takes constant time, hence, to process the activity HLL set of ω locations will be updated in $\mathcal{O}(\omega)$. The size of the HLL set is b irrespective of the number of elements in the set and thus, the memory required to store $\varphi(l)$ is $\mathcal{O}(|L|b)$. Hence, for all locations the memory required is $\mathcal{O}(|L|^2 b)$. □

4.3 Single-Influencer based Influence Oracle

In this subsection, we go one more step further and develop an even more efficient algorithm for a very special case. In real life, there may be situations in which even one information carrier can spread information among locations. Examples may include infections or information items in highly specialized information networks with confidential information. Moreover, LBSN data is often sparse, thus, usually, has a very low number of influence carriers. These situations may also have been created artificially by lumping together multiple traces for reasons of privacy; in such a situation a single visit trace may actually correspond to multiple visitors. Thus, in such situations, we may have to rely on single carriers as a proxy for larger unobserved streams of people.

In this section, for this special case, we provide two approximate but more efficient algorithms; an on-line algorithm called *On-Sin* and an off-line but far more efficient algorithm called *Off-Sin* for solving the influence oracle problem.

On-Sin approach

The **On**-line **S**ingle influencer based **in**fluence oracle (*On-Sin*) approach is based upon the simple observation that for $\tau = 1$, we do not need to maintain the bridging visitor set $V_B(s, d)$ for locations s and d, because even one visitor implies influence and hence the location influence set $\phi(s)$ can be directly maintained by adding d whenever a user u visits d within ω time after visiting s. Hence, in the special case we do not need to maintain the complete location summary $\varphi(s)$ and directly maintain $\phi(s)$.

Furthermore, in order to find the most influential locations, we just need the cardinality of the location influence summary. Hence, we can replace $\phi(s)$ by a HyperLogLog(HLL) sketch. The memory required to store $\{\phi(l)|l \in L\}$ is $\mathcal{O}(|L|b)$ as for every location we just keep a HLL sketch. Please note that the time required to process an activity still remains the same at $\mathcal{O}(\omega)$ as in the worst case we still need to iterate over ω locations in the user visit history. Next, we provide an approach in which we could actually store an approximation of the user visit history to provide tremendous speedups.

Off-Sin approach

The **Off**line **S**ingle influencer based **in**fluence oracle (*Off-Sin*) algorithm is based on the observation that while processing an activity (u, s, t), if we know all the future locations u will visit during time t to $t + \omega$, then we can directly add those locations in the location influence set $\phi(s)$. In order to achieve this, we process all the activities in reverse order of time. As we are going reverse in time we cannot run this algorithm incrementally for new activities hence it is an off-line algorithm. A simple case is shown in the following example.

Algorithm B.4: Off-Sin: Location Influence summary by using modified HLL for $\tau = 1$

```
1  Input: Activity list $A$.
2  Output: $\phi(s)\ \forall s \in L$
3  begin
4      Sort $A$ in decreasing order of time.
5      $\phi(s) \leftarrow HLL \quad \forall s \in L$
6      $u_f \leftarrow vHLL \quad \forall u \in U$
7      foreach $(u, s, t) \in A$ do
8          $\phi(s) \leftarrow \phi(s) \cup subset(\mathcal{H}_f(u), t, \omega)$
9          $\mathcal{H}_f(u).add(s, t)$
10     end
11 end
```

	$t = 5$	$t = 3$	$t = 2$	t=1
Activity:	$(d, H_2, 5)$	$(i, H_1, 3)$ $(d, M_1, 3)$	$(i, M_1, 2)$ $(d, H_1, 2)$	$(i, H_2, 1)$ $(d, H_2, 1)$
$\mathcal{H}_f(d)$:	$\{(H_2, 5)\}$	$(H_2, 5)$, $(M_1, 3)$	$\{(M_1, 3)$, $(H_1, 2)\}$	$\{(M_1, 3)$, $(H_1, 2)$, $(H_2, 1)\}$
$\mathcal{H}_f(i)$:	$\{\}$	$\{(H_1, 3)\}$	$\{(H_1, 3)$, $(M_1, 2)\}$	$\{(H_1, 3)$, $(M_1, 2)$, $(H_2, 1)\}$
$\phi(H_1)$:	$\{\}$	{}	$\{M_1\}$	$\{M_1\}$
$\phi(H_2)$:	$\{\}$	$\{\}$	$\{\}$	$\{H_1, M_1\}$
$\phi(M_1)$:	$\{\}$	$\{H_2\}$	$\{H_2, M_1\}$	$\{H_2, M_1\}$

Fig. B.3: An example of updating location influence set for locations H_1, H_2 and M_1 for $\tau = 1$ and $\omega = 2$ by processing data in reverse order of time using Off-Sin algorithm.

Example 4.2
Consider the activities of users i and j given in Figure B.1. We process the activities of these users in reverse order of time. Figure B.3 shows the update of $\phi(l)$ and $\mathcal{H}_f(u)$ for each activity. For sake of understanding, we represent the exact sets in the example but for the efficient algorithm the sets $\phi(s)$ and $\mathcal{H}_f(u)$ are approximated with HLL and vHLL sets respectively. At time $t = 3$, location H_2 is added into influence set $\phi(M_1)$ as $(H_2, 5)$ was in $\mathcal{H}_f(d)$ and hence is within time window 2. $\mathcal{H}_f(d)$ and $\mathcal{H}_f(i)$ is updated as well. Note at time $t = 2$, $\mathcal{H}_f(d)$ is pruned and $(H_2, 5)$ is removed as it is out of window from current time.

Now, instead of the visit history $\mathcal{H}(u)$, we maintain the future visit history represented by $\mathcal{H}_f(u)$. At time t, $\mathcal{H}_f(u) = \{(s, t') | t' \geq t,\ t' - t \leq \omega, (u, s, t') \in A,\ \nexists t'' : t \leq t'' < t' \wedge (u, s, t'') \in A\}$. That is, the future visit history $\mathcal{H}_f(u)$ of a user u, maintains every location a user u visits in future and the earliest time in future the user visits that location. We can see that adding all locations $\mathcal{H}_f(u)$ that will be visited by u in $\phi(s)$ is much more efficient than adding s to $\phi(d)$ for all locations d in $\mathcal{H}(u)$. This is because now for every activity (u, s, t), instead of updating summaries of all locations in $\mathcal{H}(u)$ we need to just update the summary of s by merging it with $\mathcal{H}_f(u)$. Furthermore, we do not need the individual locations anymore in history, but only their cardinality. Thus, we can approximate the set $\mathcal{H}_f(u)$.

Now, while processing an activity (u, s, t), we update $\phi(s)$ and add all the locations, d in $\mathcal{H}_f(u)$ for which $\mathcal{H}_f(u) - t \leq \omega$. We do not need to iterate over the elements of the set $\mathcal{H}_f(u)$ but just need a subset of $\mathcal{H}_f(u)$ to get elements added during current time and a time window ω. Using

	Off-Sin			**On-Sin**		
	Memory	**Time Oracle**	**Time Query**	**Memory**	**Time Oracle**	**Time Query**
M_A	$\mathcal{O}(b(\|L\| + \|U\| \log \omega))$	$O(b \log(\omega)\|A\|)$	$O(\|S\|b)$	$\mathcal{O}(b\|L\| + \|U\|\omega)$	$O(\omega\|U\|\|A\|)$	$O(\|S\|b)$
M_R	$\mathcal{O}(b(\|L\| + \|U\| \log \omega) + \|L\|\|U\|)$			$\mathcal{O}(b(\|L\| + \|U\| \log \omega) + \|L\|\|U\|)$		

Table B.3: Time and Space complexities for Single influencer-based Influence.

a versioned HyperLogLog sketch (vHLL) [27], we can achieve such a time window based approximate set with much less memory and time requirements. vHLL is an extension of the HLL data structure which approximates a set and allows to get the cardinality of the set based on a specified time window. While adding an element, vHLL also maintains the time of addition of the element. vHLL provides a *subset* function which takes time t and window ω as an input and produces an HLL representation of a set. This set consists of elements that were added in vHLL during time t and $t + \omega$.

Algorithm B.4 represents the *off-Sin* batch algorithm. We go through activities list A in reverse order. We treat the activities with later time stamps working our way from the end to the start of the log. The functions $subset(\mathcal{H}_f(u), t, \omega)$ at line 8 and $\mathcal{H}_f(u).add(d, t)$ at line 9 are functions provided by vHLL.

Proposition B.6. *The time required to process an activity by Off-Sin in Algorithm B.4 is $\mathcal{O}(b \log(\omega))$. The memory requirements to maintain $\mathcal{H}_f(u)$ improves from $\mathcal{O}(\omega)$ to $\mathcal{O}(b \log(\omega))$.*

Proof. The time required to process an activity will be equal to the time required to find subset in line 8 and then updating $\mathcal{H}_f(u)$ at line 9. According the time complexity of vHLL given by Kumar et al. in [27], the add function takes $\mathcal{O}(\log \omega)$ time and the subset function takes $\mathcal{O}(b \log(\omega))$ time. The subset function returns a HLL sketch and $\phi(s)$ is also a HLL sketch, the union of two HLL sketches takes $\mathcal{O}(b)$ time. Hence, the total time complexity of Algorithm B.4 is $\mathcal{O}(b \log(\omega) + \log(\omega) + b) = \mathcal{O}(b \log(\omega))$. $\mathcal{H}_f(u)$ is a vHLL set and as given by Kumar et al. in [27] the memory required by a vHLL set is $\mathcal{O}(b \log(\omega))$. $\square$

5 Location Influence Maximization

In this section, we show that the influence oracle can be used for finding the most influential locations. We utilize the influence oracle and apply the standard greedy algorithm to compute top-k as obtaining an exact solution is intractable as the next proposition states.

Proposition B.7. *The following problem is* **NP***-hard for all influence models: given an LBSN and bounds k and β, does there exist a set of locations S of size k such that $|\phi(S)| \geq \beta$.*

Proof. **NP**-hardness follows from a reduction from set cover. Consider an instance $\mathcal{S} = \{S_1, \ldots, S_m\}$ with all $S_i \subseteq \{1, \ldots, n\}$ and bound k of the set cover problem: does there exist a subset $\mathcal{S}'$ of $\mathcal{S}$ of size at most k such that $\bigcup \mathcal{S}' = \{1, \ldots, n\}$. We reduce this instance to a LBSN as follows: $L = \{l_1, \ldots, l_n\} \cup \{s_1, \ldots, s_m\}$, $U = \{u_1, \ldots, u_m\}$, $F = \emptyset$, $A = \{(u_i, s_i, 0) \mid i = 1 \ldots m\} \cup \{(u_i, l_j, j) \mid i = 1 \ldots m, j \in S_i\}$. That is, every element j of the domain $\{1, \ldots, n\}$ is associated to a location l_j, and for every set S_i we introduce a location s_i visited by user u_i at time 0. Furthermore, user u_i visits all locations l_j such that $j \in S_i$ at time stamp j. If we use the absolute model with $\tau = 1$ and $\omega \geq n+1$, for $i = 1 \ldots m$, $\phi(\{s_i\}) = \{l_j \mid j \in S_i\}$. As such there exists a set cover of size k if and only if there exists a set of locations S of size k such that $|\phi(S)| = n$. □

Recall that the influence of a set of locations S is computed by accumulating the effect of all locations in S. It is hence possible that two locations s and s' separately do not influence a target location d because individually they have too few bridging visitors to d, but together they reach the threshold. This situation occurs for instance in Figure B.1, for the locations H_2 and M_1. These locations individually do not reach the threshold to influence H_1 for $\tau_A = 2$ and $\omega = 1$. However, together they do. One inconvenient consequence of this observation is that the influence function that we want to optimize is not sub-modular [28]. Indeed, in the example above, adding H_2 to the set $\{M_1\}$ gives a higher additional benefit (1 more influenced location) than adding H_2 to $\{\}$. Therefore, we do not have the usual guarantee on the quality of the greedy algorithm for selecting the top-k.

The main reason that we do not have the guarantee is that the benefit is not gradual; before the threshold is reached it is 0, after the threshold is reached it is 1. This means that a location that has $\tau - 1$ bridging visitors to 1000 other locations each, gives the same benefit as a location that does not have any bridging visitors. Clearly, nevertheless, the first location is more likely to lead to a good solution if later on additional locations are selected. Therefore, we would like to incorporate potential future benefits into our objective function. Thus, in order to compute the influence of a location, we consider locations that are influenced as well as those locations that are not yet influenced but have potential to be so in future. To characterize the potential of future benefit in combination with the number of influenced locations, we use the following formula:

$$LI(S) = (1 - \alpha) \times |\phi(S)| + (\alpha) \times \sum_{d \in L - S} (\min\{|B(S,d)|, \tau\}) \tag{B.7}$$

In this formula, $\alpha = [0, 1]$ represents a trade-off between the number of influenced locations and a reward for potentially influenced locations. For relative models, we replace the $|B(S, d)|$ with $|B(S, d)|/|V(d)|$.

Next, we apply a greedy method on the basis of location influence to find top-k locations. We start with an empty set S of locations and iteratively add locations to it until we reach the required number of top elements: k. We start with an empty set S of locations and iteratively add locations to it until we reach the required number of top elements: k. In each step, for each location $s \in L$, we evaluate the effect of adding s to S, and keep the one that gives the highest benefit $LI(S)$. Then, we update $S \leftarrow S \cup \{l\}$.

Example 5.1
Consider the case in Figure B.2 for $\omega = 1$, $\varphi(H_2) = \{(H_1, \{d\}), (M_1, \{i\})\}$, $\varphi(M_1) = \{(H_1, \{i\})\}$ and $\varphi(H_1) = \{(M_1, \{d\})\}$. We aim to find top-2 locations in this example with $\alpha = 0.1$ and $\tau = 2$. During the first iteration, $LI(H_2) = 0.9 \times 0 + 0.1 \times (1 + 1) = 0.2$, because H_2 does not completely influence any other location, however H_1 and M_1 are potentially influenced locations for the bridging visitors d and i, respectively. Similarly, $LI(M_1) = 0.1$ and $LI(H_1) = 0.1$. Thus, we choose H_2 as first seed as it has maximum value. In the next iteration, we first combine the seed H_2 with M_1 and compute the combined influence. Here, $LI(\{H_2, M_1\}) = 0.9 \times 1 + 0.1 \times (2) = 1.1$. Similarly, $LI(\{H_2, H_1\}) = 1.1$. Since M_1 and H_1 provide equal benefit of 0.9 when combined with H_2, we can randomly choose either M_1 or H_1 as second seed.

6 LBSN Data Analysis

The influence models of Section 3.2 have several parameters to set: τ, ω, and θ. Furthermore, while defining the friendship-based bridging visitors, and influence-friends based bridging visitors, the assumption was made that friends tend to follow friends. Before going to the experiments, in this section, we show how to set the thresholds with reasonable values based on an analysis of the LBSN datasets given in Table B.4 and verify and confirm the friendship assumption.

Datasets. We used 3 real-world datasets : `FourSquare` [29], `BrightKite`, and `Gowalla` [30]. Each dataset consists of two parts: the friendship graph and an ordered list of check-ins. A check-in record contains the user-id, check-in time, GPS coordinates of location, and a location-id. The statistics of the datasets are given in Table B.4.

Data Prepossessing. The real-life datasets required preprocessing because many locations are associated with multiple location identifiers with

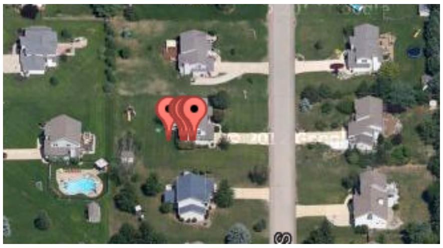


Fig. B.4: [10]An example of ambiguous location ids: GPS coordinates of 13 location-ids in `FourSquare` corresponds to a single, unique location on GoogleMaps

	Users	Locations	Check-ins	POIs
`FourSquare`	16K	803K	1.928M	582K
`BrightKite`	50K	771K	4.686M	631K
`Gowalla`	99.5K	1.257M	6.271M	1.162M

Table B.4: Statistics of datasets: number of users, location, visits and clustered locations/POIs

slightly different GPS coordinates. Consider, for instance, Figure B.4. In this figure, 13 GPS coordinates that appear in the `FourSquare` dataset are shown which correspond to different locations Ids in the dataset, but which clearly belong to one unique location. In order to resolve this issue, we clustered GPS points to get POIs. We used the density-based spatial clustering algorithm [31] with parameters *eps*=10 meters and *minpts*=1 to group the GPS points. New location Ids were assigned to each cluster for all 3 datasets. The statistics of the new Ids are reported in column POIs of Table B.4.

6.1 Setting parameters for the influence models

In order to determine the value of influence window threshold ω, we measured the time difference between consecutive visits of users to distinct lo-

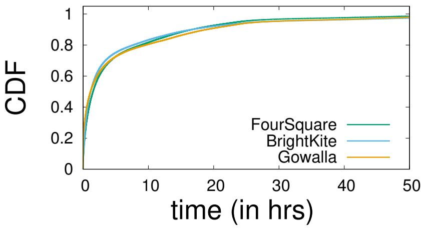
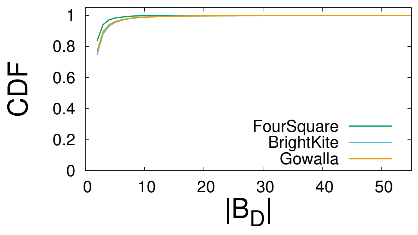
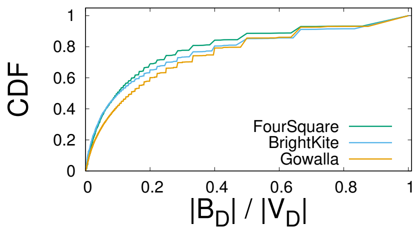


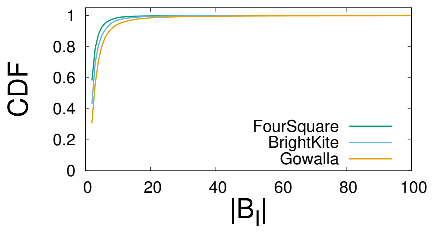
Fig. B.5: CDF of time difference between consecutive visits of users to distinct locations



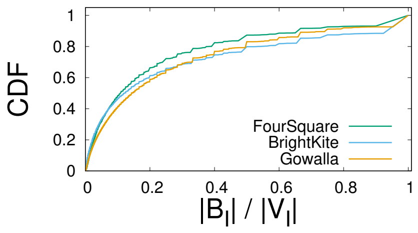
(a) M_{DA}



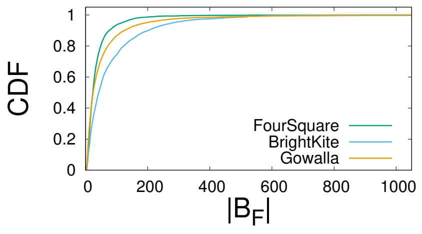
(b) M_{DR}



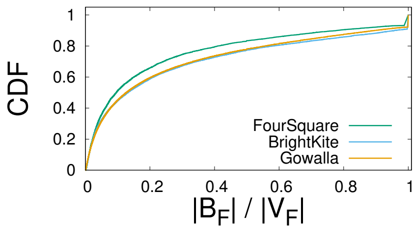
(c) M_{IA}



(d) M_{IR}



(e) M_{FA}



(f) M_{FR}

Fig. B.6: Cumulative distribution function (CDF) of thresholds for corresponding influence models. All locations pairs having at least one bridging visitors are considered and CDF values of the threshold based on the bridging visitors of these locations and visitors of the destination locations are plotted.

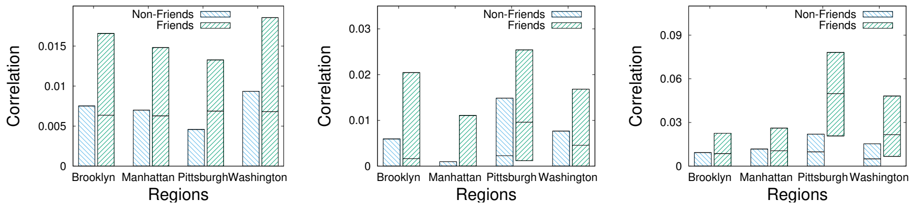
cations. The cumulative distribution functions (CDF) for three LBSNs are given in Figure B.5. It can be seen that for all LBSNs in our study, 80% of the consecutive activities are performed within 8 hours. After that, there is only a moderate increase in the number of activities with respect to the time interval. Thus, in order to capture only the most common activities, we keep

$\omega = 8$. However, it can, of course, be changed if the data distribution is different, or there are different user or application requirements.

Next, we find suitable values for the thresholds of the influence models. In order to do that, we considered all the influence models, i.e., $M_{DA}, M_{DR}, M_{FA}, M_{FR}, M_{IA}$, and M_{IR}, for each pair of locations with at least one bridging visitor. The cumulative distribution functions for each of these numbers are depicted in Figure B.6. We can utilize the CDF values for controlling the number of influences in the dataset, and thus also for finding the suitable values for thresholds in our models. The values of the thresholds are an application-dependent choice and can be considered accordingly. For example, if an application requires finding many influential relationships and indirectly many influential and influenced locations, then a lower threshold should be considered and vice versa. In this paper, we consider the top 20% influential relationships among locations for all the models. To do that for each influence model, we consider the CDF value of 0.8 (100%-20%=80%) as its threshold. Therefore, the values of $\tau_{DA}, \tau_{DR}, \tau_{IA}, \tau_{IR}, \tau_{FA}$ and τ_{FR} are $2, 0.6, 4, 0.6, 120$ and 0.6, as shown in Figures B.6a, B.6b, B.6c and B.6d, B.6e and B.6f, respectively.

6.2 Mobility analysis of friends

In real life, usually activities of friends are more similar than activities of non-friends. In LBSNs, this implies that a visit of a user to a location increases the chances of visits of her friends to the same location. We considered this assumption when constructing our friendship-based bridging visitors and influenced-friends based bridging visitors in Section 3.1 and Section 3.1, respectively. We now show the correctness of this assumption by computing the correlations between activities of users, their friends and non-friends: Let L_u and L_v be the locations visited by users u and v, respectively. The correlation between activities of u and v is measured by the Jaccard Index [32] between L_u and L_v given by $|L_u \cap L_v|/|L_u \cup L_v|$. The average correlation of activities of users and those of their friends is denoted *friendship correlation* (p^{f}_{corr}), and the average correlation between activities of users and their non-friends is denoted *non-friendship Correlation* (p^{nf}_{corr}). In order to avoid an unreasonable bias due to the fact that friends tend to live in the same city, we restricted our computation of the average non-friendship correlation to users in the same city. We picked four regions of the United States, i.e., Brooklyn, Manhattan, Pittsburgh, and Washington and considered the activities of users in these regions to study the correlations. The statistics of p^{f}_{corr} and p^{nf}_{corr} of all the users are given in Figure B.7. The figure presents boxplots without outliers. It can be seen that the median of p^{f}_{corr}, even though still small, is up to 5 times larger than of p^{nf}_{corr}. The same pattern is observed for all the datasets. This



(a) FourSquare **(b)** BrightKite **(c)** Gowalla

Fig. B.7: Jaccard index based correlations of activities of friends and non-friends users.

validates our assumption that the activities of friends are more similar than those of non-friends.

Although the activities of friends are more correlated than those of non-friends, even between friends the correlation between the sets of locations they visited is low. Thus, in order to tackle this, we considered only the potential influenced friends rather than all friends. To do that we computed the influence probabilities of users among each other using the method given in Section 3.1. We plotted the CDF of these influence probabilities and found a suitable value for the influence threshold, as shown in Figure B.8. The value of the influence threshold depends on the application. For example, a higher value would be given to θ if a stronger relationship among influential and influenced users is expected. In this paper, we assume that users having influential probabilities in the top 20% will follow the influential users within ω. Thus, by default, we consider $\theta = 0.2$. However, the insights for location influence with respect to different values of θ are also shown in Section 7.2.

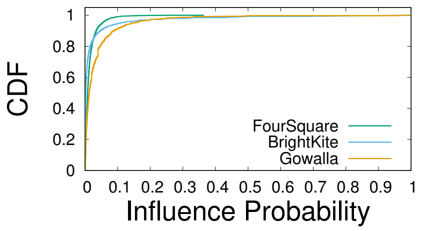
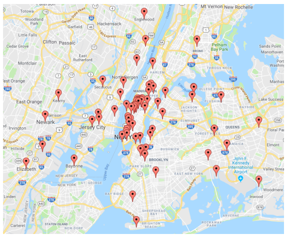
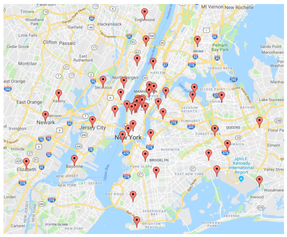
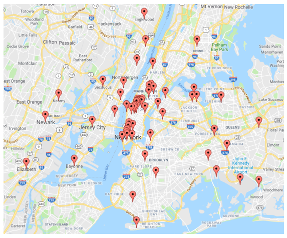
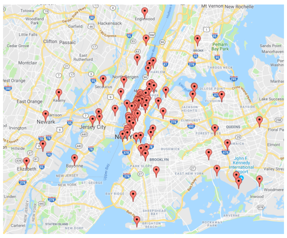


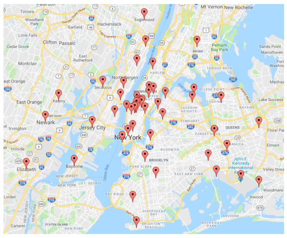
Fig. B.8: CDF of Influence Probabilities of all the pairs of influential and influenced users.

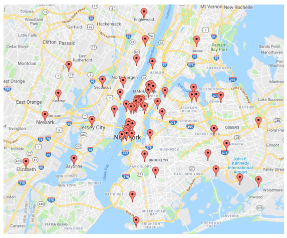

(a) M_{IA}: k=10, spread=78 locations

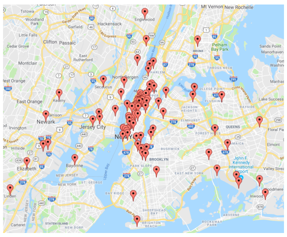

(b) IRS: k=10,spread= 53 locations

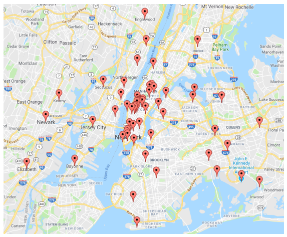

(c) $IRS - window$: k=10, spread=62 locations


(d) M_{IA}: k=15, spread=92 locations


(e) IRS: k=15, spread=55 locations


(f) $IRS - window$: k=15, spread=63 locations


(g) M_{IA}: k=20, spread=102 locations


(h) IRS: k=20, spread=63 locations

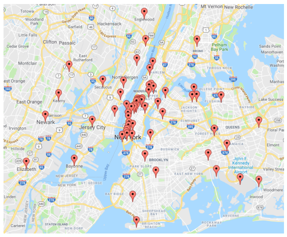

(i) $IRS - window$: k=20, spread=66 locations

Fig. B.9: Influenced locations w.r.t. to different number of top-k influential locations fetched by M_{IA}, IRS, and $IRS - window$ in NYC for `BrightKite`.

7 Evaluation

In this section, we evaluate the notions defined in Section 3 and the algorithms introduced in Section 4 and Section 5, respectively.

Experiment settings. We conducted our experiments on a Linux machine with 4 AMD Opteron 6376 processors with 2.3GH and 512 GB RAM. The algorithms [1] are implemented in Scala. The description and preprocessing of the datasets used for the experiments are given in Section 6.

[1]Code of the algorithms are given at: https://github.com/rohit13k/LBSNAnalysis

Base-line competitors: To evaluate the effectiveness, we consider the most relevant state-of-the-art influence maximization method, called *Influence Reachability Set (IRS)* [27]. For each node u, IRS fetches all the nodes which are reachable from u based on the temporal path [33] within a given time window. Once the influence reachability sets of all the nodes are obtained, the top-k most influential nodes are found using the standard greedy algorithm such that the combined influence reachability set size of these nodes is the maximum. We consider this algorithm for comparison with our models due to following common features: 1) the model of capturing interactions among users based on their activities, 2) consideration of temporal sliding window for influence estimation, 3) using a standard greedy algorithm for finding top-k influential nodes. The IRS method is used for finding the influence of users among each other based on their activities. However, in our approach, we consider bridging visitors for constructing relationships among locations. In order to evaluate IRS on LBSN data, we propose the two methods below for constructing location interaction graphs. We then run the IRS algorithm for finding top-k influential locations.

- **Location interaction graph:** In this approach, we use the check-in data to generate a graph of interactions between locations. An interaction graph is a graph where the edges between the vertices are timestamped and represents an interaction between the node at that time-stamp. In LBSN data, we consider two locations l_1, l_2 are interacting at time stamp t when a user does consecutive check-ins (l_1, t') and (l_2, t). The method of finding top-k influential locations using IRS on location interaction graph is denoted by *IRS*.

- **Time window based location interaction graph:** Usually, influence remains active within a particular time. Hence, we consider this observation for constructing interactions of locations from the LBSN data. We consider an interaction between two locations l_1, l_2 at time stamp t if a user u performs consecutive check-ins (l_1, t') and (l_2, t) such that $t - t' \leq \omega$. The method of finding top-k influential locations using IRS on time window based location graph is denoted by $IRS - window$.

Location correlation graph with PageRank: In this approach, we construct a location correlation graph [4] for a given LBSN. An edge between two locations in a location correlation graph exists if the number of common visitors between them is greater than a threshold. Since, we take the value of $\tau = 2$ for the absolute influence model so, we take the same value, i.e., 2 for the threshold. We then use the Jaccard index to compute the edge weights based on the common visitors and the visitors of the locations. Once the location correlation graph is created, we run the PageRank algorithm to find the top-k locations having the highest PageRank values. This approach is denoted by $PR - LCG$. The idea behind using the location correlation graph is to evaluate the significance of our proposed models for capturing the location influence.

7.1 Qualitative significance of the Location Influence

In order to demonstrate our notion of location influence, we compared the results of our method using the absolute influence friends based bridging visitor model (M_{IA}) with IRS and $IRS - window$. To do that, we considered the activities performed in New York and fetched top-k influential locations for k=10, 15, 20 using M_{IA}, IRS and $IRS - window$. We then plotted the coordinates of the influenced locations of the top-k influential locations using Google Map, as shown in Figure B.9. In the figure, it can be observed that M_{IA} leads to a set of locations with a larger spread as compared to the other two approaches, both geographically and in terms of the number of influenced locations. It can further be seen that the difference of influenced locations increases with an increasing value of k. This shows the significance of our proposed method. All the datasets show similar trends. Thus, due to space limits, the results are only shown for the `BrightKite` because of its median data size and check-in density among all three datasets.

7.2 Comparison of Influence Models

Influence spread prediction: Next, we evaluated the influence spread prediction ability of all the models and compared the results with the baseline approaches. To do that, we divided the activities based on time such that each part is composed of the activities of one month. We computed the seeds on one part of the dataset called training set and found their spread on the next part in the sequence called test set. M_A was used to compute the influence on the test set. To compare the results, we computed and compared the number of influenced nodes for the top-*k* influential locations where k=1, 3, 5, 10, 15, and 20. We iteratively repeated the experiment for all the parts of activities and reported the total spread of all of these iterations for each value of k. The results are shown in Table B.5. Here, it can be observed that our proposed models, outperformed both IRS and $IRS - window$ for all the values of k, for all the three datasets. Further, our proposed models also outperformed $PR - LCG$ for `BrightKite`. We only computed $PR - LCG$ for `BrightKite` due to intensive computation time, i.e., more than 48 hours, required for constructing the location correlation graph. For all the methods, the difference in spread increases with an increasing value of k. Overall, the top-k influential locations fetched by the absolute influence models influenced more locations as compared to the relative influence models. More specifically, the spread of M_{IA} is up to 45% more than M_{DA}, 700% more than M_{FA}, and 400% more than $IRS - window$. The reason is that unlike the M_{DA} and M_{FA}, M_{IA} incor-

Dataset	K	Number of Influenced Nodes								
		Absolute Influence Model			Relative Influence Model			IRS	IRS-window	PR-LCG
		M_{DA}	M_{FA}	M_{IA}	M_{DR}	M_{FR}	M_{IR}			
FourSquare	1	89	37	**98**	66	66	66	79	71	N/A
	3	190	116	**195**	159	160	159	82	105	
	5	230	117	**255**	216	196	216	121	116	
	10	322	159	**361**	273	260	273	124	167	
	15	380	200	**421**	307	288	307	130	174	
	20	432	255	**495**	330	326	330	132	191	
BrightKite	1	662	657	**671**	648	655	648	512	537	657
	3	943	882	**959**	896	882	896	613	743	924
	5	**1,153**	9,67	1,112	1,041	994	1,041	666	835	1,100
	10	1,458	1,140	**1,465**	1,269	1,259	1,269	749	1,027	1,437
	15	**1,717**	1,257	1,686	1,449	1,444	1,449	821	1,171	1,693
	20	1,921	1,381	**1,951**	1,589	1,570	1,589	867	1,275	1,928
Gowalla	1	676	111	**982**	446	405	446	453	613	N/A
	3	**1804**	153	1787	1129	1212	1129	821	1072	
	5	2834	238	**3087**	1772	1716	1772	997	1134	
	10	4875	860	**5197**	3302	3218	3302	1116	1445	
	15	6236	1395	**6767**	4136	3645	4136	1244	1854	
	20	7460	1877	**8165**	4668	4119	4668	1452	2015	

Table B.5: Influence spread of top-k influential locations fetched by the proposed influence models, IRS, $IRS - window$ and $PR - LCG$. The check-ins are divided on monthly basis. The influential keys are fetched on one part and spread is computed on the next part in the sequence. The process is iteratively repeated for all the months and the total influence spread for each value of k, for each dataset is depicted.

porate the friends of bridging visitors that have potential to become bridging visitors in future. The relative models filter out the excessive influence of popular locations thus the influence spread of the models is almost same.

Computational resources. We compared the computation time and memory requirements for the influence models. The results are shown in the Table B.6. Here, it can be observed that the computation time for M_{IA} is more than M_{DA}. The reason is that for the influenced-friends based bridging visitors, we further need to compute the influence probabilities of users among each other which requires more computation time. Since we incorporate the influence-friends of the bridging visitors as well for constructing influence oracle thus more memory is consumed. The computation time and memory consumption of M_{FA} is more than both other models. The reason is that for friends-based bridging visitors, we incorporate all the friends of bridging visitors which largely increases the bridging visitor set. Thus, it

Dataset	Time			Memory		
	M_{DA}	M_{FA}	M_{IA}	M_{DA}	M_{FA}	M_{IA}
FourSquare	596	2,220	1,140	29,562	77,334	29,654
BrightKite	808	4,363	1,259	30,199	228,391	30,602
Gowalla	2,957	Out of Memory	6,973	186,363	Out of Memory	189,195

Table B.6: Computation time and memory for all influence models.

requires more time to process and more memory to maintain the set for constructing influence oracle. The base line competitors, $IRS, IRS - time$ and $PR - LCG$ require more computational resources in comparison with all the influence models. The reason is that an intensive computation is required for constructing the location interaction graph, the time window based location interaction graph, and the location correlation graph for $IRS, IRS - time$, and $PR - LCG$, respectively.

Conclusion: Our proposed models outperformed baseline competitors in terms of influence spread as well as required computational resources. Overall, the absolute models performed better than the relative models. In case of the relative models, M_{DR} should be chosen as it needs minimum computational resources but yields the same influence as other models. For the absolute models, although M_{IA} requires more computational resources, but yields the maximum influence. On the other hand, M_{DA} requires fewer resources but the influence spread is also lower. M_{FA} is the worst in terms of computational resources and influence spread. Thus, the choice of the model among M_{IA} and M_{DA} can be made by considering the trade-off between computational resources and influence spread, i.e., if a higher influence spread is more desirable than computational resources then M_{IA} should be chosen, and vice versa.

7.3 Approximations for M_{DA} and M_{FA}

Next, we analyzed the approximate algorithms for constructing influence oracle for M_{DA} and M_{FA}. We analyzed the impact of approximation on accuracy, computation time and memory. The results are similar for all the datasets and hence we only present results for the smallest and the biggest datasets, i.e., `FourSquare` and `Gowalla`, respectively.

Approximation accuracy. For every location with a non-empty influence set, we used the HLL-based approximate version of the Oracle to predict the size of the influence set. Then the relative error as compared to the real size was computed for every location. In Table B.7 the mean and standard deviation of this relative approximation error over all locations with a non-empty influence are given. The experiments are performed for M_{DA}, M_{FA},

		`FourSquare`			`Gowalla`		
		mean $\pm\sigma$			mean $\pm\sigma$		
		b=64	b=128	b=256	b=64	b=128	b=256
Rel. error	M_{DA}	0.03 ± 0.15	0.01 ± 0.01	0.01 ± 0.06	0.03 ± 0.17	0.01 ± 0.12	0.01 ± 0.01
	M_{FA}	0.26 ± 0.45	0.19 ± 0.34	0.15 ± 0.35	Out of Memory		
	M_{DR}	0.05 ± 0.21	0.05 ± 0.21	0.05 ± 0.2	0.17 ± 0.41	0.17 ± 0.41	0.17 ± 0.4
	M_{FR}	0.05 ± 0.21	0.05 ± 0.21	0.05 ± 0.21	Out of Memory		

Table B.7: Accuracy (relative error) for approximate algorithms w.r.t bucket size.

		FourSquare			Gowalla		
		b=64	b=128	b=256	b=64	b=128	b=256
Time	M_{DA}	378	428	544	2,031	2,605	2,499
	M_{FA}	928	955	993	9,479	9,685	9,912
Memory	M_{DA}	14,275	18,636	27,372	89,418	116,847	171,908
	M_{FA}	14,569	18,943	27,726	90,793	118,173	172,985

Table B.8: Time (sec) and memory (MB) required by the approximate algorithms w.r.t, bucket size.

M_{DR}, and M_{FR}. We ran the experiments for different numbers of buckets (b) for the *HLL* sketch, being, $64, 128$ and 256. It can be seen in the table that the errors are unbiased (0 on average) and the standard deviation decreases as the number of buckets increases. The error is a bit higher for M_R as compared to M_A because in the relative model the influence is computed by taking the ratio of two approximated set cardinalities. Values for b beyond 256 yielded only modest further improvements and hence we used $b = 256$ in all further experiments. The results for M_{FA} and M_{FR} for the Gowalla could not be computed due to the huge memory requirement.

Time and space consumption. Next, we analyzed the computation time and memory requirements for the approximate approach by computing the influence oracles for M_D and M_F. The results are shown in Table B.8. It can be observed that time and memory increase with increasing number of buckets b. Furthermore, it can also be observed that the approximate approach outperformed the exact approaches in computation time and memory given in Table B.6. The improvement for M_F in the computation time is two folds while using only 18% of memory. Due to the sparsity of data, however, the gain for M_D is less, i.e., 63% of the time and 48% of the memory is required by the approximate approach as compared to exact one. This is because the sizes of the sets of bridging visitors are very modest. The results for Gowalla for M_F could not be computed by the exact algorithm due to insufficient memory. However, for all the bucket sizes the approximate algorithm computes it by taking less than half of the available memory.

7.4 Effects of parameters: ω and τ

Computation time. We studied the runtime of the algorithms on all the datasets for different values of $\omega := 8, 20$ and 50. Considering the significant improvement of the approximate algorithms in computational resources without compromising on accuracy, for all further experiments, we use approximate variants for M_{DA} and M_{FA}. However, the exact algorithm for M_{IA} is used as we do not have its approximate variant. The average runtime for processing all the activities (T_p) under the models varies only depending on

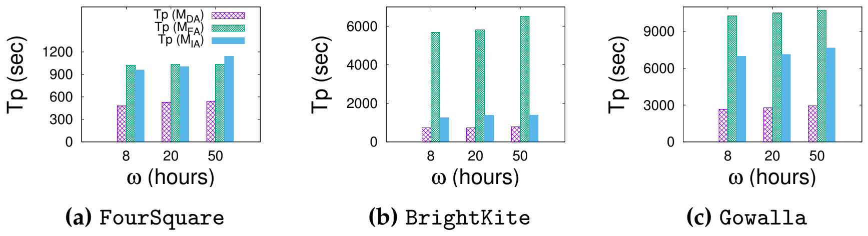


Fig. B.10: Total Time (Tp) in seconds w.r.t. ω to process all activities for $\tau = 2$ under M_{DA}, M_{FA} and M_{IA} models.

the influence models, i.e, whether or not we consider friends; it does not depend on τ. Also, the oracle query time (T_q) is independent of τ and the influence model. The run times for processing all the activities are shown in Figure B.10 for the three datasets FourSquare, BrightKite and Gowalla. The running time increases with increasing influence window size ω as more locations from the visit history remain active. Running time is higher in the M_F which is not surprising either as the number of users to include in the bridging visitors sets increases due to the addition of friends. For the M_I, we needed to find influenced users of bridging visitors thus computation time is higher as compared to M_D. The time taken to process Gowalla dataset is the highest as it has the largest number of locations.

Memory consumption. We also studied the memory required by the approximation algorithm on all the datasets for different values of $\omega := 8, 20$ and 50. Unlike for the processing time, the average memory required to process all the activities under M_D and M_F does not vary based on whether we consider friends or not. This is because the HLL sketch storing the bridging visitor set size remains constant in size even if a larger number of users is added to it. The memory requirement increases slightly with ω as more locations are getting influenced due to a larger influence window. The results are shown in Figure B.11. The total memory requirements increased linearly with time as new locations came in over time for which a complete influence

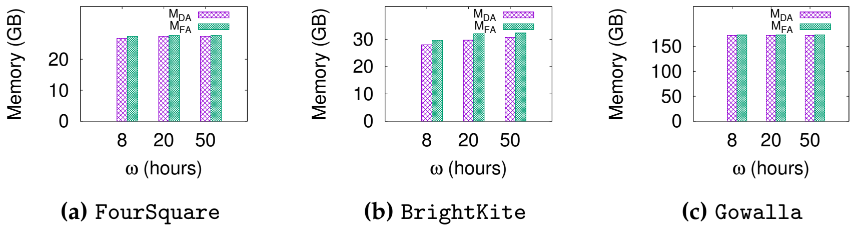


Fig. B.11: Total Memory in GB w.r.t. ω to process all activities for $\tau = 2$ under M_{DA} and M_{FA}.

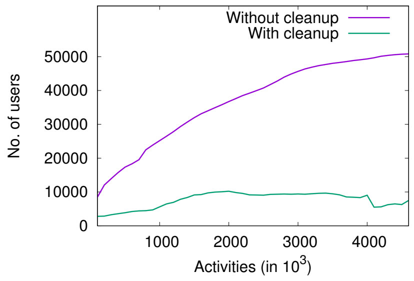
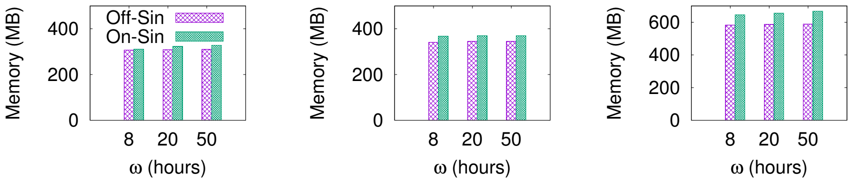


Fig. B.12: User visit history set size growth w.r.t to number of activities processed with and without cleanup process.

summary was needed to be maintained. We further pruned the outdated locations in the visit histories thus, over time the size of user history remained constant as shown in Figure B.12.

7.5 Single-Influencer based Location Influence

Next, we present the resource requirements of the algorithms for constructing single-influencer based influence oracles. The memory consumption by the *On-Sin* and the *Off-Sin* algorithms is shown in Figure B.13. Since the memory requirement for the algorithms is $log(\omega)$, thus a very modest change in the memory with respect to different values of ω was observed. The *Off-Sin* requires less memory as compared to the *On-Sin*. The reason is that for the *Off-Sin*, we consider the activities in reverse order of time, so we approximate the user history using the probabilistic data structure. However, for *On-Sin*, we maintain the exact user history which requires more memory. The difference of time and memory requirements among *On-Sin* and *Off-Sin* is modest because of the data sparsity and small window size. However, these algorithms for the special case outperformed the approximate algorithm for all influence models (shown in Figure B.11) up to 22x in memory consumption.



(a) FourSquare **(b)** BrightKite **(c)** Gowalla

Fig. B.13: Comparison of memory consumption for *Off-Sin* and *On-Sin*

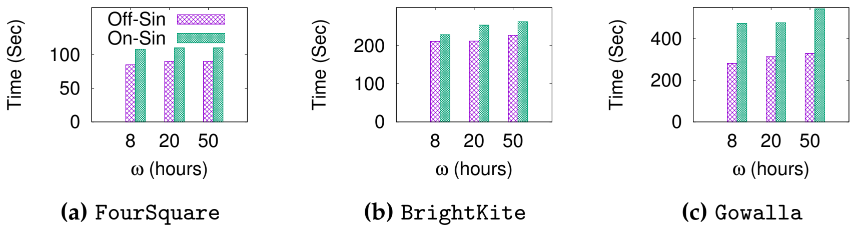


Fig. B.14: Comparison of computation time for *Off-Sin* and *On-Sin*

The reason is for the special case we do not maintain the bridging visitor set because the influence is spread by a single carrier, and thus the memory is saved.

The computation time by the *On-Sin* and *Off-Sin* are shown in Figure B.14. Here, it can be observed that for the *Off-Sin* algorithm, like for memory consumption, there was a modest improvement in computation time as compared to the *On-Sin*. Similarly, both of these algorithms required up to 20x less computation time as compared to the approximate algorithm for all influence models (shown in Figure B.10).

7.6 Influence Maximization

Influence of α. Our next goal is to study how the influence maximization algorithm performs for different values of α. In order to avoid data sparsity issues, we filtered out those locations which have only one visitor from all the datasets. We tested the spread of top 200 locations obtained by considering values of α from 0.01 to 0.99. We observed that the number of bridging visitors per location is highly skewed as can be learned from Figure B.6a. Due to this, the potential influenced locations having few bridging visitors are less likely to affect the influenced set of the locations. The effect of varying alpha

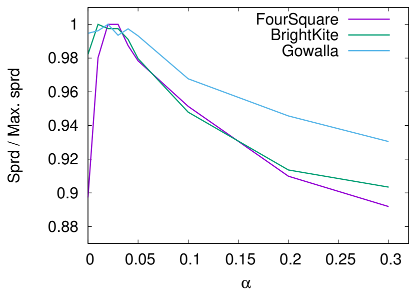


Fig. B.15: Plot of ratio of total influence spread w.r.t. alpha (200 seeds).

τ	Time (sec)								
	FourSquare			BrightKite			Gowalla		
	$k = 10$	$k = 20$	$k = 50$	$k = 10$	$k = 20$	$k = 50$	$k = 10$	$k = 20$	$k = 50$
$\tau_{DA} = 2$	37	40	120	24	50	218	132	213	723
$\tau_{DR} = 0.6$	108	101	180	84	132	545	521	525	778
$\tau_{IA} = 5$	10	18	60	15	32	208	70	173	706
$\tau_{IR} = 0.6$	50	45	99	37	73	177	292	346	563
$\tau_{FA} = 120$	19	61	525	35	553	7776	181	1168	19302
$\tau_{FR} = 0.6$	68	96	591	201	151	591	450	557	1386

Table B.9: Time taken to find top k locations.

on the influence spread is shown in Figure B.15. As expected for these sparse datasets, our algorithms performed best with a lower value of α. We use $\alpha = 0.03$ for our experiments.

Computation time. We study the computation time for finding top-k influential locations under all influence models. The runtime is close in the both M_A and M_R. The time increases with k. Nevertheless, the increase is modest; for instance, finding the top-50 locations takes less than 2 minutes for `FourSquare`. We report the results in Table B.9.

8 Conclusion and Future Work

In this paper, we introduced a location influence maximization approach that can be used to optimize outdoor marketing strategies such as finding optimal locations for advertising products to maximize the geographical spread. In order to do that, we captured the interactions of locations on the basis of their visitors to compute the influence of locations among each other. We provided two models namely the absolute influence model and the relative influence model. We further provide three variants of these models that incorporate the social graph and consider the friends of users that have potential to repeat their activities in future, and improve the location influence up to 45%. We proposed a data structure: influence oracle to efficiently compute the influence of locations on the basis of these models for finding top-k influential locations. In order to maintain this data structure, we first provided a set-based exact algorithm. Then, we optimized the time and memory requirements of the algorithms by utilizing a probabilistic data structure. We further introduced a method in which single carriers can be used to spread the influence. For this case, we provide two algorithms, an off-line and an on-line. With the help of these algorithms, we further improved the computation time and memory requirement by 20x and 22x, respectively. Finally, we provided a greedy algorithm to compute the top-k influential locations. In order to evaluate the methods, we utilized three real datasets. We first analyzed the LBSN datasets: `FourSquare`, `BrightKite` and `Gowalla` to verify some claims and to provide optimal values for thresholds of the influence models. Then, we evaluated our approaches for the computation of the Oracle and finding top-k locations in terms of accuracy, computation time, memory requirement and scalability. We further show the effectiveness of our proposed models by comparing the influence spread of top-k locations fetched by our approach with that of two variants of a state-of-the-art approach; IRS and IRS-window.

In the future, we plan to enrich location influence models by incorporating the activities users perform with their friends in groups. Moreover, we aim to provide distributed techniques for computing the Oracle data structures and influences for the models.

References

[1] A. AlDwyish, E. Tanin, and S. Karunasekera, “Location-based social networking for obtaining personalised driving advice,” in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS ’15, 2015, pp. 91:1–91:4.

[2] L. Ferrari, A. Rosi, M. Mamei, and F. Zambonelli, “Extracting urban patterns from location-based social networks,” in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, ser. LBSN ’11, 2011, pp. 9–16.

[3] F. J. Mata and A. Quesada, “Web 2.0, social networks and e-commerce as marketing tools,” *J. Theor. Appl. Electron. Commer. Res.*, vol. 9, no. 1, pp. 56–69, 2014.

[4] M. A. Saleem, X. Xie, and T. B. Pedersen, “Scalable processing of location-based social networking queries,” in *MDM*, 2016, pp. 132–141.

[5] D. Kempe, J. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network,” in *KDD*, 2003, pp. 137–146.

[6] W. Chen, Y. Wang, and S. Yang, “Efficient influence maximization in social networks,” in *KDD*, 2009, pp. 199–208.

[7] H.-H. Wu and M.-Y. Yeh, “Influential nodes in a one-wave diffusion model for location-based social networks,” in *PAKDD*, 2013, pp. 61–72.

[8] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei, “Evaluating geo-social influence in location-based social networks,” in *CIKM*, 2012, pp. 1442–1451.

[9] W.-Y. Zhu, W.-C. Peng, L.-J. Chen, K. Zheng, and X. Zhou, “Modeling user mobility for location promotion in location-based social networks,” in *KDD*, 2015, pp. 1573–1582.

[10] M. A. Saleem, R. Kumar, T. Calders, T. B. Pedersen, and X. Xie, "Location influence in location-based social networks," in *WSDM*, 2017, pp. 621–630.

[11] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," *DMTCS Proceedings*, 2008.

[12] P. Domingos and M. Richardson, "Mining the network value of customers," in *KDD*, 2001, pp. 57–66.

[13] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *KDD*, 2002, pp. 61–70.

[14] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck, "Sketch-based influence maximization and computation: Scaling up with guarantees," in *CIKM*, 2014, pp. 629–638.

[15] M. Gomez-Rodriguez and B. Schölkopf, "Influence maximization in continuous time diffusion networks," in *ICML*, 2012, pp. 313–320.

[16] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha, "Scalable influence estimation in continuous-time diffusion networks," in *NIPS*, 2013, pp. 3147–3155.

[17] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "A data-based approach to social influence maximization," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 73–84, 2011.

[18] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Discovering leaders from community actions," in *CIKM*, 2008, pp. 499–508.

[19] ——, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.

[20] G. Li, S. Chen, J. Feng, K.-l. Tan, and W.-s. Li, "Efficient location-aware influence maximization," in *SIGMOD*, 2014, pp. 87–98.

[21] P. Bouros, D. Sacharidis, and N. Bikakis, "Regionally influential users in location-aware social networks," in *SIGSPATIAL*, 2014, pp. 501–504.

[22] Y.-T. Wen, P.-R. Lei, W.-C. Peng, and X.-F. Zhou, "Exploring social influence on location-based social networks," in *ICDM*, 2014, pp. 1043–1048.

[23] T. Zhou, J. Cao, B. Liu, S. Xu, Z. Zhu, and J. Luo, "Location-based influence maximization in social networks," in *CIKM*, 2015, pp. 1211–1220.

[24] N. T. Hai, "A novel approach for location promotion on location-based social networks," in *RIVF*, 2015, pp. 53–58.

[25] X. Wang, Y. Zhang, W. Zhang, and X. Lin, "Distance-aware influence maximization in geo-social network," in *ICDE*, 2016, pp. 1–12.

[26] T.-N. Doan, F. C. T. Chua, and E.-P. Lim, "Mining business competitiveness from user visitation data," in *SBP*, 2015, pp. 283–289.

[27] R. Kumar and T. Calders, "Information propagation in interaction networks," in *EDBT*, 2017, pp. 270–281.

[28] L. LováSz, "Review of the book by alexander schrijver: Combinatorial optimization: Polyhedra and efficiency," in *Oper. Res. Lett.*, 2005.

[29] H. Gao, J. Tang, and H. Liu, "Exploring social-historical ties on location-based social networks," in *AAAI*, 2012.

[30] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *KDD*, 2011, pp. 1082–1090.

[31] Q. Liu, M. Deng, Y. Shi, and J. Wang, "A density-based spatial clustering algorithm considering both spatial proximity and attribute similarity," *Computers and Geosciences*, vol. 46, pp. 296–309, 2012.

[32] R. R. Braam, H. F. Moed, and A. F. Van Raan, "Mapping of science: Critical elaboration and new approaches, a case study in agricultural biochemistry," in *Informetrics*, 1988, pp. 15–28.

[33] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.

Paper C

IMaxer: A Unified System for Evaluating Influence Maximization in Location-based Social Networks

Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Xike Xie, Torben Bach Pedersen

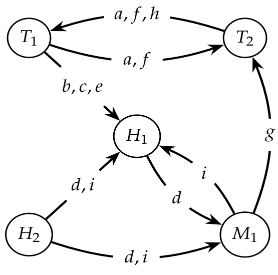
The paper has been published in the
26th ACM International Conference on Information and Knowledge Management, (CIKM) 2017, pp.2523–2526, 2017.

Abstract

Due to the popularity of social networks with geo-tagged activities, so-called location-based social networks (LBSN), a number of methods have been proposed for influence maximization for applications such as word-of-mouth marketing (WOMM), and out-of-home marketing (OOH). It is thus important to analyze and compare these different approaches. In this demonstration, we present a unified system IMaxer that both provides a complete pipeline of state-of-the-art and novel models and algorithms for influence maximization (IM) as well as allows to evaluate and compare IM techniques for a particular scenario. IMaxer allows to select and transform the required data from raw LBSN datasets. It further provides a unified model that utilizes interactions of nodes in an LBSN, i.e., users and locations, for capturing diverse types of information propagations. On the basis of these interactions, influential nodes can be found and their potential influence can be simulated and visualized using Google Maps and graph visualization APIs. Thus, IMaxer allows users to compare and pick the most suitable IM method in terms of effectiveness and cost.

Check-ins

loc	Users		
	t=1	t=2	t=3
T_1	b,c,e,f	a,h	f
T_2	a,h	f,g	a
M_1	g	i	d
H_1	–	b,c,d,e	i
H_2	d,i	–	–



Friendships

User	Friends
a	c,h,i
b	d,f,i
c	a,f
d	b,e
e	d,g
f	c,b
g	e
h	a
i	a,b

Fig. C.1: Toy example: the tables "Check-ins" and "Friendships" shows the activities of users, and their social friends in an LBSN, respectively. The graph in the center, extracted from the LBSN shows the influence among locations based on visitors that are spread from one location to another [1].

1 Introduction

In this demonstration, we present *IMaxer*, a unified system that supports a number of different models and algorithms for analyzing and comparing information propagation and influence maximization techniques in LBSNs. In an LBSN, the nodes, i.e., users and locations, form two types of edges: 1) friendships that represent relationships; 2) activities that represent check-ins of users at locations. The propagation of information in LBSNs is studied based on friendships and activities. Let A and B be two nodes representing users who are friends. We say that A *influences* B, or A spreads information to B, if B follows an activity of A. If there are many such following actions by B, A is considered to have a strong influence on B. The process of finding the most influential nodes is called *Influence Maximization*. Applications like viral marketing utilize these influential nodes to maximize the information spread for advertising purposes.

A number of techniques has been proposed in this area. For instance, in [2, 3], authors propose methods for maximizing the number of influenced users. In [1, 4, 5] methods for maximizing the number of influenced locations for geographical spread of a message and location promotion are proposed. Moreover, several information propagation models have also been proposed, such as the *Linear Threshold*, the *Information cascade models* [6], the *Time Constraint* [2] and the *Absolute and Relative Influence based information cascade models* [1]. On the other hand, industrial applications [7, 8], focus on management of social media accounts to maximize the spread such as determining the best moment to upload content to maximize viewers. However, to the best of our knowledge, there does not exist any system which provides a complete pipeline of algorithms and models required for all steps of influence maximization and which supports a range of influence maximization techniques for diverse contexts and allows comparison among them. Such a system is useful for selecting an appropriate information propagation mechanism in terms of effectiveness and cost. In this demo, we present a system to handle all such problems in influence maximization.

Example: Consider a marketer interested in spreading her promotional message to the most regions in New York City (NYC). The information can be propagated using locations or users to achieve this goal as given in the following two use cases. Consider the example LBSN in Figure C.1 where for each user, her friends and check-ins in LBSN are given. The graph represents the influence of locations among each other where nodes, labeled with capital letters represent locations and lowercase letters on edges represent visitors as carriers of influence.

Use Case 1: Find locations where she should display the advertisements such that people who see them visit the maximum number of other locations in NYC and spread the message indirectly by talking to their friends and relatives.

Use Case 2: Find users who visit together the most regions of NYC. Give them promotional gifts such as a t-shirt so that they indirectly promote the message to the people of regions they visit.

For aforementioned use cases, a system is required that can be used to select activities in NYC, find influential locations and users for both use cases and simulate their potential influence. The system should allow to analyze and compare different methods and find a suitable way to propagate information. The architecture of IMaxer achieves this by providing the following four components.

- *Data selection:* selects a desired slice from a dataset in order to target a particular audience.
- *Data preprocessing and transformation:* pre-processes the data to improve the data quality for better accuracy.
- *Influential nodes mining:* captures information propagation mechanisms of diverse contexts for applications such as WOMM or OOH, and find the corresponding influential nodes.
- *Influence spread simulation:* simulates, visualizes and analyzes the influence of influential nodes. This component can be used to compare information propagation methods and select the most suitable one for a particular use-case.

In short, IMaxer makes following contributions:

- a novel system that provides a complete pipeline of models and algorithms for evaluating and comparing influence maximization mechanisms
- a unified model for capturing diverse information propagation and influence maximization algorithms
- an extensible plugin architecture that allows to integrate new algorithms.

2 IMaxer System Overview

In this section, we map the modules of IMaxer to traditional data mining steps and explain their working:

Data Selection: Usually, marketing approaches target the audience with specific attributes. The data selection module allows users to upload the LBSN dataset and select the desired slice of data based on several given criteria, such as constraints on attributes of users and locations, time, region, combination of friendships and activities. The dataset is filtered on the basis of the given constraints and persisted on disk for subsequent processing.

Example: For both the use cases, in this module, the dataset is filtered to fetch the activities in the region of interest, i.e., NYC.

Data Preprocessing and Transformation: Due to privacy concerns, inaccuracy of GPS devices, and different business models of service providers, there exist anomalies in LBSN datasets, e.g., multiple location Ids are assigned to the same GPS coordinate. Further, in LBSNs different applications may require different location granularities e.g., from a location to a region. Thus, in order to solve data anomaly issues as well as to provide different granularities of locations, we cluster the locations such that each cluster represents a POI. IMaxer provides two algorithms for clustering locations: *grid-based clustering*, and *density-based spatial clustering*.

Example: In this module, for both use cases, we transform the location Ids in NYC such that each location represents a POI.

Influential Nodes Mining using Unified IM Model: IMaxer provides a unified model composed of three layers as shown in Figure C.2 to capture diverse information propagation scenarios. First, the node dimension is responsible for identifying the influential and influenced nodes called source node and receiver node, respectively, which can either be users or locations. Next, the characteristics dimension filters the source and receiver nodes on the required attributes such as age, and gender of users. Moreover, it defines the cardinality for source and receiver nodes, i.e., the number of nodes that should be selected as source to spread the information and as receiver that should be influenced, respectively. Finally, the interaction network layer on the basis

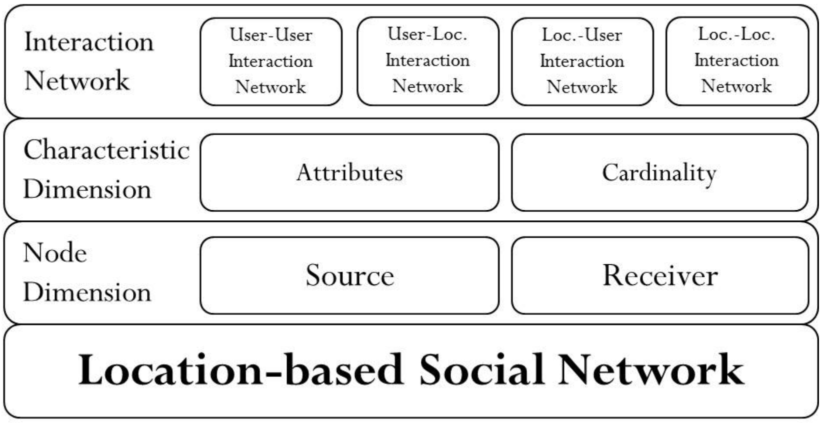

Interaction Network
User-User Interaction Network
User-Loc. Interaction Network
Loc.-User Interaction Network
Loc.-Loc. Interaction Network
Characteristic Dimension
Attributes
Cardinality
Node Dimension
Source
Receiver
Location-based Social Network

Fig. C.2: The Unified Influence Maximization Model

of possible interactions among nodes of LBSNs provides four types of interaction networks, i.e., user-user, user-location, location-user, location-location interaction networks. Based on the selection of types of source and receiver nodes the corresponding type of influential activity is fetched as shown in Table C.1 and the corresponding interactions network is constructed. Influence scores of source nodes using IM algorithms are computed on the basis of these interactions and finally, the nodes capable of influencing maximum nodes are selected as top-k influential nodes.

Example: In use case 1, we are interested in maximizing geographical spread using locations, the influential activity is spreading of visitors from a location to another location as shown in Figure C.1. Thus location is considered as influential as well as influenced node, and interaction network is location-location interaction network. On the basis of it top-2 influential locations in the figure are T_1 and H_2 as they spread their visitors to the maximum number of other locations. For the use case 2, maximizing geographical spread using users is intended. Thus, visit of a location by a user is considered an influential activity. In this case, user and location are considered influential and influenced nodes, respectively, and the interaction network is user-location interaction network. Here, the top-2 influential users are d and a as they together visit most of the locations.

Influence Spread Simulation: In this module, IMaxer takes the influential nodes as input and outputs the potentially influenced nodes. The influence of top-k nodes are simulated and propagated through the interaction network and all the nodes that are influenced are fetched as potentially influenced nodes. In order to do that IMaxer provides a number of spread simulation algorithms (discussed in Section 3). The potentially influenced nodes are further displayed using Google Maps and Gephi to visualize the spread of the influential nodes, for location and user nodes respectively. Since this module allows users to provide influential seeds directly as an input to find their influenced nodes, it can be used to compare and analyze the results of multiple influence maximization methods and choose the best one for a particular scenario.

Example: In use case 1, in the Figure C.1, T_1 and H_2 are influencing all other locations. Because, users a, f visits T_2 and b, c, e visits H_1 after visiting T_1, and users d, i visits H_1 and M_1 after visiting H_2. Similarly, for use case 2, the potentially influenced locations of the top-2 influential users d and a are all the locations. Because, d visits M_1, H_1, and H_2, and a visits T_1 and T_2. In this case, as influence of both approaches is same, decision can be made based on cost of considering a user versus location for spreading message. On the other hand, if we consider $k = 1$, then the the top influential user d propagate message to one more location that the top influential location T_1, thus, in this case it can be chosen for more influence.

3 IMaxer's technical overview

In this section, we first model four different types of information propagation and their corresponding IM mechanisms using the unified IM Model. Then based on this model, we provide abstractions of algorithms for IM and spread simulation. Finally, we present the implementation details of the system and methods to incorporate other existing as well as new algorithms.

3.1 Modeling IM using *Unfied IM Model*

Given a set of users U and a set of locations L, an *activity* is a visit/check-in of a user at a location. It is a triplet (u, l, t), where $u \in U$ is a user, $l \in L$ a location and t is time of the visit of u at l. The set of all activities over U and L is denoted by $\mathcal{A}(U, L)$. The *LBSN* over U and L consists of a graph $G_S(U, F)$, called *social graph*, where $F \subseteq \{\{u, v\} | u, v \in U\}$ represents friendships between users, and a set of activities $A \subseteq \mathcal{A}(U, L)$.

Given $u, u' \in U \cup L$, we measure $u's$ influence on u' by two factors. First, the number of influential activities $\mathcal{I}(u \rightarrow u')$ that implies an impact/influence of u on u'. Second, a time window during which the activities are performed called influence window ω. The influence among two vertices is 1 if the number of influential activities is greater than a threshold τ, otherwise 0. The values of τ and ω may vary on the basis of types of influential activities and users' requirements. However, for simplicity and to capture all activities we take $\tau = 0$ and $\omega = 3$ for all the examples given below. The unified IM Model utilizes the interactions of nodes in LBSNs to model four different types of information propagation mechanisms which are further

Source	Dest.	Influential Activity ($\mathcal{I}$)	Application (Maximizes)
User	User	u follows u'	Followers
Location	Location	l spreads visitors to l'	Geographical spread
User	Location	u visits l'	Unique visited locations
Location	User	l visited by u'	Unique visitors

Table C.1: Information propagation mechanisms w.r.t. types of the influential and influenced nodes in LBSNs. Here, u and l are the influential user and location, respectively and u' and l' are the influenced user and location, respectively.

utilized to formulate influential activities. These information propagation mechanisms with corresponding influential activities and potential applications are shown in Table C.1. Next, we define influences on the basis of these interactions and influential activities.

Influence among users: The influence of a user $u \in U$ on a user $u' \in U$ is determined by the number of activities performed by u that are followed by u' within the time window ω, such that u and u' are friends. The influence of u on u' is considered 1 if the number of such followed activities by u' is greater than the threshold τ.

Example: Most influential users in this context can be utilized to maximize the number of followers. Such users are exploited in applications like viral marketing. In Figure C.1, top-2 such influential users are b and c which influence f and i, and a and f, respectively.

Influence among locations: The influence of a location $l \in L$ on a location $l' \in L$ is determined by the number of users that after visiting l visits l', within time window ω. The influence of l on l' is 1, if such visitors are greater than τ.

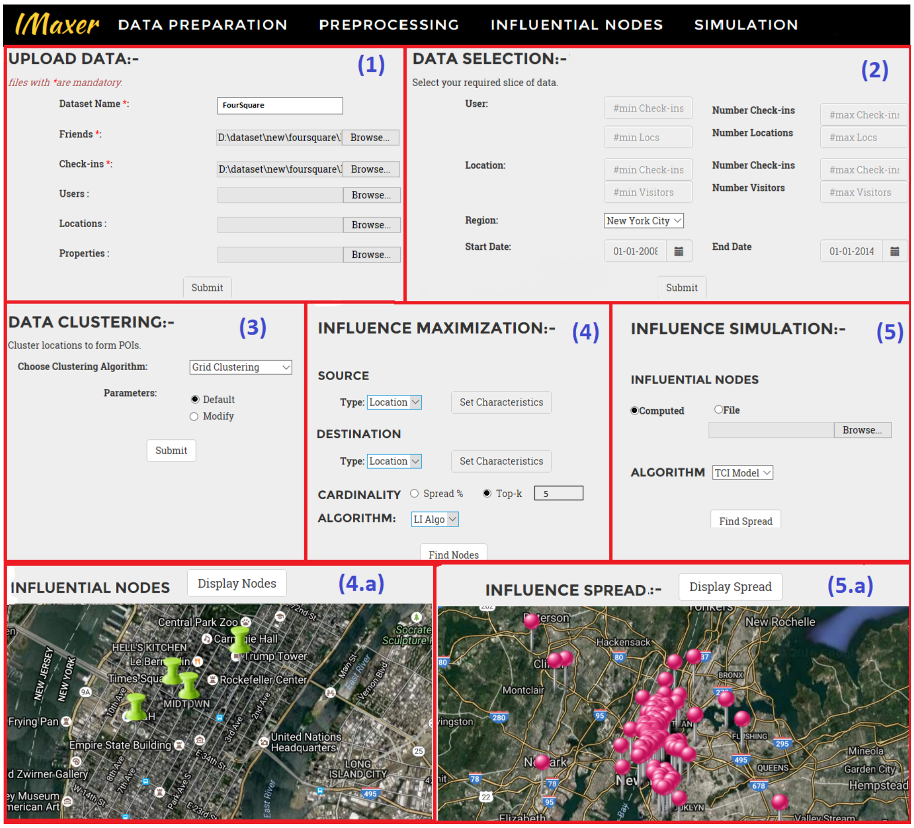
Example: Use case 1 is the example of this type of influence. In Figure C.1, the top-2 influential locations are T_1 and H_2 and the influenced locations are T_1, T_2, M_1, H_1 and H_2.

Influence of user on location: The influence of a user u on a location l' is determined by the number of visits of u at l' within ω. If such visits are greater than τ then the influence of u on l' is 1.

Example: Use case 2 lies in this category of the influence. In Figure C.1, the most influential users for this type of influence are d and a and their influenced locations are T_1, T_2, M_1, H_1, and H_2.

Influence of location on user: The influence of a location l on a user u' is determined by the number of times l is visited by u' within ω. The influence of l is considered 1 if the number of such visits of u' at l is greater than τ.

Example: Such type of influence can be used to find the locations on which advertisements can be displayed such that the message is viewed by the maximum number of users. In Figure C.1, the top-2 most influential locations for this type of influence are H_1 and T_2, which collectively attracts all the users.



UPLOAD DATA:- (1)

files with *are mandatory.

Dataset Name *:	FourSquare
Friends *:	D:\dataset\new\foursquare\ Browse...
Check-ins *:	D:\dataset\new\foursquare\ Browse...
Users :	Browse...
Locations :	Browse...
Properties :	Browse...

Submit

DATA SELECTION:- (2)

Select your required slice of data.

User:	#min Check-ins	Number Check-ins	#max Check-ins
	#min Locs	Number Locations	#max Locs
Location:	#min Check-ins	Number Check-ins	#max Check-ins
	#min Visitors	Number Visitors	#max Visitors
Region:	New York City		
Start Date:	01-01-2008	End Date	01-01-2014

Submit

DATA CLUSTERING:- (3)

Cluster locations to form POIs.

Choose Clustering Algorithm: Grid Clustering

Parameters: [✓ Default] Modify

Submit

INFLUENCE MAXIMIZATION:- (4)

SOURCE

Type: Location Set Characteristics

DESTINATION

Type: Location Set Characteristics

CARDINALITY Spread % [✓ Top-k] 5

ALGORITHM: LI Algo

Find Nodes

INFLUENCE SIMULATION:- (5)

INFLUENTIAL NODES

[✓ Computed] File Browse...

ALGORITHM TCI Model

Find Spread

INFLUENTIAL NODES Display Nodes (4.a)

INFLUENCE SPREAD:- Display Spread (5.a)

Fig. C.3: Compact snapshots of usage of IMaxer for maximizing geographical spread with locations in NYC

Influence Maximization: Next, we utilize these interactions and influential activities to create interaction networks. An interaction networks is a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \subseteq U \cup L$ is the set of vertices and $\mathcal{E}$ is the set of edges: $\{(u, u') \mid \mathcal{I}(u \to u') > \tau\}$. The set of vertices that are influenced by a vertex u is called influence set $\sigma(u)$. Once the interaction network $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is obtained for a use case we can use the corresponding IM algorithms to determine the influence set of individual vertices. IMaxer provides 2 IM algorithms taken from [2, 3] for capturing user-user influence, from [1] for location-location and two novel IM algorithms based on time constraint window and edge degree of interaction network for user-location and location-user influence. Once the influence sets of individual vertices are known we apply a greedy method based on influence scores of vertices to find top-k vertices.

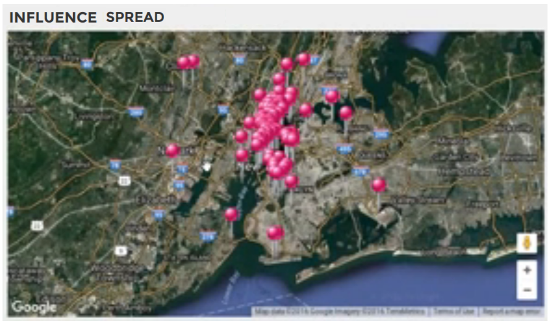


Fig. C.4: Potential geographical spread with users in NYC

Spread Simulation: The next problem IMaxer addresses is simulation of the influence spread of the selected top-k vertices after influence maximization. In order to do that, IMaxer currently, provides the implementation of Linear Threshold model, Independent Cascade model [6], Time Constrained Information Cascade Model [2], and, Absolute and Relative Influence based information cascade models [1].

3.2 System Implementation

The IMaxer's system implementation is modular as it is based on the factory design pattern. Currently, IMaxer provides three datasets taken from real world LBSNs: Foursquare, BrightKite, and Gowalla. However, it allows to add new data sources. Moreover, IMaxer allows its users to add other influence maximization algorithms by implementing an interface: *BaseAlgo.java*. Similarly, new algorithms for spread simulation can be added by implementing the interface *BaseSimulation.java*. The resulting visualization is not configurable and is fixed based on the receiver node dimension. If the receiver is location, Google Maps are used to visualize the spread and for the cases when the receiver is a user, graph visualization using Gephi is provided.

IMaxer is built using Java for business logic with HTML and Javascript for the front end. We use the Google Maps API [1] to present the location spread and Gephi [2] to present user spread. The configuration details are stored as XML files making it easier to configure new data sources and attributes.

4 Demonstration

We demonstrate the two use cases given in Section 1 to show and compare the propagation of information using locations and users, respectively, with a real

[1]https://developers.google.com/maps/
[2]https://gephi.org/

dataset from Foursquare LBSN. The information is spread to 236 locations in use case 1 as compared to 192 locations in use case 2. Due to space limit, we combined the snapshots of usage of different modules of IMaxer for use case 1, as shown in Figure C.3. Further, the potential influence of users in case 2 is shown in Figure C.4. Moreover, we explain the working of IMaxer for both use cases in the video [9].

Demonstration: First, an IMaxer's user uploads the mandatory files containing friendship and check-in information of the data set (Foursquare) as shown in step 1 in Figure C.3. Then she filters the data on the basis of user requirements as shown in step 2. By default, she considers all users and locations. However, only activities within NYC and given dates are fetched. In step 3, she transforms the location using grid clustering and maps the activities correspondingly. In step 4, first types of source and destination nodes, and their cardinalities are asked by the user. Then, the influence maximization algorithm is selected, i.e., LI. The nodes marked in green in step 4.*a* were obtained by running the LI algorithm for maximizing geographical spread [1] to find the top-5 most influential locations. The usage of spread simulation is shown in step 5. The locations marked in red in step 5.*a* indicate the locations in NYC that will potentially get influenced according to the selected spread simulation algorithm TCIC. Moreover, IMaxer also provides details of the influenced locations with their corresponding influential locations in a summary table.

References

[1] M. A. Saleem, R. Kumar, T. Calders, T. B. Pedersen, and X. Xie, "Location influence in location-based social networks," in *WSDM*, 2017, pp. 621–630.

[2] R. Kumar and T. Calders, "Information propagation in interaction networks," in *EDBT*, 2017, pp. 270–281.

[3] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.

[4] W.-Y. Zhu, W.-C. Peng, L.-J. Chen, K. Zheng, and X. Zhou, "Modeling user mobility for location promotion in location-based social networks," in *KDD*, 2015, pp. 1573–1582.

[5] T. Zhou, J. Cao, B. Liu, S. Xu, Z. Zhu, and J. Luo, "Location-based influence maximization in social networks," in *CIKM*, 2015, pp. 1211–1220.

[6] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003, pp. 137–146.

[7] “sproutsocial,” http://sproutsocial.com/.

[8] “Hootsuite,” https://hootsuite.com.

[9] “Imaxer’s demonstration,” https://youtu.be/hkj6aAfhYkk.

Paper D

Predicting Visitors Using Location-Based Social Networks

Muhammad Aamir Saleem, Felipe Costa, Panagiotis Karras, Toon Calders, Torben Bach Pedersen

The paper is under revision
for a Conference publication

Abstract

Location-based social networks (LBSN) are social networks complemented with users' location data, such as geo-tagged activity data. Predicting such activities finds application in marketing, recommendation systems, and logistics management. In this paper, we exploit LBSN data to predict future visitors at given locations. We fetch the travel history of visitors by their check-ins in LBSNs and identify five features that significantly drive the mobility of a visitor towards a location: (i) historic visits, (ii) location category, (iii) time, (iv) distance and (v) friends' activities. We provide a visitor prediction model, CMViP*, based on collective matrix factorization and influence propagation.* CMViP *first utilizes collective matrix factorization to map the first four features to a common latent space to find visitors having a significant potential to visit a given location. Then, it utilizes an influence-mining approach to further incorporate friends of those visitors, who are influenced by the visitors' activities and likely to follow them. We provide a detailed analysis on the contribution of each considered feature in the predictive task. Our experiments on two real-world datasets show that our methods outperform the state of art in terms of precision and accuracy.*

The layout has been revised.

1 Introduction

Social network analysis allows the provision of diverse recommendations to users, e.g., friends and activities. At the same time, the pervasiveness of location-aware devices allows users of online social networks to share geo-tagged contents such as their current location. Such online social networks that exploit location information are called location-based social networks (LBSNs). The geo-tagging of activities in LBSNs provides an opportunity to capture and utilize the mobility behavior of users such as to provide *location-based recommendations*. For instance, one can analyze the historical activities of users and recommend locations to them for their potential visit; in such cases, the focus is on providing recommendations to the users. An overlooked perspective to the recommendation is the *prediction* of users that will visit a given location; such a perspective finds application in event planning, traffic management, mobile phone capacity planning, etc. For instance, consider the following example:

> **Example 1.1**
> A cinema owner wants to know the persons who would choose to watch the movie *Pirates of Caribbean* at a cinema at a certain time.

To answer such queries, we need to predict that whether a user will visit a location of a particular category, e.g., cinema, in a particular region and at a particular time. An illustration of this example is shown in Figure D.1. Here, we show the users' check-ins in the form of triplets: user, location and time. We also show the characteristics of locations, i.e., location id, category and GPS coordinates and a social graph depicting friends of users in LBSNs. Here, in order to predict the visitors at l_1 at t_1 (hour of the day): 13:00, we compute the potential of users based on their check-ins at l_1, at locations of category *Sports* ($l_1's$ *category*) and at time t_1. Based on these values, u_1 and u_2 are considered as potential visitors of l_1 at t_1 as they have the highest number of check-ins under these conditions. In this paper, we address such a problem: *Given a location, its category, and a time period, predict the visitors to this location within the given time period using LBSN data on past mobility.*

To address this problem, we need to identify which features in past mobility data affects users' mobility. To that end, we analyze the available LBSN data and identify the following mobility-affecting features: (i) historic visits, (ii) location category, (iii) time, (iv) distance, and (v) friends' activities. We show the effect of each of these features on user mobility in Section 3.2.

We provide a novel model CMViP for predicting visitors that combines collective non-negative matrix factorization approach with an influence diffusion model. Initially, we compute a set of frequency matrices, capturing the

User	Location	Time
u_1	l_1	13:25 11/12/2017
u_1	l_1	14:30 12/12/2017
u_1	l_3	13:05 14/12/2017
u_2	l_1	13:10 11/12/2017
u_2	l_1	15:10 14/12/2017
u_3	l_1	14:20 11/12/2017
u_3	l_2	13:16 12/11/2017
u_3	l_3	16:20 14/12/2017
u_4	l_2	15:15 11/12/2017

Location	Category	Coordinates
l_1	Sports	42.99,-71.46
l_2	Arts	42.98,-71.45
l_3	Sports	42.97,-71.44

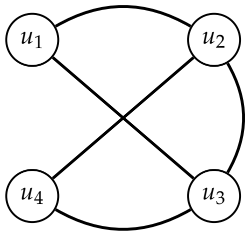


Fig. D.1: Toy example: Checkins (left), Location categories (top right) and social graph (bottom right)

frequency of *users'* visits at the given location(s), at locations of similar *categories*, as well as visiting *times* and *distances* among users' current locations and the location(s) for which visits are to be predicted. Then, we decompose these matrices into four non-negative low-rank matrices, i.e, H_U, H_C, H_T, and H_D, respectively, and a common latent space matrix W, so that $W \times H_U$, $W \times H_C$, $W \times H_T$, and $W \times H_D$ give approximated scores according to user's preferences. We use a linear combination of these matrices to form a score matrix WH_X that represents the potential of each user to visit the given location ("visit score"). We consider all users having significant visit scores and find their friends that are presumably influenced by their activities and likely to follow them in the given time period. To do so, we compute the *influence probabilities* of these users on their friends in the LBSN by a Bernoulli-distribution-based partial credit distribution and time constraint model. Users having a significant visit score, along with their significantly influenced friends, are considered as potential visitors to the location. It is worth noting that a user u may not influence another user v alone, yet, taken together with another user x, they may influence v; that is the reason why we first find all the users having a significant visit score and then fetch their influenced visitors.

In summary, we make the following contributions.

- We find the features that can be utilized for finding visitors at a location in LBSNs.

- We propose a visitor prediction model based on collective non-negative matrix factorization and influence propagation (CMViP).
- We provide an extensive experimental evaluation of our proposed methods on real-world datasets to showcase their precision and accuracy.

The rest of the paper is organized as follows. Section 2 covers related work in the domain. Section 3 provides preliminaries, mobility analysis of users in LBSNs and give details on CMViP. We provide details on our solution framework in Section 4, present evaluation results in Section 5, and conclude in Section 6.

2 Related Works

In this section, we survey existing studies on LBSN-based recommendations. We divide these studies into two parts, namely *recommendations to users* and *visitor prediction*.

2.1 Recommendations to Users

This group incorporates four types of recommendations [1].

1. *Location Recommendation* recommends locations to users for their potential visit, utilizing users' profile information such as historic visits, geo-tagged contents [2–4]. Collaborative filtering is widely used to provide such recommendations [5–7].
2. *Activity Recommendation* recommends to users one or more activities that are appropriate to perform at their location, fetching the users' current locations by their geo-tagged contents [8–10] and using inference-based and collaborative-filtering-based models. Due to data sparsity, the latter outperforms the former [11, 12].
3. *Social Media Recommendation* suggests to users digital contents such as photos, videos, and posts, utilizing spatial keyword search [13–15].
4. *User Recommendation* has been extensively explored from different perspectives, such as popular user discovery, friend recommendation, and community detection. Traditional LBSN approaches use users' geographical influence to model groups of friends through collaborative filtering [16] or factorization models [17, 18].

All aforementioned studies provide recommendations to LBSN users, yet our objective is to *predict* users at given locations. None of these studies can be applied to our objective.

2.2 Visitors Prediction

Some past studies have attempted to answer visitor prediction queries. Lasek et al. [19] focus on predicting visitors to locations with applications such as estimating restaurant sales and demands. Koshiba et al. [20] use a Bayesian network to predict users at a location, based on location categories, conditions of purchase, and demographic information of visitors. Sellers and Shmueli [21] propose Poisson regression models [22, 23] to predict the number of customers at a restaurant during a certain time period. Similarly, Morgan and Chintagunta [24] apply a regression model to predict the number of visitors throughout a calendar year.

Nevertheless, all the aforementioned approaches fail to take into consideration the sequential transitions from one location to another. To tackle this challenge, some works exploit Markov chain models [25, 26], factorization models to define the personalized sequential information [27], or both, in a Factorized Personalized Markov Chain (FPMC) [28], as well as Metric Embeddings to model the user preferences and POI transitions [29].

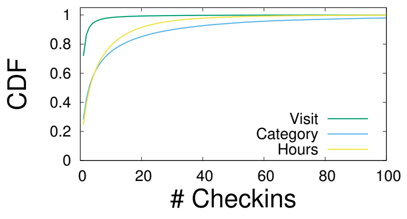
Most recently, Feng et al. [30] proposed an embedding model, POI2Vec, that predicts visitors to a Point of Interest (POI). POI2Vec modifies the Word2vec technique for word embeddings to learn POI representations by considering their geographical influence, user preferences, and sequential transitions, using the GPS coordinates of locations and visiting timestamps. POI2Vec is the most relevant previous work to ours, as it also aims to predict visitors at a location using LBSNs. However, our approach is technically different. While POI2Vec is based on a vector embedding of locations, we employ a collective matrix factorization to find joint embeddings of users' historic visits, categories, time, and distance features. Furthermore, we combine the prediction outcome of the factorized model with an influence propagation method to fetch friends of users that are likely to visit a certain location as well. Our comparison in Section 5.5 shows that our proposed model CMViP outperforms all variants of POI2Vec in both accuracy and precision.

3 Problem Formulation

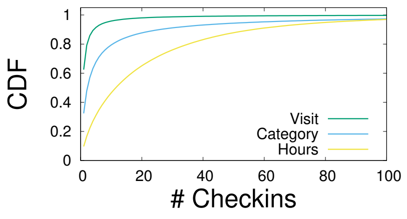
Here, we define preliminaries, present a mobility analysis of users, define the CMViP model, and formulate our problem.

3.1 Preliminaries

Definition D.1. *A **point of interest** is a geographical location (e.g., an amenity) represented by a quadruple (l, lat, lon, C), where l is the identifier, lat and lon are the latitude and longitude of the GPS coordinates of the center of the POI, and $C \subseteq$ **Cat** is a set of categories that this location belongs to (e.g., "Food", "Restaurant", and*



(a) Foursquare



(b) Wee

Fig. D.2: CDF of visits w.r.t. locations, categories and times

"French cuisine". Such categories are usually assigned by the visitors, and may thus change dynamically. However, we consider each location to have a fixed set of categories. A set of POIs is denoted as L.

Definition D.2. *A **user** is a person that visits POIs. A set of users is denoted by U.*

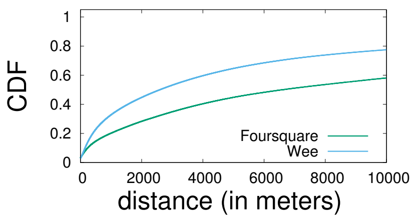
Definition D.3. *An **activity** refers to a visit or check-in of a user $u \in U$ at a location l at a discretized time interval t, represented as a triplet (u, l, t). The set of all activities over U and L is denoted $A(U, L)$.*

Definition D.4. *An **Location-Based Social Network (LBSN)** over U and L consists of a graph $G_S(U, F)$, called the* social graph, *where $F \subseteq \{\{u, v\} | u, v \in U\}$ represents friendships between users, and set of activities $A(U, L)$. It is denoted as $LBSN(G_S, A)$.*

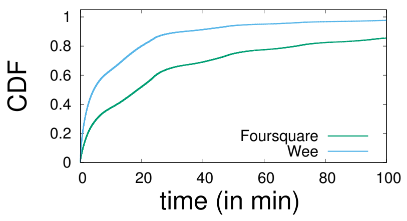
3.2 Users' mobility analysis in LBSNs

We now explore the features affecting user mobility on real-world LBSNs data. We further show the impact of these parameters in predicting visitors in Section 5.

We assume that users tend to visit locations of the same categories. To investigate this hypothesis, we compute the cumulative distribution function (CDF) of the visit frequency of users per locations and their visit frequency of users per category. Figure D.2 shows that the frequency of check-ins in at least one location is higher than that of check-ins in at least one category. This fact implies that users visit locations of the same category multiple times. Similarly, users visit several locations within the same time interval (for which we use hour). Both datasets we examine, i.e., Foursquare and Wee, exhibit the same behavior.



(a) Distance

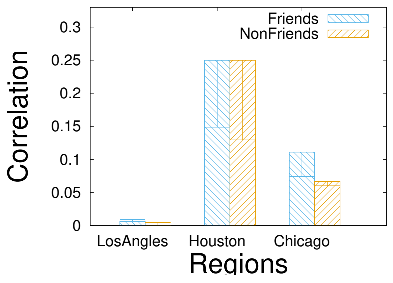


(b) Time

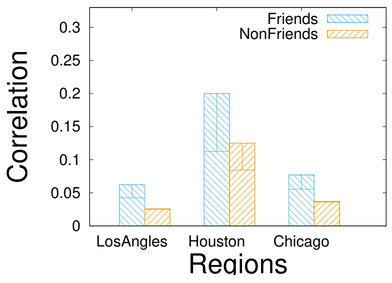
Fig. D.3: Distance and Time difference between checkins

Next, we analyze distance and time gaps between consecutive visits. We first consider the distance which users travel to visit locations. The CDF of the distance is given in Figure D.3a. In the Foursquare data, around 50% of the time, users travel no more than 4 km from one location to the next, while for about 20% of check-ins, users travel more than 10km. We deduce that users prefer to travel to nearby places. The behavior on Wee data is similar. Last, we consider the time intervals after which users travel to a new location. Figure D.3b shows the time CDF. In Wee, around 90% of the time, users check-in at a new location within 30 minutes of their last check-in.

Next, we evaluate the extent to which one's friends are influenced to visit the same location. We utilize an influence maximization approach described in Section 3.3. We compute the influence users exercise on their friends to perform the same activities within a specified time, i.e., 1 hour. We first examine whether users tend to perform the same activities as their friends do; to that end, we derive the correlation of activities of friends and non-friends as a Jaccard index. Considering that a user has fewer friends than non-friends,



(a) Foursquare



(b) Wee

Fig. D.4: Activity correlation among friends and non-friends

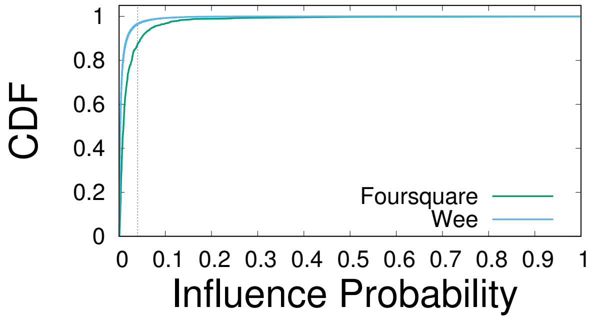


Fig. D.5: CDF of Influence Probabilities

we only capture check-ins within a limited territory. Yet a limited territory has a limited number of POIs, so the chance for a person to visit all of them is quite high. To ameliorate this effect, we consider three highly populated US cities, where the POI density is higher. Figure D.4 shows the results. The average correlation of activities among friends is up to 3 times higher than that of non-friends. Both datasets exhibit similar behavior.

These results show that users tend to follow the activities of their friends. We can then compute a suitable value of the influence probability among friends. To do so, we compute the CDF of the influence probabilities of users. Figure D.5 shows the result. An influence probability value of 0.04 (shown by a dotted line) already covers 90% of users. We then set a threshold for influence probability at 0.04 and consider those users that are influenced to follow their friends' activities with a probability higher than that threshold, i.e., 10% of users.

3.3 CMViP

By the analysis given in Section 3.2, we utilize five features to predict a visit of a user u to a location l: (i) historic visit frequency of u at l, (ii) historic visit frequency of u in categories of l, (iii) visit frequency of u at time of day T (iv) distance of l from current location of u, and (v) influence of u on friends. Next, we provide the scores for each of these features.

Visit frequency score

People tend to visit locations of their interests [31]. We evaluate the interest of a user u in a location l based on a *visit frequency score*, computed as follows.

$$\Upsilon_U(u,l) = \frac{|A(u,l)|}{|A(u)|} \tag{D.1}$$

where $A(u,l)$ is the set of activities of u at l and $A(u)$ is all activities of u.

Example 3.1
Consider the toy example, given in Figure D.1. Here, $\Upsilon_U(u_1,l_1) = 2/3, \Upsilon_U(u_2,l_1) = 2/2, \Upsilon_U(u_3,l_1) = 1/3$ and $\Upsilon_U(u_4,l_1) = 0$

Location category score

People like to visit locations of their general interest, identified by a location's category, such as "museum" and "Chinese restaurant". To incorporate this effect we compute the *location category score*, given by the following equation.

$$\Upsilon_C(u,c) = \frac{\sum_{l|c \in l.C} |A(u,l)|}{|A(u)|} \tag{D.2}$$

where $l.C$ is the set of categories of l.

Example 3.2
Consider the running example, given in Figure D.1. Let us denote $Sports$ with c_1. Here, $\Upsilon_C(u_1,c_1) = 3/3, \Upsilon_C(u_2,c_1) = 2/2, \Upsilon_C(u_3,c_1) = 2/3$ and $\Upsilon_C(u_4,c_1) = 0$.

Visit time score

Another feature that drives the visits of users is time. Usually, users visit a location at similar times of the day [31]. We compute the potential of a user to visit a location at a given time (considering time in a granularity of hours) by a *visit time score*, as follows.

$$\Upsilon_T(u,T) = \frac{|\{(u,l,t') \in A | t' = T\}|}{|A(u)|} \tag{D.3}$$

where T represents time in the form of an hour of a day.

Example 3.3
Consider the running example, given in Figure D.1. We take hours of visit time for computing Y_t. Let's $T_1 = 1300$, then $Y_T(u_1, T_1) = 2/3, Y_T(u_2, T_1) = 1/2, Y_T(u_3, T_1) = 1/3$ and $Y_T(u_4, T_1) = 0$.

Distance score

People tend to visit nearby places [31]. To capture this effect, we measure the distance between the current location of the user and the location which we aim to evaluate. We utilize the GPS coordinates of locations to measure distances and compute a *distance score* by the following equation.

$$Y_D(u, D) = 1 - \frac{D(l_c, l)}{D_{max}} \tag{D.4}$$

where l_c represents the current location of user u, l shows the location for which the visit is to be predicted, $D(l_c, l)$ shows the distance between l_c and l, and D_{max} is the maximum distance u traveled to visit any location.

Example 3.4
Consider the running example, given in Figure D.1. Let the location for which we aim to predict visitors is l_1, then $Y_D(u_1, l_1) = 1 - \frac{1}{1} = 0$. Similarly, $Y_D(u_2, l_1) = 0, Y_D(u_3, l_1) = 0.5$ and $Y_D(u_4, l_1) = -\infty$.

Next, we combine all these aforementioned scores using a linear combination to compute the overall potential of users. We call this potential a *visit score* denoted by Y_S. It is given by:

$$\begin{aligned} Y_S(u_1, l_1, c_1, T_1) = \alpha.Y_U(u_1, l_1) + \beta.Y_C(u_1, c_1) + \\ \gamma.Y_T(u_1, T_1) + \eta.Y_D(u_1, l_1) \end{aligned} \tag{D.5}$$

Here, α, β, γ, and η are the coefficients for visit frequency, location category, visit time, and distance respectively, showing their corresponding importance in predicting visitors. $\{\alpha, \beta, \gamma, \eta\} \in [0, 1]$, however, we provide optimal values for these parameters in Section 5. We utilize Equation D.5 to compute the visit score for all the users in LBSN. Then, we prune the users with less potential of visiting the location based on their visit score. To do that we use a threshold θ. All the users having visit score greater than θ are considered potential visitors U_P and given by:

$$U_P(l, c, T) = \{u | Y_S(u, l, c, T) \geq \theta\} \tag{D.6}$$

where θ is a threshold for visit score.

Example 3.5
Consider the toy example, given in Figure D.1. Let $\alpha = 0.25, \beta = 0.25, \gamma = 0.25, \eta = 0.8$, and $\theta = 0.8$. Then, $Y_S(u_1, l_1, c_1, t_1) = 0.84, Y_S(u_2, l_1, c_1, t_1) = 0.875, Y_S(u_3, l_1, c_1, t_1) = 0.66$ and $Y_S(u_4, l_1, c_1, t_1) = -\infty$.

Friends Influence

Users tend to follow the activities of their friends, as shown in Subsection 3.2. To capture this effect, we compute the influence of users on their friends, i.e., assess their potential to persuade friends to follow their activities.

We consider a user v to be influenced by his friend u if u visits a location l and v visits the same location after u within a particular time. To find such influence, we compute influence probabilities using Bernoulli distribution based on partial credit distribution and discrete time constraint models [32]. According to this model, the influence probability is measured as the ratio of the number of successful attempts to persuade the influenced user to follow the influencer user's activities over the total number of trials. Considering that a user can be influenced by multiple sources for an activity, the influence credit for each followed activity is distributed among all influencer users. Further, as the influence probability depends on time, we incorporate a discrete time constraint model which ensures that a user can influence other users only within a given time window. We consider the time window $\omega = 1$, as we capture the visit times in hours. The influenced friends of a user u are:

$$I(u) = \{v | p(u, v) \geq \xi \wedge (u, v) \in F\} \tag{D.7}$$

where $p(u, v)$ is the influence probability of u on v, ξ is a threshold representing minimum influence to persuade a user to follow an activity, and F is the set of friends pairs in an LBSN. A user v may not be influenced by a single user u, but by many other users taken together. Thus, we compute the influence of all potential visitors on all of their friends together, using the following equation.

$$I(U_P) = \{v | \sum_{u \in U_P} p(u, v) \geq \xi \wedge (u, v) \in F\} \tag{D.8}$$

where U_P is the set of users having a significant visit score. Thus, $I(U_P)$ are also considered as potential visitors. So, the total potential visitors are given by the union of U_P and $I(U_P)$ It is given by: $U_P \cup I(U_P)$.

Example 3.6
Consider the toy example, given in Figure D.1. Let $U_P = \{u_1, u_2\}, \omega = 1h$ and $\xi = 0.2$. Then, $I(U_P) = \{u_3\}$ as $p(U_P, u_3) = 0.3 \geq \xi$ Because, u_3

follows u_1 and u_2 for visiting l_1 and follows u_1 for visiting l_3 within ω. So, the predicted visitors are $\{u_1, u_2, u_3\}$.

3.4 Problem Statement

Next, we formally define our problem statement as follows:

Problem D.1. *Given a LBSN(G_S, A), a location l, the categories C of l, and a time interval T, predict the users that will visit l within T.*

Once we obtain potential visitors, we can utilize it for several other use-cases such as finding the number of visitors at a location or finding the types of visitors etc.

4 Solution Framework

In this section, we present our solution for visitor prediction using non-negative collective matrix factorization and influence propagation mechanisms. We explain the steps involved in the detail below.

4.1 Construction of Matrices

For each feature that derives the users' mobility (identified in Section 3.3), we construct a two-dimensional frequency matrix. For visit frequency, we construct a user-location frequency matrix: $Y_U \in \mathbb{R}^{U \times L}$ where U is the set of users and L is the set of locations. Each element of Y_U represents the number of times a user has visited the corresponding location. Similarly, we define visitor-category matrix $Y_C \in \mathbb{R}^{U \times C}$ which represents frequency of users at location categories, visitor-time matrix: $Y_T \in \mathbb{R}^{U \times T}$ shows the frequency of users' visit at hours of the day, and visitor-distance matrix: $Y_D \in \mathbb{R}^{U \times D}$ presenting the distance of user from their current locations to the other locations they visited. The values of each item in these matrices lie in the range $[0, \infty]$. The first column of Figure D.6 presents the frequency matrices for the toy example given in Figure D.1.

4.2 Initialization

The input matrices may possess a different range of values based on their features, as seen in Section 4.1. This may result towards a local optimal solution. However, we are interested in a globally-optimal solution. Thus, we normalize all the matrices in a row-wise fashion using $L2$-norm. Hence, we attain all the values within the range of $[0, 1]$.

Y_U	l_1	l_2	l_3
u_1	2	0	1
u_2	2	0	0
u_3	1	1	1
u_4	0	1	0

$\approx$

W	f_1	f_2	f_3
u_1	0.20	0.29	0.52
u_2	0.21	0.28	0.50
u_3	0.25	0.27	0.48
u_4	0.19	0.10	0.16

$\times$

H_U	l_1	l_2	l_3
f_1	0.16	0.12	0.07
f_2	0.21	0.08	0.09
f_3	0.39	0.12	0.16

Y_C	c_1	c_2
u_1	1	0
u_2	1	0
u_3	1	1
u_4	0	1

$\approx$

W	f_1	f_2	f_3
u_1	0.20	0.29	0.52
u_2	0.21	0.28	0.50
u_3	0.25	0.27	0.48
u_4	0.19	0.10	0.16

$\times$

H_C	c_1	c_2
f_1	0.17	0.13
f_2	0.23	0.09
f_3	0.42	0.14

Y_T	t_1	t_2	t_3	t_4
u_1	1	1	0	0
u_2	1	0	1	0
u_3	1	1	0	1
u_4	0	0	1	0

$\approx$

W	f_1	f_2	f_3
u_1	0.20	0.29	0.52
u_2	0.21	0.28	0.50
u_3	0.25	0.27	0.48
u_4	0.19	0.10	0.16

$\times$

H_T	t_1	t_2	t_3	t_4
f_1	0.13	0.08	0.12	0.04
f_2	0.17	0.11	0.09	0.04
f_3	0.31	0.20	0.15	0.08

Y_D	l_1	l_2	l_3
u_1	0	3	1
u_2	0	3	1
u_3	0	2	3
u_4	2	0	1

$\approx$

W	f_1	f_2	f_3
u_1	0.20	0.29	0.52
u_2	0.21	0.28	0.50
u_3	0.25	0.27	0.48
u_4	0.19	0.10	0.16

$\times$

H_D	l_1	l_2	l_3
f_1	0.07	0.15	0.14
f_2	0.02	0.21	0.14
f_3	0.03	0.38	0.24

Fig. D.6: Example of Collective Non-neg. Matrix Factorization

4.3 Non-Negative Collective Matrix Factorization

The LBSN data is often sparse (shown in Table D.1). This badly affects the prediction accuracy by emphasizing more on the available data. To avoid this, we use matrix factorization which leads to finding hidden latent representations in a common space. Moreover, we have multiple features: visit frequency, categories, time and distance to consider thus, we need to collectively factorize them. To do that, we use collective matrix factorization. Furthermore, since we do not anticipate any negative values in our data, we use non-negative collective matrix factorization. To do so, we decompose each of the four frequency matrices mentioned above into a common latent space matrix: W and corresponding feature latent space matrix H. Each row of W represents the relation between a visitor and the number of latent factors. Similarly, each column of H represents the relation between the number of latent factors and the features. For example in Figure D.6, Y_U is decomposed into W and H_U.

Next, we formally define the non-negative collective matrix factorization for the four frequency matrices:

$$\begin{aligned} min: f(W) = \frac{1}{2}[\alpha\|Y_U - WH_U\|_2^2 + \beta\|Y_C - WH_C\|_2^2 \\ +\gamma\|Y_T - WH_T\|_2^2 + \eta\|Y_D - WH_D\|_2^2 \\ +\lambda(\|W\|_2 + \|H_U\|_2 + \|H_C\|_2 + \|H_T\|_2 + \|H_D\|_2)] \\ s.t. W \geq 0, H_U \geq 0, H_C \geq 0, H_T \geq 0, H_D \geq 0 \end{aligned} \tag{D.9}$$

where W represents the common latent space during the decomposition of Y_U, Y_C, Y_T, and Y_D as observed in Figure D.6, while $\{\alpha, \beta, \gamma, \eta\} \in [0,1]$ are hyper-parameters that control the importance of each matrix during the factorization. Setting them as 0.25 gives equal importance to decomposition matrices, while different values give more importance to the factorization of Y_U (or Y_C, or Y_T, or Y_D). The remaining terms are Tikhonov regularization [33] of W,H_U, H_C, H_T, and H_D controlled by the hyper-parameter $\lambda \geq 0$. It is used to enforce the smoothness of the solution and avoid overfitting. Note, the matrices H_U, H_C, H_T, and H_D when multiplied by W give us an approximated score to the input matrices as seen in Figure D.6.

Multiplicative Update Rules

The proposed model applies multiplicative update rules as regularization term of H_U, H_C, H_T, H_D, and W. This technique updates the scores in each iteration until reaches the stationary point. Formalizing the update rules, we first have the partial derivatives of the functions as:

$$\begin{aligned} \nabla f(W) = \alpha WH_UH_U^T - \alpha Y_UH_U^T + \beta WH_CH_C^T - \beta Y_CH_C^T \\ +\gamma WH_TH_T^T - \gamma Y_TH_T^T + \eta WH_DH_D^T - \eta Y_DH_D^T + \lambda I_k \\ \nabla f(H_U) = \alpha W^TWH_U - \alpha W^TY_U + \alpha W^TH_U + \lambda I_k \\ \nabla f(H_C) = \beta W^TWH_C - \beta W^TY_C + \beta W^TH_C + \lambda I_k \\ \nabla f(H_T) = \gamma W^TWH_T - \gamma W^TY_T + \gamma W^TH_T + \lambda I_k \\ \nabla f(H_D) = \eta W^TWH_D - \eta W^TY_D + \eta W^TH_D + \lambda I_k \end{aligned} \tag{D.10}$$

where I_k is the identity matrix with $k \times k$ dimensions.

Calculating the derivatives of $f(W)$, $f(H_U)$, $f(H_C)$, $f(H_T)$, $f(H_D)$ from Equations (10) leads to following update rules:

$$\begin{aligned} W &= \frac{[\alpha Y_U H_U^T + \beta Y_C H_C^T + \gamma Y_T H_T^T + \eta Y_D H_D^T]}{[\alpha H_U H_U^T + \beta H_C H_C^T + \gamma H_T H_T^T + \eta H_D H_D^T + \lambda I_k]} \\ H_U &= (\alpha W^T W + \lambda I_k)^{-1} \odot \alpha W^T Y_U \\ H_C &= (\beta W^T W + \lambda I_k)^{-1} \odot \beta W^T Y_C \\ H_T &= (\gamma W^T W + \lambda I_k)^{-1} \odot \gamma W^T Y_T \\ H_D &= (\eta W^T W + \lambda I_k)^{-1} \odot \eta W^T Y_D \end{aligned} \tag{D.11}$$

where $\odot$ and $\frac{\bullet}{\bullet}$ corresponds to the element-wise matrix product and left division, respectively.

Each iteration of the proposed model gives us a solution for the pair-wise division. As we map any negative values to zero, the W matrix becomes non-negative after each update. Furthermore, the objective function and the delta decrease on each iteration of the above update rules, guaranteeing the convergence into a stationary point.

4.4 Optimization

The proposed model applies alternating least squares optimization to minimize the objective function [34], which performs as follows: (1) fix the value of W while minimizing $f(W)$ over H_U, H_C, H_T, H_D; then (2) fix the value of H_U, H_C, H_T, H_D while minimizing $f(H_U)$, $f(H_C)$, $f(H_T)$, and $f(H_D)$ over W. Considering a matrix with y_i rows and y_j columns, with a relation defined by v_ij, we can define the correlation among n neighbors' data points. This results in a matrix A which can later be used to measure the local closeness of two data points y_i and y_j.

Collective factorization reduces data points y_i from a matrix Y, into a common-latent space W as w_i. The distance between two low dimensional data points is calculated using the Euclidean distance: $\|w_i - w_j\|^2$, and mapped into a matrix $\mathcal{A}$. Based on the matrix $\mathcal{A}$, we can iteratively run these two steps until the stationary point, or until the established number of maximum iterations as follows:

$$\begin{aligned} M &= \frac{1}{2} \sum_{i,j=1}^{n} \|w_i - w_j\|^2 \mathcal{A}_i j \\ &= \sum_{i=1}^{n} (w_i^T - w_i) D_i i - \sum_{i,j=1}^{n} (w_i^T - w_i) D_i i \\ &= Tr(W^T D W) - Tr(W^T A W) = Tr(W^T L W), \end{aligned} \tag{D.12}$$

where $Tr(\bullet)$ denotes the trace function, and D is a diagonal matrix whose entries are row sums of $\mathcal{A}$ (or column, as $\mathcal{A}$ is symmetric), i.e., $D_i i = \sum_i \mathcal{A}_{ij}$;

$L = D - \mathcal{A}$ is called the Laplacian matrix, we need to incorporate it to enforce the non-negative constraints.

The problem of optimizing $f(W)$ can be re-written as:

$$\begin{aligned} min : f(W) = \frac{1}{2}[\alpha\|Y_U - WH_U\|_2^2 \\ +\beta\|Y_C - WH_C\|_2^2 + \gamma\|Y_T - WH_T\|_2^2 \\ +\eta\|Y_D - WH_D\|_2^2 + \varphi Tr(W^T LW) \\ +\lambda(\|W\|_2 + \|H_U\|_2 + \|H_C\|_2 \\ +\|H_T\|_2 + \|H_D\|_2)] \\ s.t. W \geq 0, H_U \geq 0, H_C \geq 0, H_T \geq 0, H_D \geq 0 \end{aligned} \tag{D.13}$$

where L is the Laplacian matrix, and φ is a hyper-parameter which controls objective function's extent. The hyper-parameters α, β, γ, η and λ have the same semantics as in Eq. 9.

4.5 Prediction of Visitors

The preferences scores for a user u is given by $s_i = WH_x$, where W are the factors in the common latent space that explain the preferable places of $u_i \in U$, and H_x represents the relation among the output matrices H_U, H_C, H_T, and H_D as shown in Eq. 14.

$$\begin{aligned} WH_X = \alpha.WH_U(u,l) + \beta.WH_C(u,l) \\ +\gamma.WH_T(u,l) + \eta.WH_D(u,l), \end{aligned} \tag{D.14}$$

where, + corresponds to the element-wise sum, while $\{\alpha, \beta, \gamma, \eta\} \in [0,1]$ are hyper-parameters controlling the importance of each factorized matrix. The sum of hyper-parameters are set to be 1.

The visit score of a user is given by WH_X. It is worth noting that Equation D.14 provides an approximate version of Equation D.5. Next, we prune all the users having less visit score than the threshold θ. Users having visit score more than θ are considered potential visitors U_P given in equation D.6. We further utilize them to find their influenced visitors using the algorithm given in Section 3.3. We combine the potential visitors with their influenced visitors and predict them as visitors of the location.

5 Experimental Evaluation

In this section, we provide a detailed experimental evaluation of the solutions using two real-life data sets. We first describe the datasets, then we present the data prepossessing and preparation. Finally, we provide results for our experimental evaluation.

	Users	Locations	Checkins	POIs	Friend pairs	Duration
FourSquare	4K	0.2M	0.47M	0.12M	32K	1322 days
Wee	16K	0.9M	8M	0.76M	0.1M	2796 days

Table D.1: Dataset Statistics

5.1 Datasets

We utilize two real-world datasets taken from Foursquare and Weeplaces [35]. The datasets were chosen considering two scenarios to analyze CMViP performance, where Foursquare is small and dense, while Wee is large and sparse. Table D.1 shows the statistics of the data. Each of these datasets consisted of three parts: the social friendship graph, an ordered list of check-ins, and a collection of Venues. A check-in record contains the user-id, check-in time, GPS coordinates, and a location-id. Venues provide the details of locations, i.e., city, country, and semantic categories of those locations.

Data Prepossessing. The real-life nature of the datasets requires extensive preprocessing, as many locations are associated with multiple location identifiers, each having slightly different GPS coordinates. Consider, for instance, Figure D.7. This figure shows 13 GPS coordinates that appear in the Foursquare dataset; these coordinates correspond to different locations Ids in that dataset, but they clearly belong to the same unique physical location. In order to address this multiplicity issue, we clustered GPS points to get POIs. We used a grid-based spatial clustering with a grid of size 10 meters $\times$ 10 meters, so as to group GPS points. Then, we assigned a new, unique location Id to each resulting cluster; these ids are then used in all our experiments. All two datasets presented similar multiplicity problems, which we addressed in the same manner. Statistics regarding these new POI Ids are reported in Table D.1.

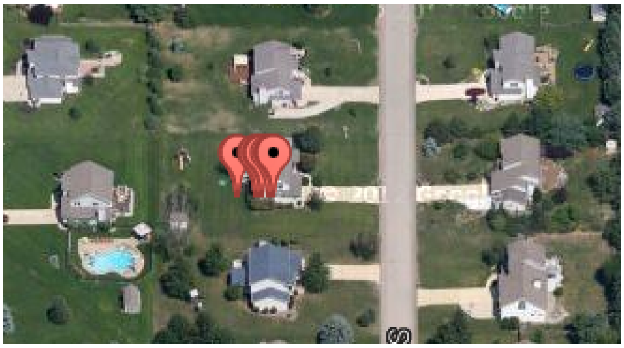


Fig. D.7: GPS coordinate of 13 location-ids on GoogleMaps

5.2 Evaluation Measures

We first evaluate our proposed methods for prediction of visitors for a location. To do so, we measure Precision and Recall. Then, we further evaluate for one use-case: predicting the number of visitors of location. For this purpose, we measure RMSE and MAE. In order to evaluate CMViP based on these measures, we utilize the cross-validation approach. We divide our datasets based on time stamp of check-ins such that each part contains check-ins of one month. We train our model on a dataset of one month and test its performance on the data set of next month. We iteratively perform this for all the parts of the datasets and then show the average results.

We consider the check-ins of training dataset to construct the queries which are composed of locations, categories and timestamps of check-ins. Moreover, for each of these queries, we predict the visitors using CMViP. With this we obtain four possible outcomes for users for the input parameters: visitors, non-visitors, predicted, not predicted. We use these values to construct confusion matrix and compute precision and recall based on it.

We follow a similar approach to calculate the error, where we check the number of visitors against a query in the test dataset in comparison to the training dataset. We use this information to calculate MAE and RMSE.

5.3 Parameter Analysis

CMViP uses several hyper-parameters while computing the visit score and incorporating the influence of potential visitors as shown in Equations D.5, D.6 and D.7. In this section, we provide optimal values of these parameters.

During the factorization process in Section 4, we use k: the number of latent factors; α, β, γ, and η: coefficients of visit score, frequency score, time score and distance score, respectively; and λ: for controlling the smoothness of the solution. The parameter k controls the number of factors considered by the system, consequently the complexity of the model. Small values of k under-fit, on the other hand, large values of k over-fit the data and lead to poor performance. We evaluate CMViP for a set of different values and consider the value $k = 50$, which provide us most optimal values. Similarly, to find the suitable values of the coefficients α, β, γ, and η we perform experiments with several combinations of values for each coefficient. The most optimal values are $\alpha = 0.9$, $\beta = 0.09$, $\gamma = 0.009$, and $\eta = 0.001$. This show that the visit score is the most important feature in predicting the visitors. λ provides the smoothness of the solution and avoid overfitting. The larger value of λ oversimplify the model and decrease the performance, however, the lower values keep the accuracy stable. The most optimal value we observed is $\lambda = 0.5$.

Next, we evaluate the optimal value for θ, which shows that what should be the visit score for considering a user a potential visitor. θ controls the number of predicted visitor: less the value of θ is more the number of predicted visitors we get. The optimal value of $\theta = 0.4$. However, we show the behavior with respect to different values of θ for the evaluations. Furthermore, we find the optimal value of influence probability threshold. We assume that users influenced with probability greater than this threshold will follow their influential friends' activities. We consider the value of this threshold 0.04 as we assume that the top 10% the most influenced visitors follow the activities. We use these values of the parameters for evaluating CMViP.

5.4 Competitors

We compare the performance of CMViP against four different variants of POI2Vec, the latest state-of-the-art method.

- PI2Vec-**U**: utilizes user's preferences.
- PI2Vec-**A**: considers only users with recent locations.
- PI2Vec-**UA**: incorporates both preference and recent locations.
- PI2Vec-**MUA**: applies aggregation to incorporate preferences and transition among locations.

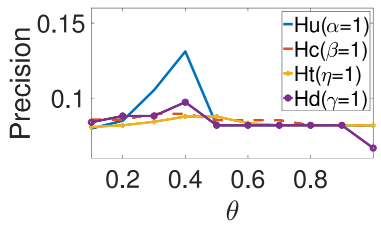
We use the optimal parameter values for each variant.

5.5 Results

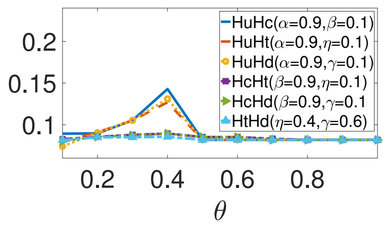
We evaluate the effect of features considered in Section 4.

Precision We first evaluate the precision. Figure D.8 presents the precision of CMViP according to different θ values for Wee dataset [1]. Figure D.8a presents the results considering a single feature, where we observe the user visit frequency score WH_U has higher precision than the other matrices: visit category WH_C, visit time WH_T, and visit distance WH_D. In figure D.8b we combine two matrices considering the importance degree is given by the first experiment with only one matrix. Overall, precision increases with two features. More specifically, combining visit frequency WH_U and visit category WH_C, give us the best precision among all the combination of two features. This presents, the significance of historic check-ins and categories for users' prediction. Figure D.8c presents the results for combining three matrices, where we observe an improvement in the precision when we combine WH_U,

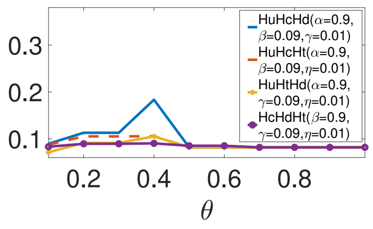
[1]Due to space constraints results for Foursquare are placed in the Appendix of Technical Report (`https://goo.gl/UA9Jzc`).



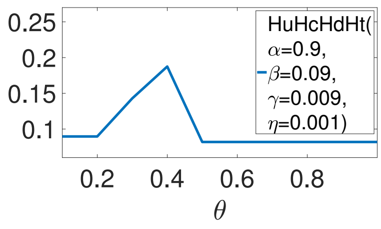
(a) One Matrix



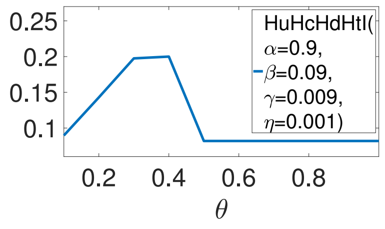
(b) Two Matrices



(c) Three Matrices



(d) Four Matrices

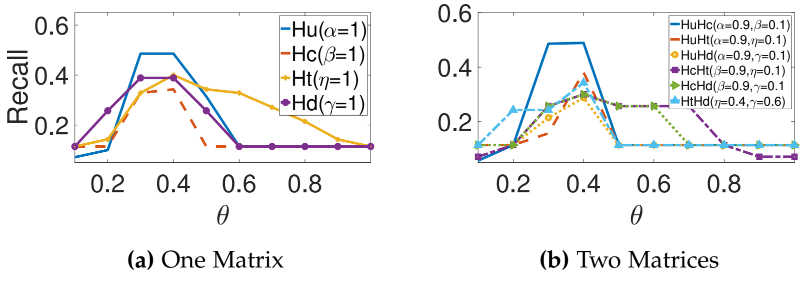


(e) FiveMatrices

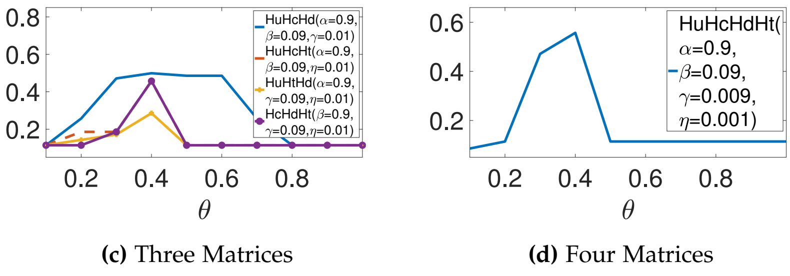
Fig. D.8: Precision for Wee dataset

WH_C, and WH_D. Combining WH_U, WH_C, WH_D, and H_T presented by Figure D.8d shows a slight improvement, due to incorporating visit time. Finally, in Figure D.8e we achieve the maximum precision by considering influenced friends of the users' who have visit score greater than θ. The improvement of 2% in precision shows the significance of considering influenced visitors for visitor prediction.

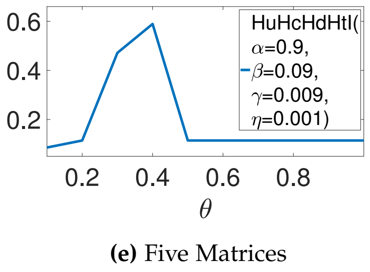
Recall We present the recall measure with respect to different values of θ as shown in Figure D.9. Figure D.9a presents the results considering single features WH_U, WH_C, WH_D, or H_T. Here, we observe that the users' historical check-ins WH_U have the highest recall than other matrices WH_c, WH_t, and WH_d. Figure D.9b combines two matrices, where WH_U and WH_C give us better recall than the previous experiment, showing us user check-ins and categories when combined improve the model's recall. It is worth noting



(a) One Matrix **(b)** Two Matrices



(c) Three Matrices **(d)** Four Matrices

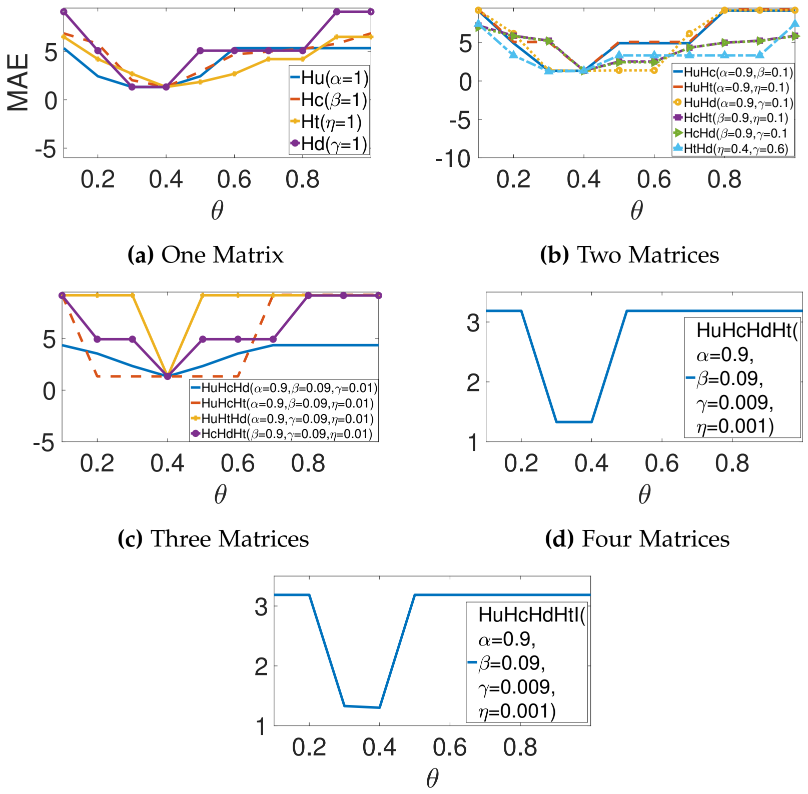


(e) Five Matrices

Fig. D.9: Recall for Wee dataset

that due to higher recall from WH_U, this experiment gives higher importance degree to WH_U than categories. Figure D.9c presents the improved recall results when combining three matrices, WH_U, WH_C, and WH_D. The results of incorporating all features: WH_U, WH_C, WH_D, and H_T is shown in Figure D.9d, with an improvement of up to 5%. Last, incorporating influenced visitors improves recall by up to 60% as shown in Figure D.9e.

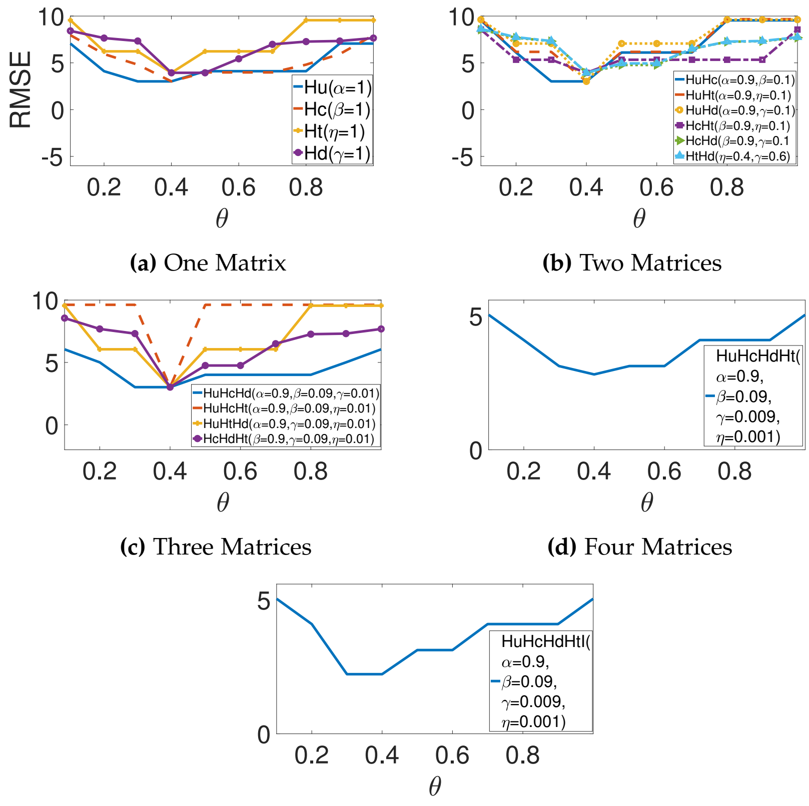
Mean Absolute Error (MAE) Furthermore, we evaluate the performance of CMViP for predicting the number of visitors. To do so, we first compute the mean absolute error (MAE) with respect to different values of θ as shown in Figure D.10. Here, our goal is to minimize the MAE value. Figure D.10a presents the results considering all the features individually. Users' historical check-ins WH_U presented the lowest MAE than other features: WH_C, WH_T, and WH_D. Figure D.10b depicts the results when two features are com-



(a) One Matrix **(b)** Two Matrices **(c)** Three Matrices **(d)** Four Matrices **(e)** Five Matrices

Fig. D.10: MAE for Wee Dataset

bined, where, by combining WH_U and WH_C the MAE is decreased. Figure D.10c presents the results for combining three matrices. We observe further decrease in MAE when we combine WH_U, WH_C, and WH_D. Similarly, by incorporating H_T further decrease the MAE. Ultimately, the minimum error is attained by incorporating influence visitors as shown in Figure D.10d. It is worth noting that the curve presents a parabolic curve, which shows that both before and after the optimal value of θ our error increases. This happens because if the value of θ is less we incorporate many users that have less visit score and do not really visit the location. On the other hand, when the value of θ is greater than optimal value, we consider a less number of potential visitors and thus miss many visitors that actually visit the location which increases the MAE value.



(a) One Matrix

(b) Two Matrices

(c) Three Matrices

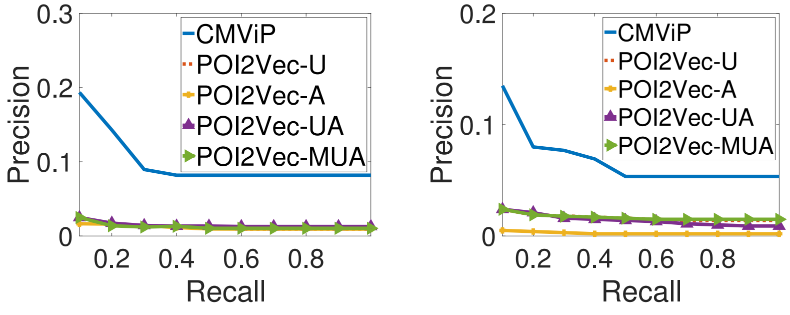
(d) Four Matrices

(e) Five Matrices

Fig. D.11: RMSE for Wee dataset

Root Mean Square Error (RMSE) Figure D.11 presents the root mean square error (RMSE) with respect to θ. Results by considering all feature individually are presented in Figure D.11a, where we observe the user historical check-ins WH_U presents lower error than other features WH_C, WH_T, and WH_D. Similar to MAE, we decrease the RMSE value with the addition of more features. The minimum error is achieved when we combine all the features and incorporate influenced visitors. These results are shown in Figure D.11e. RMSE presents similar parabolic curve to MAE, where beyond and after an optimal value of θ RMSE values increase.

Performance Comparison Last, we compare the performance of CMViP with different variants of POI2Vec as given in Section 5.4. The parameters and their optimal values for CMViP are shown in Section 5.3. The value of



(a) Wee **(b)** Foursquare

Fig. D.12: Precision Recall Curve

parameters for POI2Vec are latent factors $k = 50$, score threshold $\theta = 0.4$ and learning rate is 0.05.

Figure D.12 presents the experimental results considering precision-recall curve, where we observed, CMViP outperforms **POI2Vec**: (1) **U**, (2) **A**, (3) **UA**, and (4) **MUA**. CMViP outperforms **U** to predict the potential visitors. Even though **U** uses user's preferences to predict visitors, it does not consider other features as categories, time, and distance. This means CMViP has better user modelling in the vectorial space. **U** first learns the location representation, and the user representation is learned when the location representation is fixed. However, as CMViP presented better performance than **U**, it shows the collective factorization and linear model is more reasonable for this task.

A considers only recent check-ins presenting a poor performance in comparison with other techniques since old check-ins present an important role to determine the future visitors. The results show it is important to consider both users with recent and old check-ins. **MUA** outperforms **UA**, indicating that combining user preference and sequential transition plays an important role to predict potential visitors. However, it under-performs CMViP, since this model uses categories, time, and distance to jointly predict the visitors. Furthermore, CMViP adds user's social influencers to improve its accuracy.

Observing the Figure D.12, we can conclude collectively learning the location and visitor's latent factors and further aggregating visitor's social influencers in a linear model, significantly improves the prediction given by CMViP. Although CMViP performs better than **POI2Vec**, it presents different results given the datasets. CMViP consistently improves the visitors' prediction for locations which have received a small number of visitors as observed on sparse dataset Wee. However, it performs worse for active visitors as presented on Foursquare dataset. This happens, because CMViP explicitly makes use of jointly process latent factors to learn better latent representations, especially when visitor-location history data is sparse. **POI2Vec** cannot accurately infer inactive visitors' preferences, what is a disadvantage since most of the datasets do not have active users, a problem is known in recommend systems as *long tail*.

Aggregating visitor's social influencers in a linear model present an important contribution since it has a direct impact on visiting popular locations. Predicting the most popular locations is not a hard task, however, combining factorization and influencers presents a better balance on the final prediction score, because CMViP predicts not only the relevant (sometimes rare locations) but also the popular locations. **POI2Vec** presents lower performance than CMViP, because does not consider the popular items given by influencers, showing a lower precision, but high recall.

6 Conclusion

We proposed a model, CMViP, that predicts visitors given a location, its category and visit time. We first identify features that derive mobility of users: visit frequency, category, time and distance. We analyze the LBSN data for analyzing and finding the importance of these features. CMViP employs non-negative collective matrix factorization to find the potential of visitors and further leverages these potential visitors to find their influence on their friends in social networks and thereby incorporate highly influenced visitors as potential visitors.

We have empirically shown that the CMViP outperforms the state-of-the-art approaches using real-world datasets of two different natures: 1) small and dense, and 2) large and sparse. We evaluate the performance based on two measures. We first evaluate the performance of CMViP for predicting visitors using precision and recall measures and then evaluate the accuracy of predicting the number of visitors using RMSE and MAE. Our results show that CMViP outperform state-of-the-art methods in precision and recall up to 10 times. We further show the significance of considered features for visitor prediction.

In the future, we plan to extend the proposed model considering more contextual features, such as week-days and week-ends. Further, we plan to improve performance by pruning users with less potential visitors.

References

[1] J. Bao, Y. Zheng, D. Wilkie, and M. Mokbel, "Recommendations in location-based social networks: A survey," *Geoinformatica*, vol. 19, no. 3, pp. 525–565, 2015.

[2] L. Del Prete and L. Capra, "differs: A mobile recommender service," in *MDM*, 2010, pp. 21–26.

[3] B. Liu, Y. Fu, Z. Yao, and H. Xiong, "Learning geographical preferences for point-of-interest recommendation," in *KDD*, 2013, pp. 1043–1051.

[4] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation," in *SIGIR*, 2011, pp. 325–334.

[5] D. Lian and X. Xie, "Learning location naming from user check-in histories," in *GIS*, 2011, pp. 112–121.

[6] L.-Y. Wei, Y. Zheng, and W.-C. Peng, "Constructing popular routes from uncertain trajectories," in *KDD*, 2012, pp. 195–203.

[7] Y. Zheng and X. Xie, "Learning travel recommendations from user-generated gps traces," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 1, pp. 2:1–2:29, 2011.

[8] Z. Yin, L. Cao, J. Han, C. Zhai, and T. Huang, "Geographical topic discovery and comparison," in *WWW*, 2011, pp. 247–256.

[9] A. Pozdnoukhov and C. Kaiser, "Space-time dynamics of topics in streaming text," in *LBSN*, 2011, pp. 1–8.

[10] A. Noulas, C. Mascolo, and E. Frias-Martinez, "Exploiting foursquare and cellular data to infer user activity in urban environments," in *MDM*, 2013, pp. 167–176.

[11] V. W. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang, "Collaborative filtering meets mobile recommendation: A user-centered approach," in *AAAI*, 2010, pp. 236–241.

[12] V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang, "Collaborative location and activity recommendations with gps history data," in *WWW*, 2010, pp. 1029–1038.

[13] M. Sarwat, A. Eldawy, M. F. Mokbel, and J. Riedl, "Plutus: Leveraging location-based social networks to recommend potential customers to venues," in *MDM*, 2013, pp. 26–35.

[14] Y. Chen, W. Wang, Z. Liu, and X. Lin, "Keyword search on structured and semi-structured data," in *SIGMOD*, 2009, pp. 1005–1010.

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

[16] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Who, where, when and what: Discover spatio-temporal topics for twitter users," in *KDD*, 2013, pp. 605–613.

[17] C. Cheng, H. Yang, I. King, and M. R. Lyu, "Fused matrix factorization with geographical and social influence in location-based social networks," in *AAAI*, 2012, pp. 17–23.

[18] X. Li, G. Cong, X.-L. Li, T.-A. N. Pham, and S. Krishnaswamy, "Rank-GeoFM: A ranking based geographical factorization method for point of interest recommendation," in *SIGIR*, 2015, pp. 433–442.

[19] A. Lasek, N. Cercone, and J. Saunders, "Restaurant sales and customer demand forecasting: Literature survey and categorization of methods," in *Smart City 360*, 2016, pp. 479–491.

[20] H. Koshiba, T. Takenaka, and Y. Motomura, "A service demand forecasting method using a customer classification model," in *The Philosopher's Stone for Sustainability*, 2013, pp. 281–285.

[21] K. F. Sellers and G. Shmueli, "Predicting censored count data with com-poisson regression," 2010.

[22] S. Coxe, S. G. West, and L. S. Aiken, "The analysis of count data: A gentle introduction to poisson regression and its alternatives," *Journal of personality assessment*, vol. 91, pp. 121–136, 2009.

[23] J. Wulu, K. Singh, F. Famoye, T. Thomas, and G. McGwin, "Regression analysis of count data," *Indian Society of Agricultural Statistics*, 2002.

[24] M. S. Morgan and P. K. Chintagunta, "Forecasting restaurant sales using self-selectivity models," *Journal of Retailing and Consumer Services*, vol. 4, pp. 117–128, 1997.

[25] J.-D. Zhang, C.-Y. Chow, and Y. Li, "Lore: Exploiting sequential influence for location recommendations," in *SIGSPATIAL*, 2014, pp. 103–112.

[26] J. Ye, Z. Zhu, and H. Cheng, "What's your next move: User activity prediction in location-based social networks," in *SIAM*, 2013, pp. 171–179.

[27] C. Cheng, H. Yang, M. R. Lyu, and I. King, "Where you like to go next: Successive point-of-interest recommendation," in *IJCAI*, 2013, pp. 2605–2611.

[28] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *WWW*, 2010, pp. 811–820.

[29] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, “Personalized ranking metric embedding for next new poi recommendation,” in *IJCAI*, 2015, pp. 2069–2075.

[30] S. Feng, G. Cong, B. An, and Y. M. Chee, “Poi2vec: Geographical latent representation for predicting future visitors,” in *AAAI*, 2017, pp. 102–108.

[31] E. Cho, S. A. Myers, and J. Leskovec, “Friendship and mobility: User movement in location-based social networks,” in *KDD*, 2011, pp. 1082–1090.

[32] A. Goyal, F. Bonchi, and L. V. Lakshmanan, “Learning influence probabilities in social networks,” in *WSDM*, 2010, pp. 241–250.

[33] A. N. Tikhonov, “Solution of incorrectly formulated problems and the regularization method,” *Soviet Math. Dokl.*, vol. 4, pp. 1035–1038, 1963.

[34] M. Saveski and A. Mantrach, “Item cold-start recommendations: Learning local collective embeddings,” in *RecSys*, 2014, pp. 89–96.

[35] Y. Liu, W. Wei, A. Sun, and C. Miao, “Exploiting geographical neighborhood characteristics for location recommendation,” in *CIKM*, 2014, pp. 739–748.

Paper E

Mining Geo-Social Cohorts in Location-Based Social Networks

Muhammad Aamir Saleem, Panagiotis Karras, Toon Calders, Torben Bach Pedersen

The paper is under revision
for a Conference publication

Abstract

Given a record of past geo-tagged activities and a web of social ties, how can we predict groups, or cohorts, *of future companions? Can our predictions go beyond groups that appear in the training data set? We propose that we can mine and predict such cohorts by leveraging information on two levels: (a) social bonds, and (b) past common activities. In particular, we introduce a prediction scheme that detects sets of users that (i) form cliques of friendships* — a constraint our experimental study shows is necessary; *and (ii) maximize a function of past common pairwise activities among their members. We show that mining such groups is an* **NP***-hard problem, and propose an efficient and nontrivial algorithm therefor, which works as if it were enumerating maximal social cliques, but guides its exploration by an activity-driven criterion in place of a clique maximality condition. In an experimental study with real-world data, we demonstrate that our methodology achieves high predictive power for future consecutive common activities, surpassing an adaptation of previous work in terms of quality and a brute-force baseline in terms of efficiency, while it correctly predicts future cohorts that do not appear in the training set.*

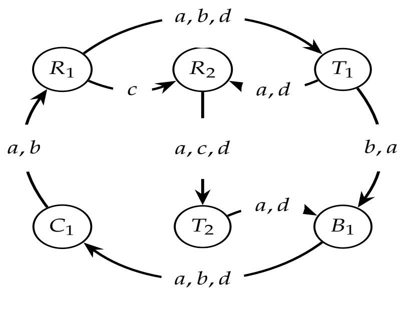
The layout has been revised.

1 Introduction

Advances in positioning and communication technologies enable the sharing of geo-tagged content in the web, and hence the provisioning of location-based social networking services. Weeplaces, Foursquare, Gowalla, GeoLife, and Twinkle are built around positioning capabilities, while Facebook and vKontakte provide users with the option to check-in at visited locations and thereby build a location history. Online social networks encompassing location information are called Location-Based Social Networks (LBSNs).

Such online LBSNs can capture the mobility of users and utilize it in recommendation services, even directed to *groups of users*, as opposed to individuals. For instance, a discount offer for a concert performance may be recommended to a group of friends that have habitually visited similar concerts and other events together; a car-pooling service may suggest group formations to its customers who may not be fully aware of their similar activities; or a travel agency may promote an offer for a group travel package to groups of users potentially interested in traveling together. Extant research has proposed location-based recommendations on locations, routes, users, activities, and media [1]. Some works refer to individual predictions [2], while others detect communities based on location preferences and mobility patterns; for example, [3] builds trajectory profiles and then clusters those profiles to discover users of similar behavior; similarly, [4] infers similarity of interests among users based on their location history, thereby building a hierarchy of user clusters. Yet these studies do not examine whether users both *act together* and are *socially linked*.

Well-connected groups of users are prone to form *dense* subgraphs, and such real-life social connectivity is manifested in two dimensions: companions are both *socially tied* — as we show, they form *cliques* in the social graph — and *also* often engage in *common activities* along with each other.



Users	Locations	Time
a,b,c,d	R_1	t_1
a,b,d	T_1	t_2
a,c,d	R_2	t_3
a,c,d	T_2	t_5
a,b,d	B_1	t_6
a,b,d	C_1	t_7
a,b	R_1	t_8
a,b	T_1	t_9

Fig. E.1: Example location history (L) and check-ins (R)

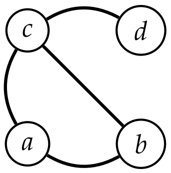


Fig. E.2: Social Graph (*undirected*) for running example

We illustrate the discussed concepts with an example in Figure E.1. The diagram on the left side shows the movements of four users across six locations, while the table on the right side shows the check-ins of those users at locations. Capital letters R, T, B, C represent categories of locations, namely *restaurant*, *transportation station*, *bar*, and *cinema* respectively, while subscripts indicate different locations within these categories. Lowercase letters a, b, c, d stand for users, while t_i's indicate check-in time stamps. Figure E.2 shows the social connections among those users. Users a, b form a cohort moving between categories R, T and B, C. Later, users a, b engage in similar behavior for categories R, T again. Thus, if we choose to observe which users move together from bars to cinemas, we can predict cohort $\{a, b\}$, which maintains such cohort behavior indeed. On the other hand, if we were interested in categories R, T, then we would detect cohort $\{a, c\}$, which does repeat such behavior in the rest of the data set. Notably, cohort $\{a, c\}$ appears only once up to time stamp t_7, contrary to two appearances of cohort $\{a, b\}$, hence we have lower confidence in the reappearance of the former.

We define an *activity graph* G_A as a graph *distinct* from the graph of social connections albeit *coterminous* in the vertex set, in which pairs of user nodes are connected by edges expressing the pairs' history of co-presence in common activities. A group of users who engage in such common activities frequently appear in G_A as a connected subgraph with high weighted edge density; in other words, such users form a *quasi-clique* of high normalized edge weight in G_A. In our example, those groups who form cohorts in terms of activities are also connected in the social graph. Yet, to our knowledge, no attempt has been made to discover groups by combining information about both social and mobility ties.

In this paper, we aim to mine *cohorts* of companions who are likely to move together *consecutively* in the future, and test the predictive power of our method. We conjecture that such companions are directly or transitively connected in terms of past common activities, but also in terms of tight social links. Therefore, we leverage information in the *social domain*, i.e., a graph of social friendship links, and in the *activity domain*, i.e. an activity graph G_A of common and consecutive geo-tagged activities. Our rationale is that, for a group of users to be predisposed to act in unison, it should exhibit both these kinds of connectedness. After all, a person may have a large circle of friends and acquaintances, yet may engage in specific activities only with particular ones, with whom they share interests related to those activities. Further, we utilize the *categories* of locations of interest, so as to tune detected cohorts to particular interests. For example, a group may have shown interest in *museums of modern art* and *extreme sports*; we can detect such a cohort and propose new locations to them.

We formulate the problem as one of mining sets of nodes that form subgraphs that maximize a certain edge density function in the activity graph G_A and also induce cliques in the social graph G. Our contributions are outlined as follows: (a) We introduce the problem of mining cohorts in LBSN data based on *subgraph density* criteria and show it is **NP**-hard; (b) we propose a heuristic that concurrently enumerates cliques on a social graph G, conditioned on forming a densest subgraphs on an *activity graph* G_A; and (c) we conduct an experimental study, in which we demonstrate the efficiency of our solution and its predictive power with regard to future groups of companions in real-world data, versus adaptations of previous works to this task.

2 Related Work

In this section, we review work on related problems. In the conventional sense, *group recommendation* refers to the recommendation of items towards a given group of users [5]. On the other hand, *group discovery* refers to the extraction of user groups from collaborative rating data sets [6]. Our problem formulation is mostly related to *community detection* problems based on the extraction of, potentially overlapping, *cliques* from a graph.

2.1 Finding Cliques

Tomita et al. [7] studied the worst-case complexity of finding and enumerating maximal cliques on a graph. Such cliques can be used to define communities in several ways; for example, one may consider only those maximal cliques of size above a threshold, and define communities as the disconnected components of the graph formed by the union of those cliques [8]. Alternatively, one may use cliques of fixed size, k; the *clique percolation* method [9] finds all such k-cliques in a network, and then finds clusters made out pairs of k-cliques sharing $k-1$ nodes.

2.2 Dense Subgraph Discovery

The *densest subgraph* problem, asking for a vertex subset $S \subseteq V$ on a graph $G(V, E)$ such that its induced subgraph achieves the maximum average degree, is solvable in polynomial time via the solution to a maximum flow problem [10], while a greedy $\frac{1}{2}$-approximation scheme requires linear time [11–13]. Asahiro et al. [14] study the $k - f(k)$ dense subgraph problem, which calls for finding a k-vertex subgraph of a given graph G that has at least $f(k)$ edges, for different functions $f(k)$. When a restriction is imposed on the size of set S, the problem becomes **NP**-hard [13]. Recently, the problem has been studied in streaming and MapReduce setting [15].

2.3 Community Detection

There is a vast literature on community detection, which bears some resemblance to the problem we examine, yet does not attempt to extract information from two graphs and is not oriented towards *consecutive activities*. Here we present a brief survey of such works.

Brown et al. [16] examine the collocation behavior among groups in a social network, defined as appearance at the same locations, yet consider neither consecutive appearances that indicate group mobility, nor the social ties among groups. Taking a step further, some works [17, 18] present methods for recommending LBSN companions to a given user, based on observed activities. Yet, again, activities are considered as stand-alone activities, without considering consecutivity. Besides, such works aim to detect groups relevant to a *particular querying user*, not to discovering *all groups of interest*. In a similar vein, Purushotham [19] devise an approach that recommends groups to a particular LBSN user, based on a hierarchical Bayesian model for joint topic modeling of *stand-alone* (i.e., no consecutive) activity locations and group preferences.

A line of work a bit closer to ours is that of detecting communities in overlapping networks. Comar et al. [20] combine two networks to mine communities; yet the notion of combining in [20] is that of *merging* diverse networks so as to discover larger communities than one would otherwise. The case of networks sharing common nodes, as in our problem, is called *community detection in multi-layer graphs* [21]. A series of algorithms have been proposed in the multi-layer domain. Li et al. [22] study community discovery among large document corpora, considering both textual attributes and relations. Other works employ matrix factorization to fuse information from different graph sources [23–25]. Zhou et al. [26] propose a graph clustering method that takes into consideration both structural and attribute similarity among heterogeneous vertices. In another direction, Zeng et al. [27] mine the complete set of frequent coherent (i.e., dense enough) closed quasi-cliques from vertex-labeled graph transaction databases, where the notion of quasi-cliques is based on a *vertex degree bound*. Yet the focus in [27] is on mining dense transaction structures based on their *frequency*, rather than their density itself. Boden et al. [28] apply a similar notion on edge-labeled multi-layer graphs, where a *multi-layer coherent subgraph* contains vertices densely connected by edges with similar *labels* in a subset of the graph's layers; This technique applies the same density criterion on multiple layers; in contrast, we apply *different density criteria* on two graph layers: a clique constraint on the one vs. quasi-clique optimality on the other; this distinction is cardinal for the functionality of our method — as we show, *by relaxing the clique constraint, it loses its predictive power*. Last, Ruan et al. [29] propose methods to mine communities based on link strength and content similarity.

Symbol	Meaning
l	POI/ Location identifier
u	User identifier ($u \in V$)
cat	Category of a POI
t	time interval
$\mathcal{A}(u, cat)$	Activities by u at POIs of category cat
$\mathcal{L}$	Categories of interest
G	Social graph
G_A	Activity graph
$G_A^{\mathcal{L}}$	Activity graph on categories $\mathcal{L}$
w_{uv}	edge weight in $G_{\mathcal{L}}$
f_α	Edge surplus function

Table E.1: Notations

3 Problem Formulation

Let V be a set of LBSN users and $\mathcal{U}$ a universe of categories (i.e., types, based on function and audience) to which locations of interest are associated. We start out by defining preliminary concepts that we use in our problem. Table E.1 lists the notations we employ.

3.1 Preliminary Concepts

Definition E.1. *A* ***point of interest*** *(POI) is a geographical location (e.g.,* the Metropolitan Museum of Art*) represented by a quadruple (l, lat, lon, cat), where l is the identifier, lat and lon are the latitude and longitude of the GPS coordinates of the center of the POI, and $cat \in \mathcal{U}$ is a category that this location belongs to.*

Definition E.2. *An* ***activity*** *refers to a visit or check-in of a user $u \in V$ at a location l at a large* discretized *time interval t, represented as a triplet (u, l, t); when the user u is clear from the context, we may represent an activity by the pair (l, t).*

Definition E.3. *An* **activity set** *for a user u, $\mathcal{A}(u)$, denotes a set of activities user u has engaged in; likewise, an activity set for a category cat, $\mathcal{A}(cat)$, denotes the set of all recorded activities associated with category cat among all users in V; last, $\mathcal{A}(u, cat)$ denotes the set of all activities by user u over POIs of category cat. We overload each of these notation to also denote ordered* sequences *of activities depending on the context.*

Definition E.4. *A* **spatiotemporal join** *operation among sets of activities $\mathcal{S}$ and $\mathcal{T}$, $\mathcal{S} \bowtie \mathcal{T}$, returns the set of pairs of activities $\{(l,t) \in \mathcal{S}, (l',t') \in \mathcal{T}\}$, where $l = l'$ and $t = t'$ Furthermore, a* **consecutive spatiotemporal join** *is defined among two temporal* sequences *of activities $\mathcal{S}$ and $\mathcal{T}$, $\mathcal{S} \bowtie^c \mathcal{T}$, and returns the set of* quadruples *of activities $(l,t), (l_{suc}, t_{suc}) \in \mathcal{S}, (l',t'), (l'_{suc}, t'_{suc}) \in \mathcal{T}\}$, where (l_{suc}, t_{suc}) is the successor activity of (l,t) in sequence $\mathcal{S}$ and (l'_{suc}, t'_{suc}) the successor activity of (l',t') in sequence $\mathcal{T}$, such that $(l,t) = (l',t')$* and *$(l_{suc}, t_{suc}) = (l'_{suc}, t'_{suc})$.*

From our last definition it follows that the spatiotemporal join between $\mathcal{S}$ and $\mathcal{T}$ returns all pairs of activities that have occurred in both $\mathcal{S}$ and $\mathcal{T}$, according to the coarse discretization we employ. Further, a consecutive spatiotemporal join returns all *quadruples* of activities that have occurred, as two consecutive pairs, in both sequences $\mathcal{S}$ and $\mathcal{T}$. These concepts are going to be useful in the following, when we define weights in the activity graph G_A.

3.2 Objective

Given a data set of users, their relationships, and their past records of activities, and a query set of categories of interest, $\mathcal{L} \subset \mathcal{U}$, we are interested to identify any group of users $\mathcal{C} \subset V$ that are likely to participate, as a group, in future activities related to $\mathcal{L}$.

We leverage two sources of information: (i) a **social graph** $G(V, E)$, where V is the set of users and E the set of friendship relationships among them; and (ii) an additional **activity graph** $G_A^{\mathcal{L}}(V, E')$, coterminous with (i.e., defined over the same set of vertices V as) G, built out of the historical log records of users' past user activities on a set of categories $\mathcal{L}$ as follows: an edge $(u, v) \in E'$ between any pair of users $u, v \in V$, acquires a non-zero weight, $w_{uv} \in (0, 1]$, representing the extent to which these two users are recorded to participate in activities associated with $\mathcal{L}$, together, according to the following definition.

Definition E.5. *The* **edge weight** *w_{uv} between the pair of users (u, v) in $G_A^{\mathcal{L}}$ is defined via the (consecutive) spatiotemporal join $\bigcup_{cat \in \mathcal{L}} \{\mathcal{A}(u, cat) \bowtie \mathcal{A}(v, cat)\}$, normalized by dividing by the highest value obtained among all pairs of users, as follows:*

$$w_{uv} = \frac{|\bigcup_{cat \in \mathcal{L}} \{\mathcal{A}(u,cat) \ltimes^{(c)} \mathcal{A}(v,cat)\}|}{max_{x,y \in V} |\bigcup_{cat \in \mathcal{L}} \{\mathcal{A}(x,cat) \ltimes^{(c)} \mathcal{A}(y,cat)\}|} \tag{E.1}$$

We intend to discover groups, or cohorts, of users having *high edge weight density* in the activity graph, i.e. having a history of common activities among group members, while also forming a *clique* (i.e., being related to each other) in the social graph. Intuitively, such cohorts tend to keep acting together, just like cohort $\{a, b\}$ did in the example of Figure E.1, as opposed to group $\{a, c\}$; therefore, they may be used for recommendation, prediction, and social analysis. Formally, we define our problem as follows:

Problem E.1. [Cohort Discovery] *Given a set of LBSN users V, a set of categories $\mathcal{L}$, a social graph $G(V, E)$ among users in V, and a weighted activity graph $G_A^{\mathcal{L}}(V, E')$ among users in V, where edges in E' are weighted according to activities in $\mathcal{L}$, find a set $\mathcal{C}$ of k* cohorts *of users, where for each $C \in \mathcal{C}$ it holds that $C \subseteq V$, such that: (i) the subgraphs these cohorts induce in the social graph G are* cliques, *and (ii) the subgraphs the same cohorts induce in the activity graph $G_A^{\mathcal{L}}$ obtain, among all activity graph subgraphs that form cliques in the social graph G, the top-k highest values in terms of an* ***activity density*** *(or* weighted edge density*) function $f^{\mathcal{L}}(\mathcal{C})$.*

3.3 Maximizing Activity Density

A question that arises at this point is, what function of weighted edge density should we aim to maximize as activity density? To answer this question, the concept of an Optimal Quasi-Clique (OQC), introduced by Tsourakakis et al. [30], comes handy. The OQC problem is defined as follows [30]:

Problem E.2. [OQC] *Given a graph $G = (V, E)$ and $\alpha \in (0, 1)$, find a subset of vertices $S^* \subseteq V$ such that*

$$f_\alpha(S^*) = e[S^*] - \alpha \binom{|S^*|}{2} \geq f_\alpha(S), \text{ for all } S \subseteq V. \tag{E.2}$$

where $e[S]$ is the number of edges in the subgraph of G induced by S. The set S^* is called an ***optimal quasi-clique*** *of G, while function f_α is called the* ***edge surplus****.*

This OQC definition captures edge density without favoring large subgraphs: a subgraph achieves a high value of edge surplus f_α not merely by means of a high average degree, as large subgraphs may have, but by coming close to completing a *clique* among its nodes. This measure is appropriate for our purposes: it offers a size-independent density measure as we need, and can also be straightforwardly generalized to the weighted edges in an activity graph. Thus, we define our activity density function as follows:

Definition E.6. *Given an activity graph $G_A^{\mathcal{L}} = (V, E')$ and a cohort (subset) of vertices $C \subseteq V$, and a parameter $\alpha \in (0,1)$, the* ***activity density*** *over C is defined as:*

$$f_\alpha^{\mathcal{L}}(C) = w[C] - \alpha \binom{|C|}{2} \tag{E.3}$$

where $w[C]$ is the sum of normalized edge weights for all edges in the subgraph induced by C: $w[\mathcal{C}] = \sum_{u,v \in \mathcal{C}} w_{uv}$.

We aim to discover cohorts under the constraint of forming a clique in a social graph and the optimization objective of maximizing an edge surplus function in the activity graph. To address this problem, it is useful to investigate the hardness of the simple OQC problem itself. After all, in case our social graph is a complete graph, and all non-zero edge weights in the activity graph are equal to 1, then Cohort Discovery is effectively reduced to the OQC problem, hence it is at least as hard as OQC. Tsourakakis et al. [30] suspect OQC to be **NP**-hard, yet provide no formal proof of hardness. We provide such a proof in the following. To arrive there, we start out with some lemmata regarding the nature of the OQC problem.

Lemma E.1. *For any $\alpha \in (0,1)$, any clique in graph $G = (V, E)$ has a positive edge surplus.*

Proof. By definition, the number of edges in a clique $S \subseteq V$ is $e[S] = \binom{|S|}{2}$. Then, the edge surplus of S is $f_\alpha(S) = e[S] - \alpha\binom{|S|}{2} = (1-\alpha)\binom{|S|}{2} > 0$. $\square$

Lemma E.2. *For any $\alpha \in (0,1)$, a maximum clique of a graph $G = (V, E)$ has the maximum edge surplus among all cliques in G.*

Proof. By Lemma E.1, the edge surplus of a clique $S \subseteq V$ is $f_\alpha(S) = (1 - \alpha)\binom{|S|}{2} > 0$. A *maximum clique* achieves the maximum number of vertices $|S|$ among all cliques in G; therefore, it also has the maximum edge surplus. $\square$

Now we can prove the following theorem.

Theorem E.1. *Given a simple undirected graph $G = (V, E)$, for $\alpha = 1 - \binom{|V|}{2}^{-1}$, a subset of vertices $S \subseteq V$ has positive edge surplus* if and only if *it is a clique.*

Proof. The $\Leftarrow$ direction is provided by Lemma E.1. We now prove the $\Rightarrow$ direction: The edge surplus of a subset of vertices $S \subseteq V$ is $f_\alpha(S) = e[S] - \alpha\binom{|S|}{2} = e[S] - (1 - \binom{n}{2}^{-1})\binom{|S|}{2} = e[S] - \binom{|S|}{2} + \frac{|S|(|S|-1)}{|V|(|V|-1)}$. If S has positive edge surplus, then $f_\alpha(S) > 0 \Leftrightarrow e[S] > \binom{|S|}{2} - \frac{|S|(|S|-1)}{|V|(|V|-1)}$. Since $S \subseteq V$, it follows that $\frac{|S|(|S|-1)}{|V|(|V|-1)} \leq 1$. Thus, $f_\alpha(S) > 0 \Rightarrow e[S] > \binom{|S|}{2} - 1$. Yet the only subgraph of $|S|$ vertices that has more than $\binom{|S|}{2} - 1$ edges is a clique. $\square$

We can now prove the following hardness result.

Theorem E.2. OQC *is* **NP**-*hard.*

Proof. We construct our proof by reduction from the **NP**-hard CLIQUE problem, which calls for finding a maximum clique in a simple undirected graph. Assume we are given a polynomial-time algorithm $\mathcal{A}(G, \alpha)$ that can solve the OQC problem on any simple undirected graph $G(V, E)$ for any parameter $\alpha \in (0, 1)$. Then, given any instance of the CLIQUE problem on a simple undirected graph $G(V, E)$, we invoke $\mathcal{A}(G, \alpha)$ with $\alpha = 1 - \binom{|V|}{2}^{-1}$. We emphasize that an elaborate *reduction* is not necessary, as we use the *same graph* in both problems. By Theorem E.1, if the returned optimal quasi-clique (OQC) has non-positive edge surplus, it follows that G has no cliques; otherwise, if the returned OQC has any positive edge surplus, it is a clique. Moreover, by definition, the returned OQC has the maximum edge surplus among all such cliques in G, hence, by Lemma E.2, it *is a maximum clique* of G. In effect, an algorithm that solves the OQC problem in polynomial time would also solve any instance of CLIQUE. Then, by reduction from CLIQUE, it follows that OQC is at least as hard as any problem in **NP**. □

Our proof resolves a question left open in [30]. Given this result, we should strive for non-optimal solutions to COHORT DISCOVERY, as this problem is at least as hard as the OQC problem.

4 COVER Algorithm

In this section we present COVER, our algorithm for the cohort discovery problem; COVER merges and builds upon techniques for maximal clique enumeration [7] and the OQC problem [30]. In a nutshell, it searches for cliques on the social graph, yet, instead of striving to satisfy just a maximality condition upon them, it checks, for any possible clique candidate, the edge surplus of its induced subgraphs on the activity graph, pruning from consideration nodes that cannot lead to higher edge surplus than already discovered. Eventually, it outputs the top-k results by our problem definition.

A cohort should form a social clique and also achieve as high activity density as possible, as outlined in Section 3.3. COVER explores the social graph in order to find cliques, as an algorithm for maximal clique enumeration would do. Yet, for each clique it finds in the social graph G, it searches, in local search fashion, for its subgraphs of high activity density in the activity graph G_A, maintaining a queue of the top-k results, and, thereby, also a global queue of the top-k cohorts overall. In this process, when considering a new node v, it evaluates its strength, in terms of marginal activity density,

Algorithm E.1: COVER: finding top-k cohorts

```
1  Input: $G(V,E)$, $G_{\mathcal{L}}(V,E)$, $k$, $T_{Max}$, $\alpha$
2  Output: Set of $k$ traveler groups : $\mathcal{CO}$
3  begin
4      $C = \emptyset$ /* a clique in $G$ */
5      $cohG = \emptyset$ /* queue of top-$k$ cohorts within $C$ */
6      $\mathcal{CO} = \emptyset$ /* global queue of top-$k$ cohorts */
7      searchCliques($V, V, C$) /* recursive function */
8  end
```

that it may bring to any cohort under construction; node v is further considered only if its marginal activity density can bring an advantage compared to the top-k cohorts discovered so far.

Algorithm E.1 is the main shell of COVER; it initializes global variables and priority queues and finds top-k activity-based cohorts recursively on the back of social cliques.

Algorithm E.2: searchCliques($Sub, Cand, C$)

```
1  begin
2      /* Sub: seed set to be searched for cliques */
3      /* Cand: expansion set for building cliques */
4      /* $N_G(u)$: friends of $u$ in $G$*/
5      if $Sub \neq \emptyset$ then
6          $u \leftarrow$ vertex $\in Sub$ maximizing $|Cand \cap N_G(u)|$
7          foreach $v \in Cand \setminus N_G(u)$ do
8              $Sub_v \leftarrow Sub \cap N_G(v)$
9              $Cand_v \leftarrow Cand \cap N_G(v)$
10             $cohG_M \leftarrow Cand_v \cup C \cup \{v\}$
11             /* $cohG_M$: largest possible clique on $v$ */
12             if $\binom{|cohG_M|}{2}(1-\alpha) > min_{S \in \mathcal{CO}}\,(f_\alpha(S))$ then
13                 searchCliques($Sub_v, Cand_v, C \cup \{v\}$)
14             end
15             $Cand \leftarrow Cand \setminus \{v\}$
16         end
17     end
18     else
19         $cohG \leftarrow$ findCohorts($G_A, C, k$)
20         $\mathcal{CO} \leftarrow \mathcal{CO} \cup cohG$
21     end
22 end
```

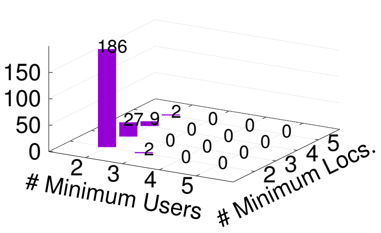
Algorithm E.2 searches for promising cliques C in the social graph G. We start with the set of all users V, and recursively explore subgraphs having a clique property in depth-first-search manner. We maintain two set variables: *Sub* maintains the intersection of the neighbor sets of all nodes already entered in the clique C currently under construction. On the other hand, *Cand* maintains the intersection of such neighbor sets *minus* any nodes that have already been checked, i.e., included, or considered for inclusion, in C. We use *Cand* to generate new candidates for checking at each iteration (Line 7). Besides, to accelerate the search, at each iteration we pick up a high-degree *pivot* node $u \in Sub$ having many neighbors $N_G(u)$ in $Cand$ (Line 6), and exclude those neighbors from the search (Line 7), as they will be considered later (or have already been considered) by virtue of being neighbors of u anyway. Then, we check candidate nodes $v \in Cand \setminus N_G(u)$ (Lines 7-14) for inclusion in C. A critical check is performed in Line 12: if the largest possible clique that can be built by including v and its neighbors can yield best-case activity density *among* the current top-k cohorts, then C is recursively expanded with v (Line 13); otherwise, we discard the depth-first search path leading to v is discarded, as it cannot bring forth new results, and *thereby we avoid redundant computations*. Last, when clique C cannot be expanded further, we search for the top-k cohorts therein by calling Algorithm E.3 and update the global priority queue $\mathcal{CO}$ accordingly (Lines 16-17).

Algorithm E.3 finds top-k cohorts within activity graph G_A and social clique C by local search. A candidate cohort S starts out as the node $u \in C$ of highest ratio of adjacent triangles to degree and its neighbors (Line 3). Then, we iteratively revise S, first by adding nodes, as long as that can bring a benefit in activity density, choosing the best such option (Lines 8-12); then by removing the best node whose removal can bring benefit (Lines 13-16). The process is repeated until we reach a local optimum or the maximum allowed iterations T_{max}. At each intermediate step, we insert the running S to the priority queue $cohG$, and eventually merge the result in the global priority queue $\mathcal{CO}$.

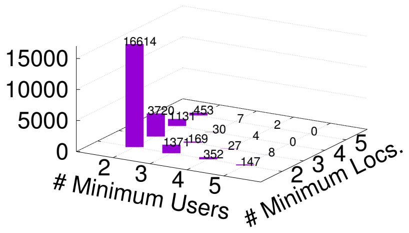
Algorithm E.3: findCohorts(G_A, C, k)

```
1  begin
2      $u$ : vertex with max $\frac{\#\text{triangles}}{\text{degree}}$ ratio in $G_A[C]$
3      $S \leftarrow N(u) \cup \{u\}$ /* $u$ and $G_A$ neighbors */
4      $cohG \leftarrow \{S\}$
5      $b_1 \leftarrow$ True, $t \leftarrow 1$ /* local search begins */
6      while $b_1$ and $t \leq T_{max}$ do
7          $b_2 \leftarrow$ True
8          while $b_2$ do
9              if $\exists v \in C \setminus S$ such that $f_\alpha(S \cup \{v\}) \geq f_\alpha(S)$ then
10                 $S \leftarrow S \cup \{v\}$; $cohG \leftarrow cohG \cup S$
11             end
12             else
13                 $b_2 \leftarrow$ False /* growth of $S$ stops */
14             end
15         end
16         if $\exists x \in S$ such that $f_\alpha(S \setminus \{x\}) \geq f_\alpha(S)$ then
17             $S \leftarrow S \setminus \{x\}$; $cohG \leftarrow cohG \cup S$
18         end
19         else
20             $b_1 \leftarrow$ False /* local search stops */
21         end
22         $t \leftarrow t + 1$ /* iteration counter */
23     end
24 end
```

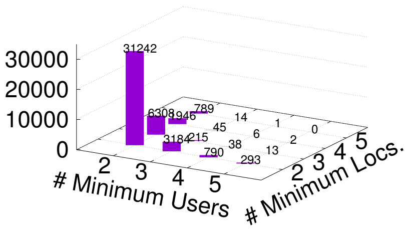
Given the analysis of the worst-case complexity of clique enumeration in [7], the worst-case complexity of the COVER algorithm is $O(3^{\frac{n}{3}} + cT_{max}m)$, where n is the number of nodes, c the number of enumerated cliques that reach Line 12 in Algorithm E.2, m the number of activity graph edges, and T_{max} the number of iterations in Algorithm E.3, which touches at most once per iteration [30]. In practice, we avoid this worst-case scenario by a *massive* discarding of paths in Line 12 of Algorithm E.2. Therefore, as we will see, the algorithm is efficient in practice.



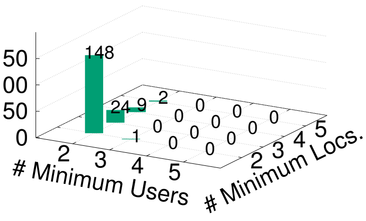
(a) All Cohorts (FS)



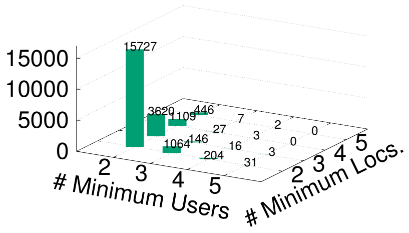
(b) All Cohorts (GW)



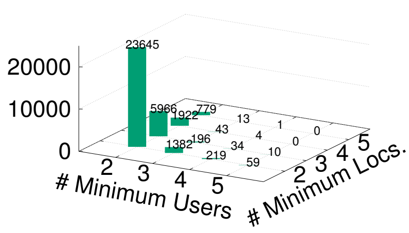
(c) All Cohorts (Wee)



(d) Clique Cohorts (FS)



(e) Clique Cohorts (GW)



(f) Clique Cohorts (Wee)

Fig. E.3: Statistics on cohorts: all users vs. cliques

5 Experimental Study

In this section, we present the result of an extensive experimental evaluation of COVER, including its ability to predict future group of companions, *regardless of whether they form a social clique*.

	Users	Locations	Checkins	POIs	Friend pairs	Duration	Categories
FourSquare	4K	0.2M	0.47M	0.12M	32K	1322 days	35
Gowalla	77K	2.8M	18M	2M	4M	913 days	363
Wee	16K	0.9M	8M	0.76M	0.1M	2796 days	770

Table E.2: Dataset characteristics

5.1 Datasets

We utilize three real-world datasets [31] taken from Foursquare, Gowalla, and Weeplaces. Table E.2 gathers information about the data. Each of the datasets consisted of three parts: the social friendship graph, an ordered list of check-ins, and a collection of Venues. A check-in record contains the user-id, check-in time, GPS coordinates, and a location-id. Venues provide the details of locations, i.e., city, country, and semantic categories of those locations.

Data Prepossessing. The data required cleaning, as many locations were associated with multiple location identifiers, each having slightly different GPS coordinates. We clustered GPS points to get POIs. We used a grid based spatial clustering with a grid of size 10 meters $\times$ 10 meters, so as to group GPS points. Then, we assigned a new, unique location Id to each resulting cluster; these Ids are then used in all our experiments. All three datasets presented similar multiplicity problems, which we addressed in the same manner. Statistics regarding these new POI Ids are reported in Table E.2.

5.2 Brute-Force Cohort Discovery

We first present a baseline that mines groups of users having similar mobility behavior by brute force.

In order to find groups of common mobility, we first divide the activities of each user into a series of time intervals, or snapshots, using a time-stamp threshold t_S, recording *one* activity per interval: the last recorded activity. We can tune t_s so as to strike a fine balance in the tradeoff between time granularity and computation time. With larger t_s values, it becomes likelier to miss activities within a time snapshot. In order to set a suitable t_s value, we plotted all time differences between consecutive user activities. Figure E.4

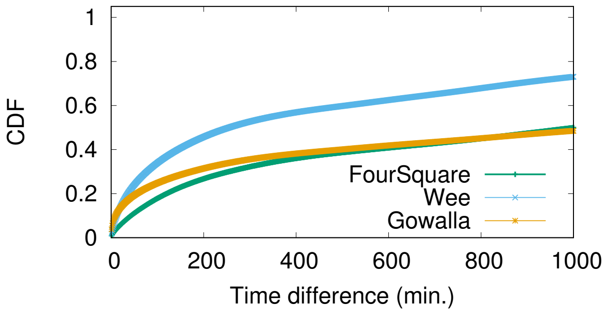


Fig. E.4: CDF of time between consecutive activities

shows their cumulative distribution. Based on this plot, we set $t_s = 1$ hour, which covers a sufficiently large ratio of activities.

We mine cohorts from recorded activities. A cohort should: (i) contain a *at least two users* moving together across locations; and (ii) involve a *at least two locations* consecutively visited by those users during its lifetime. We maintain an ongoing cohorts list L and iterate over the data snapshots in increasing temporal order. For each snapshot, we group activities by location and check each group against the ongoing cohorts list. If a group extends an existing cohort C, we update C accordingly. Otherwise, if a group first forms a cohort in this snapshot onwards, we insert it in L. We store away any items in L that are no longer expandable and already form a cohort. This process goes on until the last snapshot. We emphasize that this brute-force approach is more computationally demanding than COVER's activity density estimation: it detects and expands all groups in the training set, while COVER only considers pairs of users and at most pairs of consecutive common activities; in other words, COVER uses data about *only* the smallest possible value, 2, for *number of users* and *number of locations* consecutively visited by those users, whereas the brute-force method explicitly mines cohorts for all values. Figure E.3 shows statistics on cohorts so discovered vs. those whose members *also* form social cliques. About 76% of all cohorts form *social cliques*. This finding validates our conjecture that people are likely to *move along with friends in social cliques*, hence vindicates the clique constraint in COVER.

5.3 Holdout Approach

We configure an experiment to assess the predictive power of COVER on groups that form mobility companions in the future. That is, we *do not test* its ability to find groups forming a social cliques in the future. We just surmise that the clique constraint in our definition allows to predict future common mobility.

We use a *holdout approach*: first, we sort activities by ascending timestamp; then, we divide them into two parts, the training (earlier) and test (later) data, such that both have an equal number of activities. If a group found in the training data (*or its superset*) exists as common companions in the test data, we count it as a success. Under this approach, we mine cohorts in two ways.

Without Input Categories: This way we simply find groups that are most likely act as cohorts in the future. We first find cohorts in the naive way and then we count their appearances. The cohort with the highest count is the best candidate.

With Input Categories: In this case, we are given not only a log of check-in data, but also a set of categories of locations of interest $\mathcal{L}$. We can then find groups that are most likely to act in cohort fashion in the future, moving among locations of the given categories. In order to identify such groups, we filter the dataset such that only activities at locations of $\mathcal{L}$ are maintained. Then, we fetch cohorts as explained above; again the group that has formed most cohorts in the logs is the best candidate.

As there is no previous work the problem we study, we evaluate COVER on its ability to predict the results of naive cohort discovery. We measure this ability by an accuracy metric, defined as:

$$Acc = \frac{|C \cup T|}{\min\{|C|, |T|\}} \tag{E.4}$$

where C is the set of top-k cohorts returned by COVER and T is the set of cohorts in the test data. We sanitize the measure with division by the minimum of $|C|, |T|$ so that its values are in $[0, 1]$.

5.4 Revalidating the Clique Constraint

Before we proceed, it is worth to re-validate our social clique conjecture. To do so, we test COVER *without considering the social graph*. We extract top-k groups in terms of activity edge surplus in the training data, for $k = 1, 3, 5$. This way, we only get a positive prediction with the Wee dataset for $k = 3$, which is five times less accurate than what we achieved using the clique constraint. This result reconfirms the necessity of that constraint: users who participate in common activities but do not form a social clique are not likely to do so again. Further, we relaxed the clique constraint to find *quasi-cliques* in the social graph for several α values. Unfortunately, this way we could not predict *any* group of travel companions. We emphasize that the groups we aim to predict are *not required* to form social cliques. We simply observe empirically that groups of future travel companions *form* social cliques, and *cannot be predicted without this constraint*.

5.5 Prediction without Input Categories

We first present prediction results for the case *without* a restrictive set of given categories of interest $\mathcal{L}$.

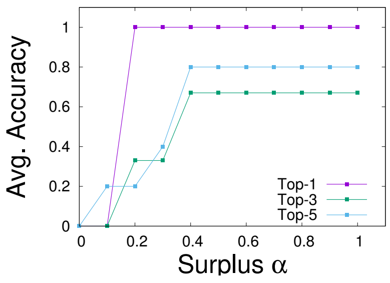
Brute-force cohort discovery: When no categories of interest are specified in the problem input, we can still run the naive cohort discovery algorithm on the training dataset and test its predictions on the test dataset. Then, the naive method returns the k most frequently observed cohorts. If a predicted cohort exists as such in the test dataset, we score a success; otherwise, a failure. The last three columns of Table E.3 show the top-k cohort prediction accuracy results for $k = 1, 3, 5$. Accuracy reaches 100% in all but two cases.

	Input Cat			**All Cat**		
k	**FS**	**GW**	**Wee**	**FS**	**GW**	**Wee**
1	0.60	0	0.50	1	1	1
3	0.66	0.17	0.60	1	1	1
5	0.64	0.20	0.70	0.30	0.80	1

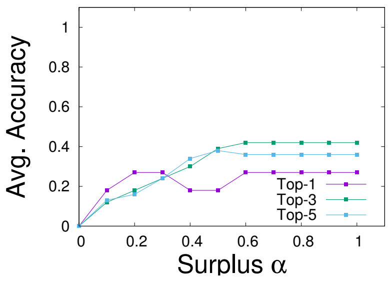
Table E.3: Accuracy for brute-force cohort prediction

COVER on plain activity density: Then, we tested COVER on the same cohort prediction problem. First, we try the case in which we do not take consecutive activities into consideration when forming edge weights in the activity graph. We present results on the accuracy of top-k cohort set predictions as a function of the chosen density surplus α in Figures E.5a, E.5e, and E.5i. We observe that the top cohort prediction becomes correct at the value of 0.1 and remains true for all the values of α. On the other hand, for top-3 and top-5 returned cohorts, the accuracy starts out lower. This result is due to variations among training and test data; similar divergencies were observed in Table E.3 even with the brute-force method. The result that COVER can achieve almost equally well is remarkable, as it is up to 3 orders of magnitude faster. In most cases, accuracy drops as the value of k increases, as it is easier to get a top cohort correctly than the whole group of top 3 or 5.

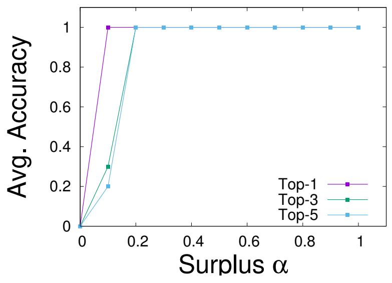
COVER on consecutive activity density: Next, we examine the case in which the activity density is defined by a consecutive spatiotemporal join, considering the extent to which users *move* along each other. As Figures E.5c, E.5g, and E.5k show, now the maximum accuracy is achieved with smaller α. This result is intuitive, as the purpose of α is to force the detection of small and tightly connected groups. Yet, the consecutivity requirements creates smaller cohorts on its own, rendering the impact of α less significant. Overall, COVER's performance is remarkably high.



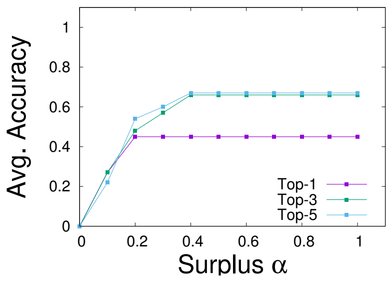
(a) Plain Density, All, Wee



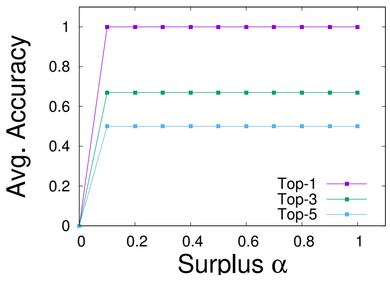
(b) Plain Density, Input, Wee



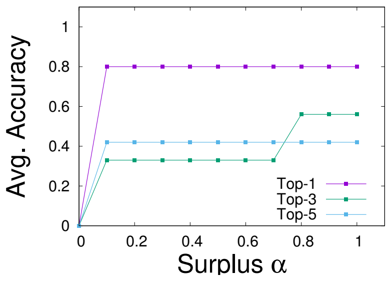
(c) Consecutive Density, All, Wee



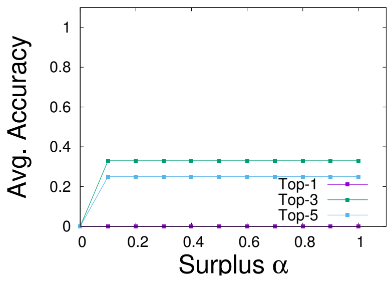
(d) Consecutive Density, Input, Wee



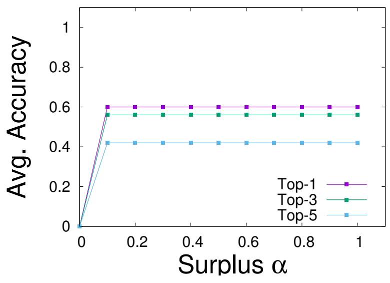
(e) Plain Density, All, FS



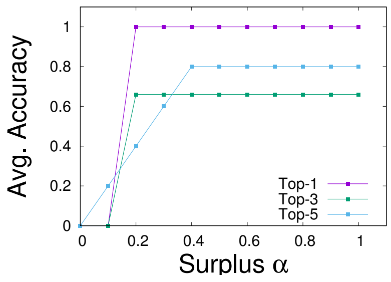
(f) Plain Density, Input, FS



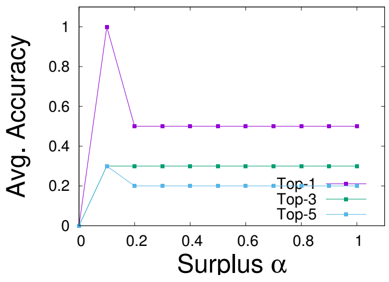
(g) Consecutive Density, All, FS



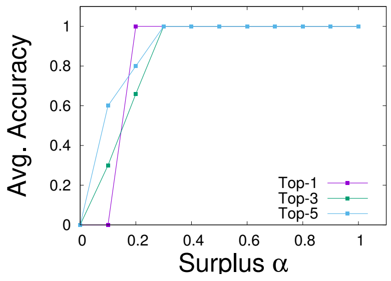
(h) Consecutive Density, Input, FS



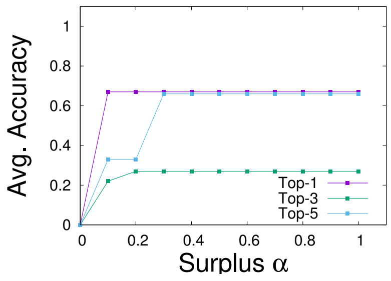
(i) Plain Density, All, GW



(j) Plain Density, Input, GW



(k) Consecutive Density, All, GW



(l) Consecutive Density, Input, GW

Fig. E.5: Effect of different factors on cohort discovery with three datasets

5.6 Prediction with Input Categories

Now we turn our attention to the case *with* a restrictive set of given categories of interest $\mathcal{L}$. We devise 10 categories sets, which appear together in a real-world context, and we average accuracy over all ensuing queries. For COVER, edge weights in the activity graph $G_A^{\mathcal{L}}$ are based solely on locations in $\mathcal{L}$.

Brute-force cohort discovery: The brute-force method achieves accuracy up to 66% (Table E.3). The lower accuracy values are due to the fact that cohorts specific to the input categories may not be met in the training data at all, but appear in the test data. Likewise, cohorts characterized by the given categories in the training data may not reappear in the test data, due to data sparsity.

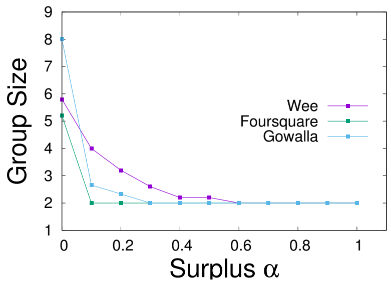
COVER on plain activity density: The average top-k set prediction accuracy results for COVER based on plain activity density appear in Figures E.5b, E.5f, and E.5j. While the problem is quite more challenging due to data sparsity, we still obtain high accuracy in most cases. Less accuracy variation with varying α appears with the Foursquare data, the smallest of our datasets, as fewer cohorts arise in it. Accuracy still drops with increasing

k, except for the case of the Wee data, where including more cohorts in the result set raises the chances that they will be encountered in the test data.

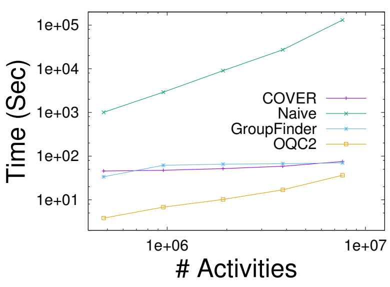
COVER on consecutive activity density: Last, Figures E.5d, E.5h, and E.5l show the results for COVER using consecutive activity density. Here, we both choose input categories and also take consecutivity into consideration. Remarkably, here accuracy is either similar to or *significantly higher* than that in the non-consecutive case reaching 67% for the top-5 cohorts set. This result vindicates the use of consecutive activity density. In the cases of Foursquare and Gowalla, by virtue of smaller returned cohort sizes, accuracy increases sharply with growing α, then stabilizes above 60%.

5.7 Group size vs. surplus α

Prediction accuracy grows with the surplus parameter α. Yet, the size of returned cohorts is bound to decrease as α grows, as higher values demand stronger cohesiveness. On the other hand, for several applications, one would prefer to detect sizeable cohorts. Hence, a tradeoff between size and cohesiveness arises. To study this tradeoff, we measure the average returned cohort size vs. α on the top-5 cohorts with input categories and consecutive activity density. The results in Figure E.6a show that average cohort size decreases with increasing surplus up to 0.6; thereafter, cohort size becomes 2, the minimum allowed. A *sweet* value of α is around 0.4, yielding both high cohort size and predictive power. We employ this value in the comparative experiments of Section 5.9.



(a) Average group size vs. α



(b) Runtime vs. data size (Wee)

Fig. E.6: Effect of α and data size

5.8 Scalability

Here, we assess all methods in terms of scalability, measuring runtime on 6.25%, 12.5% 25%, 50% and 100% on the Wee data set. Figure E.6b shows the results for COVER, Brute-Force (called Naive) and GroupFinder (same for GF-PAV and GF-PLM), and OQC ($\alpha = 0.6$). The runtime of COVER is comparable to that of GroupFinder, while that of Brute-Force (Naive) is much more demanding and does not scale well. Trends with other data were similar.

5.9 Comparative Evaluation

We now conduct a comparative evaluation of COVER with fixed $\alpha = 0.4$ against other approaches on the Wee data set; we juxtapose the COVER results to those of the following methods:

- **BF:** The Brute-Force cohort mining method of Section 5.2.
- **GroupFinder:** A method that finds groups of a given size k in LBSNs for a given user u and a set categories [18]. As this method does not solve the same problem as we do, we adapt it for the sake of conducting a reasonable comparison: Given a set of categories, we apply GroupFinder to each user in the data set and then choose the best group of size k, where k is the most popular group size returned by COVER (in all cases, 2). We utilize both of the pairwise user-item relevance measures proposed in [18]: pairwise aggregated voting (PAV) and pairwise least misery (PLM).
- **OQC**: A variant of COVER that relaxes the clique constraint on the social network and finds groups that achieve high edge surplus on both the social and the activity graph; this variant employs two α values, one for each edge surplus component; to avoid any bias against this variant, we try out α values with step 0.1 in the edge surplus formula, as well as different weights for the two surplus components in $\{0, 0.25, 0.66, 1, 1.5, 4, \infty\}$, and present the best results.

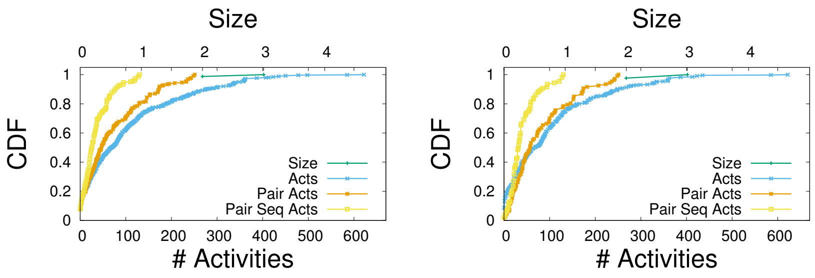
We evaluate the competitors on the following measures:

- *Accuracy*, the metric we have used in previous results;
- *Precision@K*, the ratio of the number of true top-k cohorts *by frequency* in the test data that we fetch in the top-k cohorts a method returns, over k, or over the total number of test data cohorts, if that is less than k;
- *Mean Average Precision* (MAP), the mean of Precision@K for all k values up to the examined one; and
- *Normalized Discounted Cumulative Gain* (NDCG), a popular information retrieval measure; we apply it on the *ranking* of top-k results by activity density with respect to their ranking by frequency in the test data, computed for cohorts existing in both rankings.

	Accuracy					P@K					MAP					NDCG				
K	**BF**	**GF PAV**	**GF PLM**	**OQC**	**COVER**	**BF**	**GF-PAV**	**GF-PLM**	**OQC**	**COVER**	**BF**	**GF-PAV**	**GF-PLM**	**OQC**	**COVER**	**BF**	**GF-PAV**	**GF-PLM**	**OQC**	**COVER**
1	0.5	0.01	0.01	0.25	**0.6**	0	0	0	0	0	0	0	0	0	0	**0.5**	0.09	0.1	0.25	0.35
5	**0.7**	0.03	0.03	0.1	0.4	**0.35**	0	0	0	0.15	**0.18**	0	0	0	0.17	0.7	0.25	0.1	0.5	**0.76**
10	0.6	0.03	0.03	0.05	**0.63**	**0.22**	0	0	0	0.15	**0.22**	0	0	0	0.16	0.79	0.25	0.15	0.5	**0.81**
15	0.5	0.08	0.05	0.03	**0.57**	**0.18**	0	0	0	0.15	**0.22**	0	0	0	0.16	0.81	0.1	0.25	0.5	**0.81**
20	0.47	0.08	0.05	0.025	**0.53**	0.14	0	0	0.01	**0.14**	**0.2**	0	0	0	0.16	0.8	0.25	0.25	0.5	**0.82**

Table E.4: Comparative Evaluation on Wee ($\alpha = 0.4$ for COVER, best values for OQC)

Table E.4 shows our results on consecutive activity density, with input categories, for five values of k. Brute-Force sometimes outperforms COVER, as it finds all exact cohorts in the training dataset, and these appear in the test data too; yet it is remarkable that COVER stands its ground vs. Brute-Force in several measures, even while avoiding such an exhaustive calculation. On the other hand, the groups found by GF-PAV, GFPLM and OQC do not perform well by any measure; none of them appears in the top-20, resulting in 0 value for P@K and MAP. GroupFinder performs poorly because it does not consider group and sequential activities; it only relies on users' mutual interests. This result indicates that users tend to visit different types of locations when they are in certain groups than when they are in other groups or alone. On the other hand, OQC performs poorly as it abolishes the social clique constraint, thus does not detect strongly connected groups. This finding further *corroborates the need* for the clique constraint.



(a) All Predicted Cohorts **(b)** Correctly Predicted Cohorts

Fig. E.7: Analysis on Predicted Cohorts

5.10 Analysis on Predicted Cohorts

Figure E.7 presents the CDF of cohort characteristics for all predicted cohorts (E.7a) and correctly predicted ones (E.7b), namely their size and number of activities performed by their constituent parts (i.e., individuals and pairs). The size of predicted cohorts goes up to 3, while we predict cohorts correctly even in cases where they have not performed any activities together on input categories. This capacity of COVER sets it apart from the brute-force baseline that can only predict cohorts who act together in the training set.

6 Conclusion

We proposed the problem of predicting future companions in LBSNs, and an efficient, nontrivial solution, COVER; this solution mines geo-social cohorts that satisfy (i) a *pragmatically necessary* clique constraint on a social graph and (ii) a density objective on a coterminous graph (i.e., defined on the same set of nodes) capturing common (and *consecutive*) pairwise past activities. Our experimental study with real-life data sets showed that COVER efficiently predicts cohorts of users that do move along each other in the future, including cohorts that do not appear in the training set, while adaptations of previous work cannot deliver the same results.

References

[1] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "Recommendations in location-based social networks: a survey," *GeoInformatica*, vol. 19, no. 3, pp. 525–565, 2015.

[2] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo, "Mining user mobility features for next place prediction in location-based services," in *ICDM*, 2012, pp. 1038–1043.

[3] C.-C. Hung, C.-W. Chang, and W.-C. Peng, "Mining trajectory profiles for discovering user communities," in *LBSN*, 2009, pp. 1–8.

[4] X. Xiao, Y. Zheng, Q. Luo, and X. Xie, "Inferring social ties between users with human location history," *J. Ambient Intelligence and Humanized Computing*, vol. 5, no. 1, pp. 3–19, 2014.

[5] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu, "Group recommendation: Semantics and efficiency," *PVLDB*, vol. 2, no. 1, pp. 754–765, 2009.

[6] B. O. Tehrani, S. Amer-Yahia, P. Dutot, and D. Trystram, "Multi-objective group discovery on the social web," in *ECML PKDD*, 2016, pp. 296–312.

[7] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006.

[8] M. G. Everett and S. P. Borgatti, "Analyzing clique overlap," *Connections*, vol. 21, no. 1, pp. 49–61, 1998.

[9] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[10] A. V. Goldberg, "Finding a maximum density subgraph," UC Berkeley, Tech. Rep., 1984.

[11] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama, “Greedily finding a dense subgraph,” *J. Algorithms*, vol. 34, no. 2, pp. 203–221, 2000.

[12] M. Charikar, “Greedy approximation algorithms for finding dense components in a graph,” in *APPROX*, 2000, pp. 84–95.

[13] S. Khuller and B. Saha, “On finding dense subgraphs,” in *ICALP*, 2009, pp. 597–608.

[14] Y. Asahiro, R. Hassin, and K. Iwama, “Complexity of finding dense subgraphs,” *Discrete Applied Mathematics*, vol. 121, no. 1-3, pp. 15–26, 2002.

[15] B. Bahmani, R. Kumar, and S. Vassilvitskii, “Densest subgraph in streaming and mapreduce,” *PVLDB*, vol. 5, no. 5, pp. 454–465, 2012.

[16] C. Brown, N. Lathia, C. Mascolo, A. Noulas, and V. Blondel, “Group colocation behavior in technological social networks,” *PLOS ONE*, vol. 9, no. 8, pp. 1–9, 08 2014.

[17] Y. Liao, W. Lam, S. Jameel, S. Schockaert, and X. Xie, “Who wants to join me?: Companion recommendation in location based social networks,” in *ICTIR*, 2016, pp. 271–280.

[18] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso, “Group finder: An item-driven group formation framework,” in *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, vol. 1, 2016, pp. 8–17.

[19] S. Purushotham, C.-C. J. Kuo, J. Shahabdeen, and L. Nachman, “Collaborative group-activity recommendation in location-based social networks,” in *GeoCrowd*, 2014, pp. 8–15.

[20] P. M. Comar, P. Tan, and A. K. Jain, “A framework for joint community detection across multiple related networks,” *Neurocomputing*, vol. 76, no. 1, pp. 93–104, 2012.

[21] J. Kim and J.-G. Lee, “Community detection in multi-layer graphs: A survey,” *SIGMOD Rec.*, vol. 44, no. 3, pp. 37–48, 2015.

[22] H. Li, Z. Nie, W.-C. Lee, L. Giles, and J.-R. Wen, “Scalable community discovery on textual data with relations,” in *CIKM*, 2008, pp. 1203–1212.

[23] G.-J. Qi, C. C. Aggarwal, and T. Huang, “Community detection with edge content in social media networks,” in *ICDE*, pp. 534–545.

[24] W. Tang, Z. Lu, and I. S. Dhillon, “Clustering with multiple graphs,” in *ICDM*, 2009, pp. 1016–1021.

[25] X. Dong, P. Frossard, P. Vandergheynst, and N. Nefedov, “Clustering with multi-layer graphs: A spectral perspective,” *Trans. Sig. Proc.*, vol. 60, no. 11, pp. 5820–5831, 2012.

[26] Y. Zhou, H. Cheng, and J. X. Yu, “Graph clustering based on structural/attribute similarities,” *PVLDB*, vol. 2, no. 1, pp. 718–729, 2009.

[27] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, “Coherent closed quasi-clique discovery from large dense graph databases,” in *KDD*, 2006, pp. 797–802.

[28] B. Boden, S. Günnemann, H. Hoffmann, and T. Seidl, “Mining coherent subgraphs in multi-layer graphs with edge labels,” in *KDD*, 2012, pp. 1258–1266.

[29] Y. Ruan, D. Fuhry, and S. Parthasarathy, “Efficient community detection in large networks using content and links,” in *WWW*, 2013, pp. 1089–1098.

[30] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli, “Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees,” in *KDD*, 2013, pp. 104–112.

[31] Y. Liu, W. Wei, A. Sun, and C. Miao, “Exploiting geographical neighborhood characteristics for location recommendation,” in *CIKM*, 2014, pp. 739–748.

Paper F

Location Influence in Location-Based Social Networks

Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Xike Xie, Torben Bach Pedersen

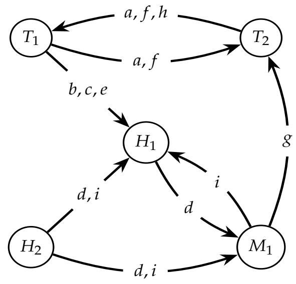
The paper has been published in the
10th ACM International Conference on Web Search and Data Mining, WSDM 2017, pp.621–623, 2017.

Abstract

Location-based social networks (LBSN) are social networks complemented with location data such as geo-tagged activity data of its users. In this paper, we study how users of a LBSN are navigating between locations and based on this information we select the most influential locations. In contrast to existing works on influence maximization, we are not per se interested in selecting the users with the largest set of friends or the set of locations visited by the most users; instead, we introduce a notion of location influence *that captures the ability of a set of locations to reach out* geographically. *We provide an exact on-line algorithm and a more memory-efficient but approximate variant based on the HyperLogLog sketch to maintain a data structure called* Influence Oracle *that allows to efficiently find a top-k set of influential locations. Experiments show that our algorithms are efficient and scalable and that our new location influence notion favors diverse sets of locations with a large geographical spread.*

Check-ins

loc	Users		
	t=1	t=2	t=3
T_1	b,c,e,f	a,h	f
T_2	a,h	f,g	a
M_1	g	i	d
H_1	–	b,c,d,e	i
H_2	d,i	–	–



Friendships

User	Friends
a	c,h,i
b	d,f,i
c	a,f
d	b,e
e	d,g
f	c,b
g	e
h	a
i	a,b

Fig. F.1: Running example of a LBSN

1 Introduction

One of the domains in social network analysis [1–4] that received ample attention over the past years is *influence maximization* [5], which aims at finding influential users based on their social activity. Applications like viral marketing utilize these influential users to maximize the information spread for advertising purposes [6]. Recently, with the pervasiveness of location-aware devices, social network data is often complemented with geographical information. For instance, users of a social network share geo-tagged content such as locations they are currently visiting with their friends. These social networks with location information are called location-based social networks (LBSN). In LBSNs, the location information offers a new perspective to view users' social activities. This information can be utilized to provide more constructive marketing strategies. For example, unlike viral marketing which focuses on finding influential users and spreading the message via word of mouth marketing (WOMM), influential locations can be found and information can be spread using outdoor marketing (OOH) e.g., by putting advertisements on billboards and distributing promotional items on such locations.

In this paper, we study navigation patterns of users based on LBSN data to determine influential locations. Where other works concentrate on finding influential users [7], popular events [8], or popular locations [9], we are interested in identifying sets of locations that have a large *geographical* impact. Although often overlooked, the geographical aspect is of great importance in many applications. For instance, consider the following example.

Example 1.1
A marketer is interested in creating visibility of her products to the maximum regions in a city by offering free promotional items say T-shirts with

a printed promotional message. To do that she would like to choose locations where she should distribute the promotional items to visitors.

In order to choose the most suitable locations for offering these items, not only the popularity of the places is important, but also the geographical reach. By visiting other locations, people that were exposed to the advertisement, especially the receivers of the promotional items, may indirectly promote the products. For example, by wearing the shirt they expose the T-shirt's message to the people of the places they go to later and talk about it with their friends and relatives etc. Thus, when the goal is to create awareness of the product name, it may be preferable to have a moderate presence in many locations throughout the whole city rather than high impact in only a few locations. An illustration of this example is given in Figure 1. Nodes represent popular locations of different categories, such as tourist attractions (T_1, T_2), a metro station (M_1), and hotels (H_1 and H_2). Lowercase letters represent users. For each user, her friends in the social network and check-ins have been given. The top-2 locations with the maximal number of unique visitors are T_1, and M_1. The geographical impact of these locations, however, is not optimal; visitors of these locations reach only T_2 and H_1. On the other hand, the visitors of T_1 and H_2 visit all locations, i.e., users a, f and b, c, e visits T_2 and H_1 after visiting T_1, respectively, and d, i after H_2 visits H_1 and M_1.

To capture geographical spread and influence, in Section 3 we introduce the notion of a *bridging visitor* between two locations as a user that visits both locations within a limited time span. If there are many bridging visitors from one location to another, we say that there is an influence. We introduce different models that capture when the number of bridging visitors is considered to be sufficient to claim influence between locations. One model is based on the absolute number of visitors, one on the relative number, and we also have variants that take the friendship graph into account. Based on these models, we define influence for sets of locations and the *location influence maximization problem*: *Given a LBSN and a parameter k, find a set of k locations such that their combined location influence on other locations is maximal.*

To solve this problem, in Section 4 a data structure, called *Influence Oracle* (Oracle *in short*), is presented that maintains a summary of the LSBN data that allows to determine the influence of any set of locations at any time. Based on this data structure, we can easily solve the location influence maximization problem using a greedy algorithm. As for large LBSNs with lots of activities the memory requirements of our algorithm can become prohibitively large, we also develop a more memory-friendly version based upon the well-known HyperLogLog sketch [10].

In Section 5 we analyze several LBSNs to select reasonable threshold values for our models. In Section 6 the effectiveness and efficiency of our algorithms are demonstrated on these datasets. In a qualitative experiment, the effect of our new location influence notion is illustrated.

In summary, the main contributions of this paper are (i) the introduction and motivation of a new location influence notion based on LBSN data, (ii) the development of an efficient online Influence Oracle, and (iii) the demonstration of the usefulness of the location influence maximization problem in real-life LBSNs.

2 Related Work

Influence maximization in the context of social networks has already been studied in much detail [11–13]. We focus here mainly on works that study the identification of influential users, events, or locations from LBSNs data. We divide the studies into two groups. The first group covers studies using check-ins as an additional source of data to identify influential users, whereas the second group utilizes the check-ins for finding influential locations.

Influential users and events. Zhang et al. [8] use social and geographical correlation of users to find influential users and popular events. Users with many social connections are considered influential as well as events visited by them. Similarly, Wu et al. [7] identify influential users in LBSNs on the basis of the number of followers of their activities (check-ins). Li et al. [14] and Bouros et al. [15] on the other hand, identify regionally influential users on the basis of their activities. The focus of the work by Wen et al. [16] and Zhou et al. [17] is to find and utilize the influential users for product marketing strategies such as word-of-mouth. Our focus, however, is to find influential *locations* that could be used, e.g., for outdoor marketing. None of the previous works applies directly to our problem.

Influential locations in LBSNs. Zhu et al. [9], Hai [18], and Wang et al. [19] study location promotion. Given a target location, their aim is to find the users that should be advertised to attract more visitors to this location. Doan et al. [20] computes the popularity ranks of locations based on the number of visitors. On the other hand, in Zhou et al. [17] study the problem of choosing an optimal location for an event such that the event's influence is maximized; that is, they aim at finding a single location which attracts most users.

Novelty. Our work is different from all of the above as we focus on finding a *set of influential locations* where influence is defined using visitors as a mean to spread influence to other locations. Applications include outdoor marketing by selecting locations with maximal geographical spread.

3 Location-based Influence

We first provide preliminary definitions and then present location influence. Moreover, we formally define the *Oracle* problem and *Location Influence Maximization* problem.

3.1 Location-based Social Network

Let a set of users U and a set of locations L be given.

Definition F.1. *An* activity *is a visit/check-in of a user at a location. It is a triplet* (u, l, t)*, where* $u \in U$ *is a user,* $l \in L$ *a location and* t *is time of the visit of* u *at* l*. The set of all activities over* U *and* L *is denoted* $\mathcal{A}(U, L)$*.*

Definition F.2. *A* Location-based Social Network (LBSN) *over* U *and* L *consists of a graph* $G_S(U, F)$*, called* social graph*, where* $F \subseteq \{\{u, v\} | u, v \in U\}$ *represents friendships between users, and a set of activities* $A \subseteq \mathcal{A}(U, L)$*. It is denoted* $LBSN(G_S, A)$*.*

3.2 Models of Location-based Influence

We define the influence of a location by its capacity to spread its visitors to other locations. The intuition behind this is to capture a location on the basis of its ability to spread its visitors that are exposed to a message, to other locations. Thus, the location influence indirectly captures the capability of a location to spread a message to other geographical regions. Recall our running example that depicts the influence of locations in Figure 1. We can further filter the locations on the basis of their categories to find the particular type of influential and influenced locations. For example, in Figure 1, by considering the hotels as influential locations and their influence only on tourist attractions (influenced locations), the most influential hotels can be found which can spread the information to the maximum number of tourist attractions. The effect of an activity in a location, however, usually remains effective only for a limited time. We capture this time with the *influence window* threshold ω. Visitors that travel from one location to another within a time ω are called *Bridging visitors*:

Definition F.3. *Bridging Visitor: Given* $LBSN(G_S, A)$ *and* ω*, a user* u *is said to be a* bridging visitor from location s to location d *if there exist activities* $(u, s, t_s), (u, d, t_d) \in A$ *such that* $0 < t_d - t_s \leq \omega$*. We denote the set of all bridging visitors from* s *to* d *by* $V_{B(\omega)}(s, d)$*.*

The influence of a location s is measured by two factors, i.e., the number of locations that are influenced by s and the impact by which s influences the locations. The impact of an influence between two locations s and d is captured by the influence models (M).

Absolute Influence Model (M_A)

In practice, if a significant number of people perform an activity, then it is considered compelling. Thus, in order to avoid insignificant influences among locations, we use a threshold τ_A. The influence of a location s on a location d is considered only if the number of bridging visitors from s to d is greater than τ_A. The influence of a location s on d under M_A is represented by $I_{A(\omega,\tau_A)}(s,d)$:

$$I_{A(\omega,\tau_A)}(s,d) := \begin{cases} 1, & \text{if } |V_{B(\omega)}(s,d)| \geq \tau_A \\ 0, & \text{otherwise} \end{cases} \tag{F.1}$$

We omit ω and τ_A from the notations when they are clear from the context.

Example 3.1
Consider the running example of Figure 1. Let $\tau_A = 2$ and $\omega = 2$. Then, $I_A(T_1, H_1) = 1$ because $|V_B(T_1, H_1)| = 3$ ($\geq \tau_A$). Similarly, $I_A(H_2, H_1) = 1$. However, $I_A(M_1, H_1) = 0$ because $|V_B(M_1, H_1)| = 1(\ngeq \tau_A)$.

The influence between two locations may change with the value of τ_A and ω. For example, if we update the value of τ_A to 3 and ω to 2, $I_A(T_1, H_1) = 1$, however, $I_A(H_2, H_1)$ becomes 0 because $|V_B(H_2, H_1)| = 2(\ngeq \tau_A)$.

Relative Influence Model (M_R)

In M_A, the influences of two pairs of locations are considered equal as long as the number of their bridging visitors is greater than τ_A. Sometimes, however, the relative number of contributed bridging visitors is important. Consider, for example, a popular location s that attracts many visitors and a non-popular location d with few visitors. In such a setting, to capture the influence of s on d, we may have to set the absolute threshold τ_A very low. This low value of τ_A, however, may result in many other popular locations being influenced by s even if only a very small fraction of their visitors come from s. Therefore, in such situations, it may be beneficial to use different thresholds for different destinations, relative to the number of visitors in these destination locations. This notion is captured by the *relative influence model (M_R)*. The influence of s on d under M_R is represented by $I_{R(\omega,\tau_R)}(s,d)$ and is parameterized by the relative threshold τ_R:

$$I_{R(\omega,\tau_R)}(s,d) := \begin{cases} 1, & \text{if } \dfrac{|V_{B(\omega)}(s,d)|}{|V(d)|} \geq \tau_R \\ 0, & \text{otherwise} \end{cases} \tag{F.2}$$

where $V(d)$ is the set of users who visited location d.

Example 3.2
Consider the running example given in Figure 1. Let $\tau_R = 0.4$ and $\omega = 2$. In this example, $I_R(T_1, H_1) = 1$ because $\frac{|V_B(T_1,H_1)|}{|V_{H_1}|} = \frac{|\{b,c,e\}|}{|\{b,c,d,e,i\}|} = \frac{3}{5} \geq \tau_R$, Similarly, $I_R(H_2, H_1) = 1$ and $I_R(M_1, H_1) = 0$.

3.3 Friendship-based Location Influence

Activity data in LBSNs is often sparse in the sense that the number of check-ins per location is low. In Section 6 we see that in the real-world datasets we use there have only up to 6 check-ins per location on average. This sparsity of data affects the computation of location influence. In order to deal with this issue, we use the observation that users tend to perform similar activities as their friends (This claim is verified and confirmed in Section 5). Hence, we define friendship-based influence between locations, by incorporating also friends of bridging visitors, which we consider *potential visitors*. The set of bridging visitors together with the potential visitors from a location s to d is represented by $V_{Bf(\omega)}(s,d)$, and the set of visitors to a location d together with their friends is denoted $V_f(d)$.

In order to incorporate potential visitors in the influence models, we replace $V_{B(\omega)}(s,d)$ in Equation (F.1) and Equation (F.2) by $V_{Bf(\omega)}(s,d)$, and $V(d)$ in Equation (F.2) by $V_f(d)$. The updated influence of s on d under M_A and M_R respectively are represented by $I_{Af(\omega,\tau_{Af})}(s,d)$ and $I_{Rf(\omega,\tau_{Rf})}(s,d)$. Again, we omit ω, τ_{Af} and τ_{Rf} from the notations when it is clear from the context.

Example 3.3
Let $\tau_{Af} = 2$ and $\omega = 2$. We have $I_{Af}(T_1, H_1) = 1$ because $|V_{Bf}(T_1, H_1)| = |\{a,b,c,d,e,f,g,i\}|$ exceeds τ_{Af}. Similarly, $I_{Af}(H_2, H_1) = 1$ and $I_{Af}(M_1, H_1) = 1$.

Furthermore, let $\tau_{Rf} = 0.4$ and $\omega = 2$. We have $I_{Rf}(T_1, H_1) = 1$ because $\frac{|V_{Bf}(T_1,H_1)|}{|V_{H_1 f}|} = \frac{|\{a,b,c,d,e,f,g,i\}|}{|\{a,b,c,d,e,f,g,i\}|} = 1 (\geq \tau_{Rf})$. Similarly, $I_{Rf}(H_2, H_1) = 1$ and $I_{Rf}(M_1, H_1) = 0$.

3.4 Combined Location Influence

Based on the influence models, a location can influence multiple other locations. In order to capture such influenced locations, we define the *location influence set*:

Definition F.4. *Given a location s, and an influence model M, the* location Influence Set $\phi_{I_M}(s)$ *is the set of all locations for which the influence of s on that location under M is* 1*, i.e.,* $\phi_{I_M}(s) = \{d \in L \mid I_M(s,d) = 1\}$.

Next, we define *combined location influence* for a set of locations S. To do this, we use the following principled approach: any activity at one of the locations of S is considered an activity from S. In that way we can capture the cumulative effect of the locations in S; even though all locations in S in isolation may not influence a location d, together they may influence it. The bridging visitors from a set of locations S to d is represented by $V_{B(\omega)}(S,d)$:

$$V_{B(\omega)}(S,d) = \bigcup_{s \in S} V_{B(\omega)}(s,d) \tag{F.3}$$

The influence of a set of locations S on location d under M_A and M_R is defined similarly as for single locations.

Example 3.4
In Figure 1, let $\omega = 2$, $\tau_A = 3$ and $S = \{T_1, M_1\}$. Under M_A, $T_2 \notin \phi(T_1)$ and $T_2 \notin \phi(M_1)$. However, $T_2 \in \phi(S)$ as $|V_B(S,T_2)| = |\{a,f,g\}| \geq \tau_A$.

3.5 Problem Formulation

Based on these influence models, we now define two problems related to finding influential locations in a LBSN. We first present a problem statement of constructing a data structure that can be utilized for providing many interesting applications called Influence Oracle. Next, we present a problem statement for one such application, i.e., finding the top-k most influential locations.

Problem F.1. *(Oracle Problem) Given a LBSN and an influence model M, construct a data structure that allows to answer: Given a set of locations* $S \subseteq L$ *and a threshold* τ*, what is the combined location influence* $\phi_{I_M}(S)$ *of S.*

Problem F.2. *(Location Influence Maximization Problem) Given a parameter k, a LBSN, and an influence model M, the location influence maximization problem is to find a subset* $S \subseteq L$ *of locations, such that* $|S| \leq k$ *and the number of influenced locations* $|\phi_{I_M}(S)|$ *is maximum.*

4 Solution Framework

We first provide a data structure to solve the Oracle problem. We present an exact algorithm in Section 4.1 and an approximate but more memory-

and time-efficient algorithm in Section 4.2. Finally, in Section 4.3, we solve Problem F.2 with a greedy algorithm.

4.1 Influence Oracle

In this section, we provide a data structure for maintaining location summaries for each location. We assume activities arrive continuously and deal with them one by one. The summary $\varphi(s)$ for a location s consists of the list of all locations to which it has bridging visitors. We present an online algorithm to incrementally update these summaries.

Definition F.5. *The* Complete location summary *for a location $s \in L$ is the set of locations that have at least one bridging visitor from s, together with these bridging visitors; i.e., $\varphi(s) := \{(d, V_B(s,d)) \mid d \in L \wedge |V_B(s,d)| > 0\}$.*

If a user u visits a location s at time t, then u acts as a bridging visitor between all the locations u visited within the last ω time stamps and s. Therefore, for each user $u \in U$, we maintain a set of locations the user has visited and the corresponding latest visiting time. This is called the *visit history* $\mathcal{H}(u)$ and is defined as $\mathcal{H}(u) := \{(s, t_{max}) | u \in V(s), t_{max} = \max\{t \mid (u,l,t) \in A\}\}$. Suppose that we have the complete location summary for the check-ins so far and the visit history of all users, and a new activity (u,d,t) arrives. We update the complete location summary as follows: the location-time pair (d,t) is added in $\mathcal{H}(u)$ if d does not already appear in the visit history, otherwise the latest visit time of d is updated to t in $\mathcal{H}(u)$. Furthermore, for every other location-latest visit time pair (s,t') in the history of u, $\varphi(s)$ is updated by adding user u to the set of bridging visitors from s to d provided that the difference between the time stamps $t - t'$ does not exceed the threshold ω. This procedure is illustrated in Algorithm F.1.

Example 4.1
We illustrate the algorithm using the running example shown in Figure 1. For simplicity, we only consider the activities of two users: d and i. We also add a new activity of d at H_2 at time stamp 5. In this example, we consider $\omega = 2$. The activities are processed one by one in increasing order of time. We show how the visit history $\mathcal{H}(i)$, $\mathcal{H}(d)$ and the complete location summaries $\varphi(H_1)$, $\varphi(H_2)$, $\varphi(M_1)$ evolve with different activities at different time stamp in Figure F.2. Note, at time stamp 5 only $\varphi(M_1)$ is updated even though M_1 and H_1 are both in the visit histories of d because $\omega = 2$. The visit history of d is cleaned by removing H_1 from the $\mathcal{H}(d)$ as no future activities by d affect $\varphi(H_1)$. The visit time of H_2 is updated to the latest visit time. Similarly, $\mathcal{H}(i)$ is also cleaned up.

Algorithm F.1: Updating complete location summaries

```
Input: New activity $(u,d,t)$, threshold $\omega$, $\varphi(l)$ for $l \in L$
Output: Updated $\varphi(.)$ and $\mathcal{H}(.)$
begin
    foreach $(s,t') \in \mathcal{H}(u)$ do
        if $t - t' \leq \omega$ then
            if $(d, V_B(s,d)) \in \varphi(s)$ then
                $V'_B(s,d) \leftarrow V_B(s,d) \cup \{u\}$
                $\varphi(s) \leftarrow \varphi(s) \setminus \{(d, V_B(s,d))\}$
            end
            else
                $V'_B(s,d) \leftarrow \{u\}$
            end
            $\varphi(s) \leftarrow \varphi(s) \cup \{(d, V'_B(s,d))\}$
        end
        else
            $\mathcal{H}(u) \leftarrow \mathcal{H}(u) \setminus \{(s,t')\}$
        end
    end
    if $\exists t' : (d,t') \in \mathcal{H}(u)$ then
        $\mathcal{H}(u) \leftarrow (\mathcal{H}(u) \setminus \{(d,t')\})$
    end
    $\mathcal{H}(u) \leftarrow \mathcal{H}(u) \cup \{(d,t)\}$
end
```

It can be observed from the example that a new activity of a user u only updates the complete location summary of the locations in the recent visit history of u. Notice that, since the activities of a user arrive in strictly increasing order of time, the size of $\mathcal{H}(u)$ is upper bounded by ω, as only locations that are visited within a time window ω are processed. The proofs of the following proposition are trivial and thus omitted.

Proposition F.1. *The time required to process an activity is $\mathcal{O}(\omega \log(|U|))$. The the complete location summary $\varphi(.)$ can be stored in $\mathcal{O}(|L||U|)$ memory and for the visit history $\mathcal{H}(.)$ in $\mathcal{O}(\omega)$ memory.*

The time required to produce $\phi(S)$ from $\varphi(.)$ for given threshold τ and set of locations S is $\mathcal{O}(|S||L||U| \log |U|)$.

Relative and Friendship-based Location Influence. For the relative models, we additionally have to maintain the total number of unique visitors per location, which can be done in the worst case time $O(log(|U|))$ and space

	$t = 1$	$t = 2$	$t = 3$	t=5
Activity:	$(i, H_2, 1)$ $(d, H_2, 1)$	$(i, M_1, 2)$ $(d, H_1, 2)$	$(i, H_1, 3)$ $(d, M_1, 3)$	$(d, H_2, 5)$
$\mathcal{H}(i)$:	$\{(H_2, 1)\}$	$\{(H_2, 1), (M_1, 2)\}$	$\{(H_2, 1), (M_1, 2), (H_1, 3)\}$	$\{(H_1, 3)\}$
$\mathcal{H}(d)$:	$\{(H_2, 1)\}$	$\{(H_2, 1), (H_1, 2)\}$	$\{(H_2, 1), (H_1, 2), (M_1, 3)\}$	$\{(M_1, 3), (H_2, 5)\}$
$\varphi(H_1)$:	$\{\}$	$\{\}$	$\{(M_1, \{d\})\}$	$\{(M_1, \{d\})\}$
$\varphi(H_2)$:	$\{\}$	$\{(H_1, \{d\}), (M_1, \{i\})\}$	$\{(H_1, \{d\}), (M_1, \{i, d\})\}$	$\{(H_1, \{d\}), (M_1, \{i, d\})\}$
$\varphi(M_1)$:	$\{\}$	$\{\}$	$\{(H_1, \{i\})\}$	$\{(H_1, \{i\}), (H_2, \{i\})\}$

Fig. F.2: Updating $\varphi(l)$ and $\mathcal{H}$ for $\omega = 2$ for M_A

$O(|U|)$ per activity and hence does not affect the overall complexity. For the friendship-based location influence, for every activity, we process the same activity at the same time for all friends as well. As the number of friends is bounded by $|U|$, we get:

Proposition F.2. *The time required to process an activity in the friendship-based influence models is $\mathcal{O}(\omega|U| \log(|U|))$. The memory required is the same as for the other models.*

4.2 Approximate Influence Oracle

In the worst case the memory requirements of the exact algorithm presented in the last section are quite stringent: for every pair of locations (s, d), in $\varphi(s)$ the complete list of bridging visitors from s to d is kept. Therefore, here we present an approximate algorithm for maintaining the complete location summaries in a more compact form. This compact representation will represent a significant saving especially in those cases where the window size ω is large since in that case the number of bridging visitors increases.

We observe that when computing the number of bridging visitors between s and d we do not need the set of bridging visitors between s and d, but only the cardinality of that set. For the relative number of bridging visitors, we additionally need only the numbers of visitors $|V(s)|$. Furthermore, as per Equation F.3, in order to find the accumulated complete location summary, we need to combine two complete location summaries; for instance: the complete location summary $\varphi(\{s_1, s_2\})$ is obtained by taking the following pairwise union of $\varphi(s_1)$ and $\varphi(s_2)$: if $\varphi(s_1)$ and $\varphi(s_2)$ respectively contain the pairs $(d, V_B(s_1, d))$ and $(d, V_B(s_1, d))$, then $\varphi(\{s_1, 2_2\})$ contains $(d, V_B(s_1, d) \cup V_B(s_2, d))$. But then again, for further computations, we only need the cardinality of the bridging visitor sets. Hence, if we accept approximate results, we could replace the exact set $V_B(s, d)$ with a succinct sketch of the set that allows to take unions and get an estimate of the cardinality of the set. In our algorithm, we use the HyperLogLog sketch (HLL) [10] to replace the exact sets $V_B(s, d)$ and $V(s)$. The HLL sketch is a memory-efficient data structure of size 2^k that can be used to approximate the cardinality of a set by using an array. The constant k is a parameter which determines the accuracy of the approximation and is in our experiments in the order of 6 to 10. Furthermore, the HLL sketch allows unions in the sense that the HLL sketch of the union of two sets can be computed directly from the HLL sketches of the individual sets. For our algorithm, we consider the HLL algorithm as a black box. By using HLL, we not only reduces memory consumption but also improve computation time, because adding an element in a HLL sketch can be done in constant time and taking the union of two HLL sketches takes time $\mathcal{O}(2^k)$; that is: the time to take the union of two sets is independent of the size of the sets.

Proposition F.3. *Let $b = 2^k$ be the number of buckets in the HLL sketch. The time needed to process an activity using the HLL sketch is $\mathcal{O}(\omega)$. The memory required to maintain the complete location summary is $\mathcal{O}(|L|b)$.*

4.3 Influence Maximization

In order to solve the location influence maximization problem, we apply the standard greedy algorithm to compute top-k as obtaining an exact solution is intractable as the next proposition states.

Proposition F.4. *The following problem is **NP**-hard for all influence models: given a LBSN and bounds k and β, does there exist a set of locations S of size k such that $|\phi(S)| \geq \beta$.*

Proof. **NP**-hardness follows from a reduction from set cover. Consider an instance $\mathcal{S} = \{S_1, \ldots, S_m\}$ with all $S_i \subseteq \{1, \ldots, n\}$ and bound k of the set cover problem: does there exist a subset $\mathcal{S}'$ of $\mathcal{S}$ of size at most k such that $\bigcup \mathcal{S}' = \{1, \ldots, n\}$. We reduce this instance to a LBSN as follows: $L = \{l_1, \ldots, l_n\} \cup \{s_1, \ldots, s_m\}$, $U = \{u_1, \ldots, u_m\}$, $F = \emptyset$, $A = \{(u_i, s_i, 0) \mid i = 1 \ldots m\} \cup \{(u_i, l_j, j) \mid i = 1 \ldots m, j \in S_i\}$. That is, every element j of the domain $\{1, \ldots, n\}$ is associated to a location l_j, and for every set S_i we introduce a location s_i visited by user u_i at time 0. Furthermore, user u_i visits all locations l_j such that $j \in S_i$ at time stamp j. If we use the absolute model with $\tau = 1$ and $\omega \geq n + 1$, for $i = 1 \ldots m$, $\phi(\{s_i\}) = \{l_j \mid j \in S_i\}$. As such there exists a set cover of size k if and only if there exists a set of locations S of size k such that $|\phi(S)| = n$. □

Recall that the influence of a set of locations S is computed by accumulating the effect of all locations in S. It is hence possible that two locations s and s' separately do not influence a target location d because individually they have too few bridging visitors to d, but together they reach the threshold. This situation occurs for instance in Figure 1, for the locations H_2 and M_1. These locations individually do not reach the threshold to influence H_1 for $\tau_A = 2$ and $\omega = 1$. However, together they do. One inconvenient consequence of this observation is that the influence function that we want to optimize is not sub-modular [21]. Indeed, in the example above, adding H_2 to the set $\{M_1\}$ gives a higher additional benefit (1 more influenced location) than adding H_2 to $\{\}$. Therefore, we do not have the usual guarantee on the quality of the greedy algorithm for selecting the top-k.

The main reason that we do not have the guarantee is that the benefit is not gradual; before the threshold is reached it is 0, after the threshold is reached it is 1. This means that a location that has $\tau - 1$ bridging visitors to 1000 other locations each, gives the same benefit as a location that does not have any bridging visitors. Clearly, nevertheless, the first location is more likely to lead to a good solution if later on additional locations are selected. Therefore, we would like to incorporate potential future benefits into our objective function. Thus, in order to compute the influence of a location, we consider locations that are influenced as well as those locations that are not yet influenced but have potential to be so in future. To characterize the potential of future benefit in combination with the number of influenced locations, we use the following formula:

$$LI(S) = (1-\alpha) \times |\phi(S)| + (\alpha) \times \sum_{d \in L-S} (\min\{|V_B(S,d)|, \tau\}) \tag{F.4}$$

In this formula, $\alpha = [0,1]$ represents a trade-off between the number of influenced locations and a reward for potential influenced locations. For relative models, we replace the $|V_B(S,d)|$ with $|V_B(S,d)|/|V(d)|$.

Next, we apply a greedy method on the basis of location influence to find top-k locations. We start with an empty set S of locations and iteratively add locations to it until we reach the required number of top elements: k. In each step, for each location $s \in L$, we evaluate the effect of adding s to S, and keep the one that gives the highest benefit $LI(S)$. Then, we update $S \leftarrow S \cup \{l\}$.

Example 4.2
Consider the case in Figure F.2 for $\omega = 1$, $\varphi(H_2) = \{(H_1, \{d\}), (M_1, \{i\})\}$, $\varphi(M_1) = \{(H_1, \{i\})\}$ and $\varphi(H_1) = \{(M_1, \{d\})\}$. We aim to find top-2 locations in this example with $\alpha = 0.1$ and $\tau = 2$. During the first iteration, $LI(H_2) = 0.9 \times 0 + 0.1 \times (1+1) = 0.2$, because H_2 does not completely influence any other location, however H_1 and M_1 are potential influenced locations for the bridging visitors d and i, respectively. Similarly,

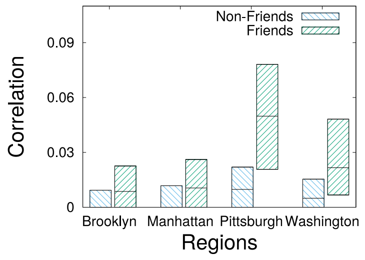


Fig. F.3: Visit correlations

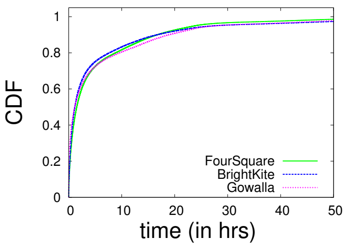


Fig. F.4: CDF of visit time

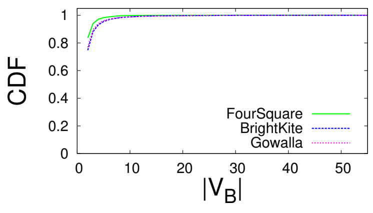
$LI(M_1) = 0.1$ and $LI(H_1) = 0.1$. Thus, we choose H_2 as first seed as it has maximum value. In the next iteration, we first combine the seed H_2 with M_1 and compute the combined influence. Here, $LI(\{H_2, M_1\}) = 0.9 \times 1 + 0.1 \times (2) = 1.1$. Similarly, $LI(\{H_2, H_1\}) = 1.1$. Since, M_1 and H_1 provide equal benefit of 0.9, when combined with H_2, thus we can randomly choose either M_1 or H_1 as a second seed.

5 LBSN Data Analysis

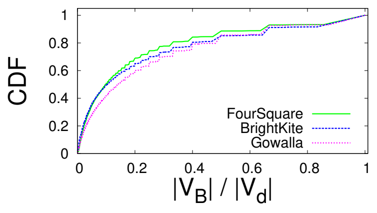
When constructing the friendship-based influence model the assumption was made that friends tend to follow friends. Furthermore, the influence models of Section 3.3 have several parameters to set: τ and ω. Before going to the experiments, first in this section we verify and confirm the friendship assumption and show how to set the thresholds with reasonable values based on an analysis of the LBSN datasets given in Table F.1.

5.1 Mobility analysis of friends

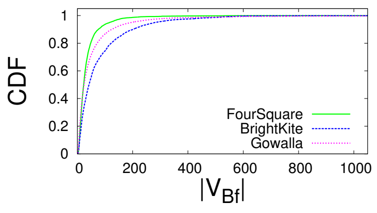
In real life, usually activities of friends are more similar than activities of non-friends. In LBSNs, this implies that a visit of a user to a location increases the chances of visits of his/her friends to the same location. We considered this assumption when constructing our friendship-based influence model in Section 3.3. We illustrate the correctness of this assumption by computing the correlations between activities of users, their friends, and non-friends: Let L_u and L_v be the locations visited by users u and v, respectively. The correlation between activities of u and v is measured by the Jaccard Index [22] between L_u and L_v. The average correlation of activities of users and those of their friends is denoted *friendship correlation* (p^f_{corr}), and the average correlation



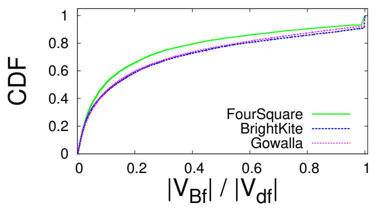
(a) Absolute without-friends



(b) Relative without-friends



(c) Absolute with-friends



(d) Relative with-friends

Fig. F.5: Cumulative distribution function (CDF) of thresholds for all influence propagation models

between activities of users and their non-friends is denoted *Non-friendship Correlation* (p^{nf}_{corr}). In order to avoid an unreasonable bias due to the fact that friends tend to live in the same city, we restrict our computation of the average non-friendship correlation to users in the same city. We randomly picked four regions of the United States, i.e., Brooklyn, Manhattan, Pittsburgh, and Washington and consider the activities of users in these regions to study the correlations. The statistics of p^{f}_{corr} and p^{nf}_{corr} of all the users are given in Figure F.3. The figure presents boxplots without outliers. It can be seen that median of p^{f}_{corr}, even though still small, is up to 5 times larger than p^{nf}_{corr}. The same pattern is observed for all the datasets, thus only results for `Gowalla` are shown due to space constraints. This validates the claim that the activities of friends are more similar than non-friends.

5.2 Setting ω and τ

In order to determine the value of influence window threshold ω, we measure the time difference between consecutive visits of users to distinct locations. The cumulative distribution functions (CDF) for three LBSNs are given in Figure F.4. It can be seen that for all LBSNs in our study, 80% of the consecutive activities are performed within 8 hours. After that, there is only a moderate increase in the number of activities with respect to the time interval. Thus, in order to capture only the most common activities, we keep $\omega = 8$. However, it can, of course, be changed if the data distribution is different, or there are different user or application requirements.

We furthermore compute the absolute and relative number of bridging visitors. In order to do that, we consider both the models with-friends and without-friends, for each pair of locations with at least one bridging visitor. The cumulative distribution functions for each of these numbers are depicted in Figure F.5. We can utilize the CDF values for controlling the number of influences in the dataset, and thus also for finding the suitable values of thresholds for models. The values of thresholds are an application dependent choice and can be considered accordingly. For example, if an application requires to find many influential relationships, and indirectly many influential and influenced locations, then a lower threshold should be considered and vice versa. In this paper, we consider the top 20% influential relationships among locations for all the models. Thus, the thresholds for all the models are their corresponding CDF values of 0.8 (100%-20%=80%). Therefore, the values of $\tau_A, \tau_R, \tau_{Af}$ and τ_{Rf} are $2, 0.4, 120$ and 0.6, as shown in Figures F.5a, F.5c, F.5b, and F.5d, respectively.

6 Evaluation

We conducted our experiments on a Linux machine with Intel Core i5-4590 CPU @3.33GHz CPU and 16 GB of RAM, running the Ubuntu 14 operating system. We implemented the exact and the approximate algorithms in C++.

Datasets. We used 3 real-world datasets : `FourSquare` [23], `BrightKite`, and `Gowalla` [24]. These datasets each consisted of two parts: the friendship graph and an ordered list of check-ins. A check-in record contains the user-id, check-in time, GPS coordinates of location, and a location-id. The statistics of the datasets are given in Table F.1.

Data Prepossessing. The real-life datasets required preprocessing because many locations are associated with multiple location identifiers with slightly different GPS coordinates. Consider, for instance, Figure F.6. In this figure, 13 GPS coordinates that appear in the `FourSquare` dataset are shown

	Users	Locations	Check-ins	POIs
`FourSquare`	16K	803K	1.928M	582K
`BrightKite`	50K	771K	4.686M	631K
`Gowalla`	99.5K	1.257M	6.271M	1.162M

Table F.1: Statistics of datasets

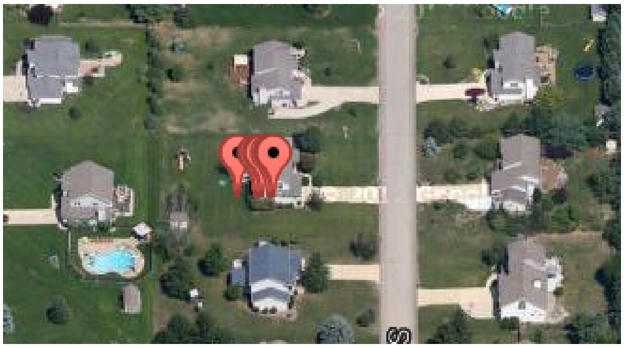


Fig. F.6: GPS coordinate of 13 location-ids on GoogleMaps

which corresponds to different locations Ids in the dataset, but which clearly belong to one unique location. In order to resolve this issue, we clustered GPS points to get POIs. We used the density-based spatial clustering algorithm [25] with parameters *eps*=10 meters and *minpts*=1 to group the GPS points. New location Ids are assigned to each cluster which were used in all our experiments. All 3 datasets have similar problems. The statistics of the new Ids are reported in column POIs of Table F.1.

			No. of Buckets (b)		
			64	128	256
Rel. error	Abs.	mean $\pm\sigma$	0.02 ± 0.15	0.01 ± 0.1	0.01 ± 0.08
	Abs. friends	mean $\pm\sigma$	0.167 ± 0.63	0.08 ± 0.45	0.04 ± 0.49
	Rel.	mean $\pm\sigma$	0.06 ± 0.23	0.06 ± 0.23	0.06 ± 0.23
	Rel. friends	mean $\pm\sigma$	0.05 ± 0.21	0.05 ± 0.21	0.05 ± 0.2
Time	with-out	Exact	38.7		
	friends	Approx	40	37.5	42.9
	with	Exact	389.6		
	friends	Approx	61.9	67.1	70.9
Memory	with-out	Exact	505		
	friends	Approx	531	644	835
	with	Exact	3790		
	friends	Approx	541	658	855

Table F.2: Exact vs Approx algorithm comparison for accuracy (relative error), time (sec) and memory (MB)

6.1 Approximate vs. Exact Oracle

We analyzed the accuracy of the influence approximation based on the HLL sketch. We also analyzed memory consumption and computation time improvement for the approximate approach. The results are similar for all the datasets and hence we only present results for `BrightKite` due to space constraints.

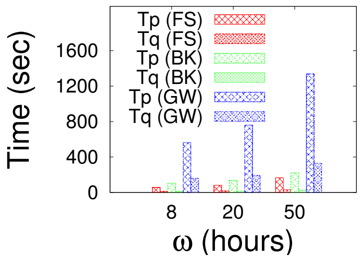
Approximation Accuracy. For every location with a non-empty influence set, we used the HLL-based approximate version of the Oracle to predict the size of the influence set. Then the relative error as compared to the real size was computed for every location. In Table F.2 the mean and standard deviation of this relative approximation error over all locations with a non-empty influence are given. The experiments are performed for both with-friends and without-friends for the absolute influence model and relative influence model. We ran the experiments for different numbers of buckets (b) for the *HLL* sketch, being, $64, 128$ and 256. As can be seen in the table, the errors are unbiased (0 on average), and the standard deviation decreases as the number of buckets increases. The error is a bit higher in the relative model as compared to the absolute model because in the relative model the influence is computed by taking the ratio of two approximated sets. Values for b beyond 256 yielded only modest further improvements and hence we used $b = 256$ in all further experiments.

Approximation Efficiency. Next, we compare the computation time and memory requirements for the approximate approach with that of the exact approach. In order to do so, we computed influence sets with friends and without friends. The computation times and memory consumption are shown in Table F.2. The approximate approach outperforms the exact approach up to a factor 6 in time using only 15% of memory for the models including friends. Due to the sparsity of data, however, the gain for the without-friend case is negligible. This is because the sizes of the sets of bridging visitors are very modest and hence there is no need to reduce memory consumption. It can be observed that time and memory of the approximate approach increase with increasing number of buckets b.

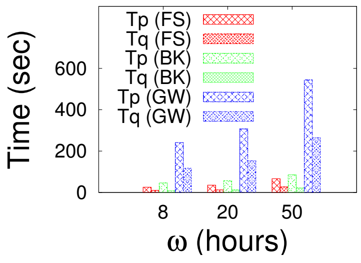
[h]

6.2 Influence of ω and τ

Runtime. We study the runtime of the approximate algorithm on all the datasets for different values of $\omega := 8, 20$ and 50. The average runtime for processing all the activities (T_p) under the models varies only depending on whether or not we consider friends; it does not depend on τ. The oracle query time (T_q) is independent of τ and model. Hence we only show results for $\tau = 2$. The run times are shown in Figure F.7 for the three datasets



(a) with-Friends



(b) without-Friends

Fig. F.7: Time to process all activities (Tp) and query Oracle (Tq) for $\tau = 2$ at different ω

`FourSquare`, `BrightKite` and `Gowalla`. The running time increases with increasing influence window size ω as more locations from the visit history remain active. Running time is higher in the with-friends case which is not surprising either as the number of users to include in the bridging visitors sets increases due to the addition of friends. The time taken to process dataset `Gowalla` is the highest as it has the largest number of locations.

In Figure F.9b, we report the time taken in function of the number of activities for $\omega = 8$. Per 1,000 activities in the `BrightKite` dataset the runtime is reported. As can be seen in the figure, the average time taken per 1,000 activities remains constant. The time taken for the friendship-based influence model is the highest as more users are merged.

Memory Consumption. We also study the memory required by the approximation algorithm on all the datasets for different values of $\omega := 8, 20$ and 50. Unlike for the processing time, the average memory required to pro-

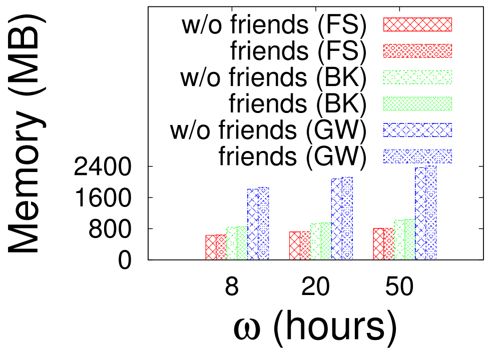
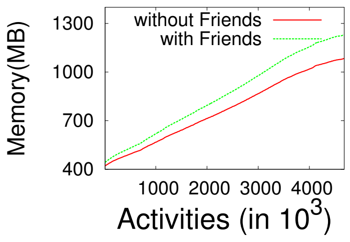
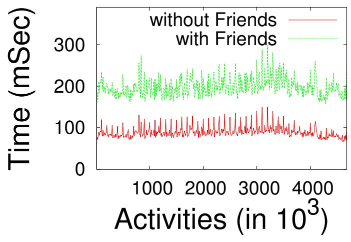


Fig. F.8: Memory to process all activities at different ω



(a) Memory



(b) Time

Fig. F.9: Performance evaluation for processing 1000 activities for $\omega = 8$

cess all the activities under the models does not vary based on whether we consider friends or not. This is because the HLL sketch storing the bridging visitor set size remains constant in size even if a larger number of users is added to it. The memory requirement increases slightly with ω as more locations are getting influenced due to a larger influence window. The results are shown in Figure F.8. In Figure F.9a, we report the memory used as a function of the number of activities for $\omega = 8$. Per 1,000 activities in the `BrightKite` dataset the runtime is reported. The total memory requirements increase linearly with time as new locations come in over time for which a complete influence summary needs to be maintained. In Figure F.10 on the other hand, we see that over time the size of user visit history remains constant due to the pruning of outdated locations in the visit histories.

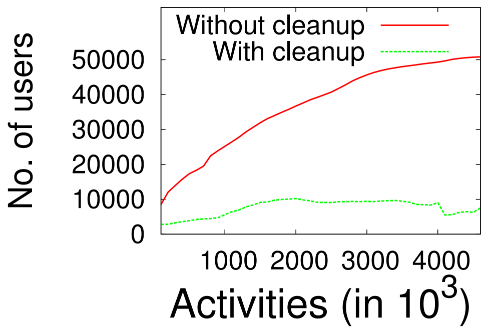


Fig. F.10: User visit history growth w.r.t. cleanup process

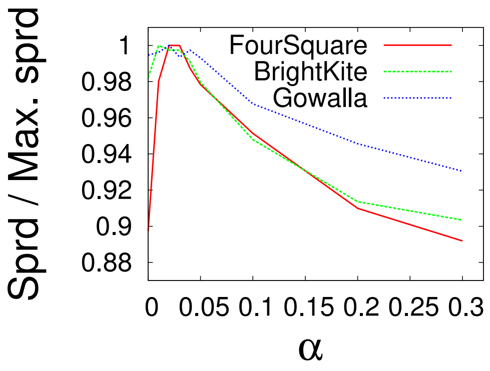


Fig. F.11: Influence spread w.r.t. alpha (200 seeds)

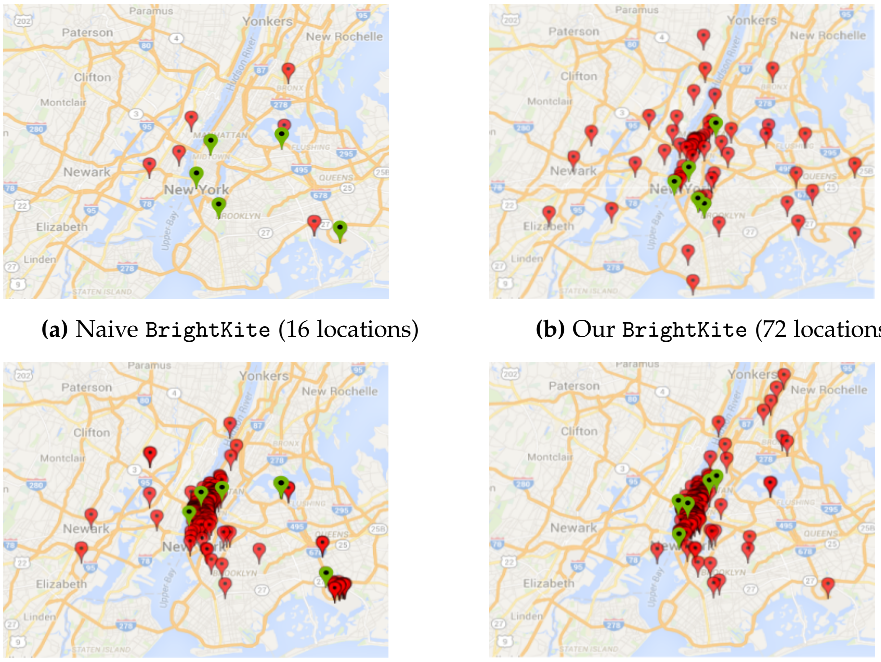
6.3 Influence Maximization

Influence of α. Our next goal is to study how the influence maximization algorithm performs for different values of α. In order to avoid data sparsity issues, we filter out those locations which have only one visitor from all the datasets. We tested the spread of top 200 locations obtained by considering values of α from 0.01 to 0.99. We observed that the number of bridging visitors per location is highly skewed as can be learn from Figure F.5a. Due to this, the potential influenced locations having few bridging visitors are less likely to affect the influenced set of the locations. The effect of varying alpha on the influence spread is shown in Figure F.11. As expected for these sparse datasets, our algorithms perform best with a lower value of α. We use $\alpha = 0.03$ for our experiments.

τ	Time (sec)		
	$k = 10$	$k = 20$	$k = 50$
$\tau_A = 2$	2	3	35
$\tau_R = 0.4$	5	6	46
$\tau_{Af} = 120$	2	5	46
$\tau_{Rf} = 0.6$	4	6	53

Table F.3: Time taken to find top k locations (`BrightKite`)

Computation time. We study the computation time for finding top-k influential locations under both the with-friends and the without-friends influence models. The runtime is close in the both absolute and relative models. The time increases with k. Nevertheless, the increase is modest; for instance,



(a) Naive `BrightKite` (16 locations) **(b)** Our `BrightKite` (72 locations)

(c) Naive `FourSquare` (239 locations) **(d)** Our `FourSquare` (239 locations)

Fig. F.12: Comparison of top- 5 influential locations (green) and their spread (red) between naive and our approach

finding the top-50 locations takes less than a minute. We report the results in Table F.3.

6.4 Qualitative Experiment

In order to demonstrate our model of location influence, we compared the results of our method with a naive approach for selecting top-k locations. In the naive approach, we selected the top k locations such that the number of distinct users visiting those locations is maximized. This result is compared to the top-k most influential locations found using the absolute influence model with $\tau = 1$. We compared the influence spread by the top-k locations of both approaches.

We considered the activities performed in the area of New York in all the three data-sets and fetched top-5 locations for $\omega = 8$ *hours* for both approaches. We further computed the influence spread for the selected locations of both approaches using the absolute influence model. Top-5 locations with their influenced locations are plotted using Google Maps as shown in Figure F.12 for `FourSquare` and `BrightKite`. In the figure, it can be observed that for BrightKite our method leads to a set of locations with a much larger spread as compared to the naive approach, both geographically and in terms of the number of locations influenced. On the other hand, the spread for both approaches for FourSquare is similar. The reason is that for this dataset the problem of selecting the top locations is almost trivial as there is only a small set of locations visited multiple times with as a result that once this limited set of locations is selected, it does not matter which other users are selected.

7 Conclusion and Future Work

In this paper, we introduced a notion that can be used to optimize outdoor marketing strategies such as finding optimal locations for advertising products to maximize the geographical spread. In order to do that, we captured the interactions of locations on the basis of their visitors to compute the influence of locations among each other. We provided two models namely the absolute influence model and the relative influence model. We further incorporated friends of users in order to deal with data sparsity. We proposed an Oracle data structure to efficiently compute the influence of locations on the basis of these models. Oracle can be used for different applications such as finding top-k influential locations. In order to maintain this data structure, we first provided a set-based exact algorithm. Then, we optimized the time and memory requirements of the algorithm up to 6 times and 7 times, respectively, by utilizing a probabilistic data structure. Finally, we provided a greedy algorithm to compute the top-k influential locations. In order to evaluate the methods, we utilized three real datasets. We first analyzed the LBSN datasets: FourSquare, BrightKite and Gowalla to verify some claims and to provide optimal values for thresholds of the influence models. Then, we evaluated our approaches for the computation of the Oracle and finding top-k locations in terms of accuracy, computation time, memory requirement and scalability. We further show the effectiveness of our proposed models by comparing the influence spread of top-k locations fetched by our approach with that of a naive approach.

In the future, we plan to enrich location influence models by incorporating the activities users perform with their friends in groups. Moreover, we aim to provide distributed mechanisms for computing the Oracle data structures and influences for the models.

References

[1] A. AlDwyish, E. Tanin, and S. Karunasekera, "Location-based social networking for obtaining personalised driving advice," in *Proceedings of the*

23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, ser. GIS '15, 2015, pp. 91:1–91:4.

[2] L. Ferrari, A. Rosi, M. Mamei, and F. Zambonelli, "Extracting urban patterns from location-based social networks," in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, ser. LBSN '11, 2011, pp. 9–16.

[3] F. J. Mata and A. Quesada, "Web 2.0, social networks and e-commerce as marketing tools," *J. Theor. Appl. Electron. Commer. Res.*, vol. 9, no. 1, pp. 56–69, 2014.

[4] M. A. Saleem, X. Xie, and T. B. Pedersen, "Scalable processing of location-based social networking queries," in *MDM*, 2016, pp. 132–141.

[5] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003, pp. 137–146.

[6] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *KDD*, 2009, pp. 199–208.

[7] H.-H. Wu and M.-Y. Yeh, "Influential nodes in a one-wave diffusion model for location-based social networks," in *PAKDD*, 2013, pp. 61–72.

[8] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei, "Evaluating geo-social influence in location-based social networks," in *CIKM*, 2012, pp. 1442–1451.

[9] W.-Y. Zhu, W.-C. Peng, L.-J. Chen, K. Zheng, and X. Zhou, "Modeling user mobility for location promotion in location-based social networks," in *KDD*, 2015, pp. 1573–1582.

[10] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *DMTCS*, 2008.

[11] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "A data-based approach to social influence maximization," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 73–84, 2011.

[12] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.

[13] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *ICDM*, 2010.

[14] G. Li, S. Chen, J. Feng, K.-l. Tan, and W.-s. Li, "Efficient location-aware influence maximization," in *SIGMOD*, 2014, pp. 87–98.

[15] P. Bouros, D. Sacharidis, and N. Bikakis, “Regionally influential users in location-aware social networks,” in *SIGSPATIAL*, 2014.

[16] Y.-T. Wen, P.-R. Lei, W.-C. Peng, and X.-F. Zhou, “Exploring social influence on location-based social networks,” in *ICDM*, 2014, pp. 1043–1048.

[17] T. Zhou, J. Cao, B. Liu, S. Xu, Z. Zhu, and J. Luo, “Location-based influence maximization in social networks,” in *CIKM*, 2015, pp. 1211–1220.

[18] N. T. Hai, “A novel approach for location promotion on location-based social networks,” in *RIVF*, 2015, pp. 53–58.

[19] X. Wang, Y. Zhang, W. Zhang, and X. Lin, “Distance-aware influence maximization in geo-social network,” in *ICDE*, 2016, pp. 1–12.

[20] T.-N. Doan, F. C. T. Chua, and E.-P. Lim, “Mining business competitiveness from user visitation data,” in *SBP*, 2015, pp. 283–289.

[21] L. LováSz, “Review of the book by alexander schrijver: Combinatorial optimization: Polyhedra and efficiency,” in *Oper. Res. Lett.*, 2005.

[22] R. R. Braam, H. F. Moed, and A. F. Van Raan, “Mapping of science: Critical elaboration and new approaches, a case study in agricultural biochemistry,” in *Informetrics*, 1988, pp. 15–28.

[23] H. Gao, J. Tang, and H. Liu, “Exploring social-historical ties on location-based social networks,” in *AAAI*, 2012.

[24] E. Cho, S. A. Myers, and J. Leskovec, “Friendship and mobility: User movement in location-based social networks,” in *KDD*, 2011.

[25] Q. Liu, M. Deng, Y. Shi, and J. Wang, “A density-based spatial clustering algorithm considering both spatial proximity and attribute similarity,” *Computers and Geosciences*, vol. 46, pp. 296–309, 2012.

Paper G

Activity-Driven Influence Maximization in Social Networks

Rohit Kumar, Muhammad Aamir Saleem, Toon Calders, Torben Bach Pedersen

The paper has been Published in
European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases.

Abstract

Interaction networks consist of a static graph with a time-stamped list of edges over which interaction took place. Examples of interaction networks are social networks whose users interact with each other through messages or location-based social networks where people interact by checking in to locations. Previous work on finding influential nodes in such networks mainly concentrate on the static structure imposed by the interactions or are based on fixed models for which parameters are learned using the interactions. In two recent works, however, we proposed an alternative activity data driven approach based on the identification of influence propagation patterns. In the first work, we identify so-called information-channels to model potential pathways for information spread, while the second work exploits how users in a location-based social network check in to locations in order to identify influential locations. To make our algorithms scalable, approximate versions based on sketching techniques from the data streams domain have been developed. Experiments show that in this way it is possible to efficiently find good seed sets for influence propagation in social networks.

1 Introduction

Understanding how information propagates in a network has a broad range of applications like viral marketing [1], epidemiology and outdoor marketing [2]. For example, imagine a computer games company that has budget to hand out samples of their new product to 50 gamers, and want to do so in a way that achieves maximal exposure. In that situation the company would like to target those customers that have maximal influence on social media. For this purpose they monitor interactions between gamers, and learn from these interactions which ones are the most influential. Notice that for the company it is also important that the selected people are not only influential, but that their combined influence should be maximal; selecting 50 highly influential gamers in the same sub-community is likely less effective than targeting less influential users but from different communities. This example is a typical instance of the *Influence maximization problem* [1]. The common ingredients of an influence maximization problem are: a graph in which the nodes represent users of a social network, an information propagation model, and a target number of seed nodes that need to be identified such that they jointly maximize the influence spread in the network under the given propagation model.

Earlier works in this area studied different propagation models, such as linear threshold (LT) or independent cascade (IC) models [3], the complexity of the influence maximization problem under these models, and efficient heuristic algorithms. For instance, Kempe et al. [3] proved that the influence maximization problem under the LT and IC models is NP-hard and they provided a greedy algorithm to select seed sets using maximum marginal gain. As the model was based on Monte Carlo simulations it was not very scalable for large graphs.

A critical issue in the application of influence maximization algorithms is that of selecting the right propagation model. Most of these propagation models rely on parameters such as the influence a user exerts on his/her neighbors. Therefore, a second important line of work deals with learning these parameters based on observations. For instance, in a social network we could observe that user a liking a post is often followed by user b, who is friend of a, liking the same post. In such a case it is plausible that user a has a high influence on user b, and hence that the parameter expressing the influence of a on b should get a high value. The parameter learning problem is hence, based on a record of activities in the network, estimate the most likely parameter setting for explaining the observed propagation. The resulting optimized model can then be used to address the problem of selecting the best seed nodes. Goyal et al. [4] proposed the first such data based approach to find influential users in a social network. They estimate

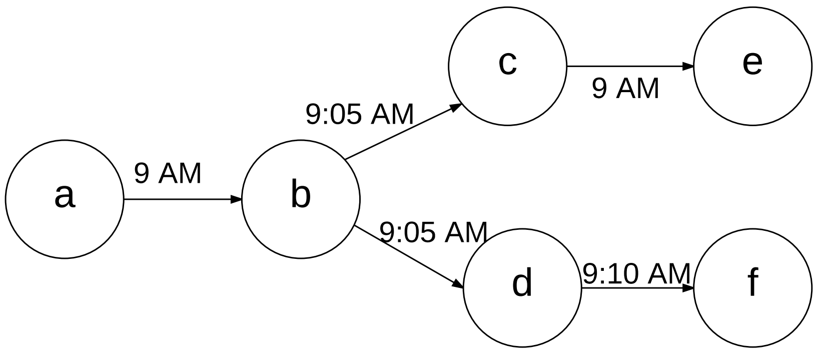


Fig. G.1: Information channels between different nodes in the network. Every node is a user in a social network and the edges represents an interaction between users.

the influence probability of one user on his/her friend by observing all the different activities the friend follows in a given time window divided by total number of activities done by the user.

All these works share one property: they are based on models and if activity data is used, it is only indirectly to estimate model parameters. Recently, however, new, model-independent and purely data-driven methods have emerged. Our two papers, [2] published at WSDM and [5] published at EDBT should be placed in this category of data-based approaches.

2 Data-Driven Information Maximization

In [5] we proposed a new time constrained model to consider real interaction data to identify influence of every node in an interaction network [6]. The central idea in our approach is to mine frequent *information channels* between different nodes and use the presence of an information channel as an indication of possible influence among the nodes. An *information channel*($ic(u, v)$) is a sequence of interactions between nodes u and v forming a path in the network which respects the time order. As such, an information channel represents a potential way information could have flown in the interaction network. An interaction could be bidirectional, for instance a chat or call between two users where information flows in both directions, or unidirectional where information flows from one user to another, for example in an email interaction or a re-tweet.

Figure G.1 illustrates the notion of an information channel. There are interactions from user $a \rightarrow b$ and $c \rightarrow e$ at 9 AM, from $b \rightarrow d$ and $b \rightarrow c$ at 9:05 AM and $d \rightarrow f$ at 9:10 AM. These interactions form an interaction network. There is an information channel $a \rightarrow c$ via the temporal path $a \rightarrow b \rightarrow c$ but there is no information channel from $a \not\rightarrow e$ as there is no time respecting path

Check-ins

loc	Users		
	t=1	t=2	t=3
T_1	b,c,e,f	a,h	f
T_2	a,h	f,g	a
M_1	g	i	d
H_1	–	b,c,d,e	i
H_2	d,i	–	–

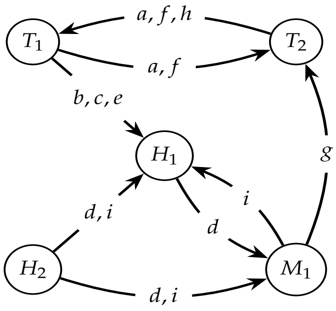


Fig. G.2: Running example of a LBSN [2].Nodes in the graph are the locations visited by users a-h. Edges are the movement of user between locations in a time window.

from a to e. We define the $duration(dur(ic(u,v)))$ of an information channel as the time difference of the first and last interaction on the information channel. For example, the duration of the information channel $a \to b \to c$ is 10 minutes. There could be multiple information channels of different durations between two nodes in a network.The intuition of the information channel notion is that node u could only have sent information to node v if there exists a time respecting series of interactions connecting these two nodes. Therefore, nodes that can reach many other nodes through information channels are more likely to influence other nodes than nodes that have information channels to only few nodes. This notion is captured by the *influence reachability set*. The *Influence reachability set (IRS)* $\sigma(u)$ of a node u in a network $G(V,\mathcal{E})$ is defined as the set of all the nodes to which u has an information channel

In [5] we presented a one-pass algorithm to find the IRS of all nodes in an interaction network. We developed a time-window based HyperLogLog sketch [7] to compactly store the IRS of all the nodes and provided a greedy algorithm to do influence maximization.

3 Finding Influential Locations

Outdoor marketing can also benefit from the same data based approach to maximize influence spread [2]. Recently, with the pervasiveness of location-aware devices, social network data is often complemented with geographical information, known as location-based social networks (LBSNs). In [2] we study navigation patterns of users based on LBSN data to determine influence of a location on another location. Using the LBSN data we construct an interaction graph with nodes as locations and the edges representing the users traveling between locations. For example, in Figure G.2 there is an edge from location T_1 to T_2 due to users a and f visiting both locations within one trip.

We define the influence of a location by its capacity to spread its visitors to other locations. The intuition behind this definition is that good locations to seed with messages such as outdoor marketing promotions, are locations from which its visitors go to many other locations thus spreading the message. Thus, location influence indirectly captures the capability of a location to spread a message to other geographical regions. For example, if a company wants to distribute free t-shirts to promote some media campaign in a city, it would get maximum exposure by selecting neighborhoods such that the visitors of these neighborhood spread to maximum other neighborhoods in the city. In [2] we provide an exact on-line algorithm and a more memory-efficient but approximate variant based on the HyperLogLog sketch to maintain a data structure called *Influence Oracle* that allows to greedily find a set of influential locations.

4 Conclusion

In both of our works, through simulation experiments, we have shown that the data driven approach is quite accurate in modeling influence spread in the network. We also used time window based variations of the HyperLogLog sketch as an alternative to capture the influence set of every node in the network enabling us to scale our algorithms to very high data volumes. Currently, we just use the interaction data or activity as a measure to determine influence. Considering other meta data associated with these interactions and activities could improve the influence spread model even further.

References

[1] M. Richardson and P. Domingos, “Mining knowledge-sharing sites for viral marketing,” in *KDD*, 2002, pp. 61–70.

[2] M. A. Saleem, R. Kumar, T. Calders, T. B. Pedersen, and X. Xie, “Location influence in location-based social networks,” in *WSDM*, 2017, pp. 621–630.

[3] D. Kempe, J. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network,” in *KDD*, 2003, pp. 137–146.

[4] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "A data-based approach to social influence maximization," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 73–84, 2011.

[5] R. Kumar and T. Calders, "Information propagation in interaction networks," in *EDBT*, 2017, pp. 270–281.

[6] R. Kumar, M. A. Saleem, T. Calders, X. Xie, and T. B. Pedersen, "Activity-driven influence maximization in social networks," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part III*, 2017, pp. 345–348.

[7] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," *DMTCS Proceedings*, 2008.

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-185-9

AALBORG UNIVERSITY PRESS