Identifying Points of Interest from Crowd-Sourced Trajectory Data

by
Tanzima Azad

Submitted to
Department of Computer Science and Engineering
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka - 1000

September 2018
Dedicated to my loving parents

Author’s Contact

Tanzima Azad
Assistant Programmer
Email: tanzima.azad@bb.org.bd
The thesis titled “Identifying Points of Interest from Crowd-Sourced Trajectory Data”, submitted by Tanzima Azad, Roll No. 0412052048P, Session April 2012, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on September 29, 2018.

Board of Examiners

1. 
Dr. Mohammed Eunus Ali  
Professor  
Department of CSE, BUET, Dhaka - 1000.  
Chairman  
(Supervisor)

2. 
Dr. Md. Mostofa Akbar  
Head and Professor  
Department of CSE, BUET, Dhaka - 1000.  
Member  
(Ex-Officio)

3. 
Dr. M. Kaykobad  
Professor  
Department of CSE, BUET, Dhaka - 1000.  
Member

4. 
Dr. Atif Hasan Rahman  
Assistant Professor  
Department of CSE, BUET, Dhaka - 1000.  
Member

5. 
Dr. Mohammad Nurul Huda  
Professor  
Department of CSE  
United International University, Dhaka - 1212.  
Member  
(External)
Candidate's Declaration

This is hereby declared that the work titled "Identifying Points of Interest from Crowd-Sourced Trajectory Data" is the outcome of research carried out by me under the supervision of Dr. Mohammed Eunus Ali, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka - 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Tanzima Azad
Candidate
Acknowledgment

First and foremost I offer my sincerest gratitude to my supervisor, Dr. Mohammed Eunus Ali, who has supported me throughout my thesis with his patience, motivation, enthusiasm, and immense knowledge. He helped me a lot in every aspect of this work and guided me with proper directions whenever I sought one. I could not have imagined having a better supervisor and mentor for my M.Sc. study and research. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Dr. Md. Mostofa Akbar, Dr. M. Kaykobad, Dr. Atif Hasan Rahman and specially the external member Dr. Mohammad Nurul Huda.

In this regard, I remain ever grateful to my beloved parents, who always exist as sources of inspiration behind every success of mine I have ever made.
Abstract

The high availability of GPS-enabled devices and easy use of smart phones have made it easy to generate and store location histories in trajectory data format. Large amounts of spatio-temporal data pertaining to an individual’s trajectories has given a rise to a variety of geographic information systems, and also brings us opportunities and challenges to determine valuable knowledge from these trajectories. Recent research efforts on trajectory dataset focus on identifying the labels of points of interest (POIs), recommending personalized POIs, deriving popular locations from GPS traces, etc. On the other hand, popular map services such as Google Maps, Bing Maps, etc. facilitate users to navigate different POIs. However, large number of POIs are yet to be covered by these map services. Comprehensive and up-to-date coverage of POIs are challenging because the number of POIs are large and the information related to POIs are always changing. Though some recent efforts have been made to identify POIs from check-in data or automatically identify POIs from images, these methods are costly and not suitable for changing circumstances. Crowdsourcing is a low-cost and efficient way to extract useful information from data acquired by crowd participants or volunteers. We could efficiently find approximate location of a given point from the spatio-textual data collected from crowd sourcing. In this paper, we propose an approach to find an approximate location for a POI. The approach combines information from different sources such as the user’s texted input, movement and direction to find an approximate location for a POI. A large set of experiments on real datasets shows that our approach can perform significantly in terms of identifying a location.
# Table of Contents

*Board of Examiners*  
Candidate’s Declaration  
Acknowledgment  
Abstract

1 Introduction 1  
1.1 Research Challenges .................................................. 4  
1.2 Objectives of the Thesis .................................................. 5  
1.3 Users Motivations to Participate ........................................ 5  
  1.3.1 Games with a purpose ................................................ 5  
  1.3.2 Micro-task markets .................................................... 6  
  1.3.3 Open innovation contests ............................................ 6  
  1.3.4 Social identification ................................................ 6  
1.4 Contributions ............................................................... 7  
1.5 Outline ................................................................. 8  

2 Problem Formulation 9  
2.1 Definitions ............................................................... 9  
  2.1.1 Spatio-textual log .................................................. 9  
  2.1.2 Spatio-textual trajectory .......................................... 9  
  2.1.3 Problem Description ............................................... 10  
  2.1.4 POI Identification Problem ...................................... 10
2.2 Recommendation System Requirements ............................................. 11
   2.2.1 Efficiency ................................................................. 11
   2.2.2 Accuracy ................................................................. 12
   2.2.3 Scalability ............................................................... 12
   2.2.4 On-line Recommendation Capability .................................... 13

3 Related Work .................................................................................. 14
   3.1 Trajectory Mining ................................................................. 14
   3.2 Spatial Crowdsourcing ........................................................... 15
   3.3 Extraction of Points (or Regions) of Interest ............................. 16
   3.4 Mining Location History ....................................................... 17
      3.4.1 Mining location history of individuals’ ................................. 17
      3.4.2 Mining Location Histories of Multiple Users .................... 17
   3.5 Location Recommenders ....................................................... 18
      3.5.1 Real-time location based recommenders ............................ 18
      3.5.2 Location History based Recommenders ............................ 18

4 Our Solution .................................................................................. 20
   4.1 Preliminaries ............................................................................ 20
      4.1.1 Spatial and Spatiotemporal Data Attributes ....................... 20
      4.1.2 Machine Learning for Spatio-temporal Prediction ............... 21
      4.1.3 Techniques for finding facts from Big Data ........................ 22
      4.1.4 Classification ................................................................ 23
         4.1.4.1 Linear classifiers ...................................................... 23
      4.1.5 Naive Bayes classifier ..................................................... 23
      4.1.6 Support Vector Machines ................................................ 24
      4.1.7 Clustering .................................................................... 25
         4.1.7.1 K-means clustering .................................................. 25
         4.1.7.2 DBSCAN clustering ............................................... 27
      4.1.8 String Searching Algorithms .......................................... 27
         4.1.8.1 The Knuth-Morris-Pratt Algorithm ............................ 28
   4.2 System Architecture ............................................................... 29
4.2.1 Identifying individual keywords .............................................. 30
4.2.2 Clustering ............................................................................... 30
4.2.3 Identifying dense regions ......................................................... 30
4.2.4 Calculating Centroids of the regions ......................................... 30
4.2.5 Finding locations of POIs ......................................................... 31
4.3 Identifying individual keywords .................................................. 31
4.3.1 Data Collection ....................................................................... 31
4.3.2 Preprocessing .......................................................................... 35
   4.3.2.1 Data Cleaning .................................................................. 35
   4.3.2.2 Feature Extraction .............................................................. 36
4.4 Clustering ..................................................................................... 37
4.5 Identifying dense regions ............................................................ 39
4.6 Calculating Centroids of the regions ............................................. 43
4.7 Finding locations of POIs ............................................................. 46

5 Algorithms .................................................................................. 48
   5.1 POI Identification ..................................................................... 48
   5.2 Circular Regions ....................................................................... 50
      5.2.1 Bresenham’s Algorithm ...................................................... 50
   5.3 Elliptical Regions ..................................................................... 51
      5.3.1 Midpoint Ellipse Algorithm .............................................. 51
   5.4 Intersection Area ..................................................................... 53
      5.4.1 Sweep Line Algorithm ...................................................... 54
      5.4.2 R*-Tree Algorithm .............................................................. 57

6 Experiments .................................................................................. 59
   6.1 Answer Accuracy ..................................................................... 60
      6.1.1 Offline Version ................................................................. 61
      6.1.2 Travel mode and speed range .......................................... 61
      6.1.3 User and POI direction ..................................................... 62
      6.1.4 Time of location tag ......................................................... 63
      6.1.5 Online Version ................................................................. 64
## List of Figures

1.1 Motivation of the Thesis .................................................. 3
  1.1a Data collection through crowdsourcing ................................. 3
  1.1b Identified region for the POI ......................................... 3

2.1 Problem Formulation ....................................................... 10

4.1 Process of Knowledge Discovery in Databases (KDD) ..................... 22
4.2 Architecture of our System ................................................ 29
4.3 Data Collection .......................................................... 32
4.4 Screenshot of all the data collected in notepad .......................... 35
4.5 All occurrences for a single keyword ...................................... 37
4.6 Cluster ................................................................. 39
4.7 Steps of outlining regions for a single keyword ........................... 42
  4.7a POIs retrieved for a single keyword ................................. 42
  4.7b Circular regions for lower speed points ......................... 42
  4.7c Elliptical regions for higher speed points ...................... 42
  4.7d Elliptical regions for zero speed points ......................... 42
4.8 An example of outlining regions for a single keyword .................. 43
4.9 Steps of reshaping circular and elliptical regions into squares and rectangles ........................... 44
  4.9a Reshape circles into squares ...................................... 44
  4.9b Reshape ellipses into rectangles for higher speed points .......... 44
  4.9c Reshape ellipses into rectangles for zero speed points .......... 44
  4.9d All square and rectangle regions for the POI location identification .......... 44
4.10 Steps of calculating centroids of squares and rectangles ............ 45
List of Tables

1.1 Sample data collected by two users ............................................. 3
4.1 Sensors’ Data Details ............................................................... 33
4.2 Location sources ........................................................................ 34
4.3 Notations used in this text and their definition .............................. 35
6.1 Example of spatial trajectory data collected by crowdsourcing .......... 60
6.2 Accuracy with different travel mode data sets ................................. 62
6.3 Accuracy with different user and POI direction ............................... 63
6.4 Percentage of points per data set .................................................. 64
6.5 Parameter settings for POI identification queries ............................. 66
# List of Algorithms

1. **KMEANS**(\(X, k\)) .......................................................... 26  
2. **KMP-Matcher**(\(T[1..n], P[1..m], f[1..m]\)) .................. 28  
3. **POI Extraction**(\(L\)) .................................................. 49  
4. **Circular Region**\((C_j)\) .............................................. 50  
5. **Draw Circle**\((c, r)\) .................................................... 52  
6. **Find Intersections**\((E)\) ............................................. 54  
7. **Handle Event Point**\((p)\) ............................................ 55  
8. **Find New Event**\((s_l, s_r, p)\) .................................. 56
Chapter 1

Introduction

We often need to go to unfamiliar places and to find our way web map services provide location information through satellite imagery. This can make the journey easier and help locate businesses, venues and private addresses that users are not familiar with. Web map services help us reach our destination in the most efficient way, e.g., driving through the day, navigating public transportation, walking unfamiliar streets. In recent years we have witnessed the increased development of location-based social networking (LBSN) services. LBSNs allow users to explore points of interest (POIs) for better services through sharing check-in experiences and opinions on the POIs they have checked in.

But there are lots of POIs around the world and the location of the points are changing at all times. Pervious efforts for identifying POIs have assigned workers for POI labelling and also considered ‘check-ins’ for POI recommendation. In an effort to get full and up-to-date coverage it is required to assign dedicated resources which is very costly and time consuming. This manual data collection is also prone to response bias and inaccuracy. Crowdsourcing costs significantly less time and money and there is no or very little overhead. We aim to collect trajectory data from crowdsourcing and extract location information to add marker to map services. This can give us a complete and up-to-date coverage of the map services.

At present smartphones are the most essential device and people have become so attached to their mobile phones. Most commuters travel regularly or often repeatedly between locations for atleast 45 – 90 minutes each way. We aim the digitization of web map services by utilizing the idle time
of daily commuters. Almost all the POIs are mapped along the roadside which is easily visible to travellers. We collect spatio-textual data for a POI from the commuters only when it is in the visible range. We then propose an automated system to identify the locations of POIs from the crowdsourced spatio-textual data.

During past years, a lot of users have started recording their outdoor activities with GPS trajectories for various reasons. Examples include, sharing and logging travel and life activities, maintaining and managing sports and multimedia activities etc. Simultaneously some other websites or forums [1], [2], [3], which enable users to establish some geo-related web communications, have appeared on the internet. By submitting their logs to these web sites, forums are able to generate and maintain their trajectories on Web maps. In addition, they can visualize related information from other users life experiences by recording and sharing their logs among each other. For example, a user is able to find some places that is interesting to them from other peoples travel experiences, hence, the user may plan an amusing and efficient journey based on other peoples' experiences.

Other research applications have been geared towards analyzing and mining user’s travel experiences from trajectory databases. These researches contain querying multiple sequence of travelling activities [4], based on travelling sequences some mines similarity between travellers [5], deriving popular locations from GPS traces [6], etc. Mostly, these researches determine a set of Regions of Interest that provides information about the points of specific activities.

Figure 1.1 shows the objective to find location of POI from crowdsourced trajectory data. In Figure 1.1(a) the data collection through crowdsourcing is shown for the given POI from different vehicles and road networks. Figure 1.1(b) shows the identified region for the POI. For example, user A and user B collects the following data for the user defined text/ keyword ‘pizza hut’:

In table 1.1 sample data collected by user A and user B is given. Our goal is to identify the location of the restaurent ‘pizza hut’. This thesis addresses the problem of identifying accurate locations of both familiar and unfamiliar places using crowdsourced trajectory data.

Smartphones have become ubiquitous in the lives of billions. The development of smartphone technology has led to the incorporation of increasingly advanced sensors, opening up a number of
opportunities for data mining applications. In particular, smartphones generate a constant stream of data describing the phone’s acceleration (via an onboard accelerometer) and location (via the Global Positioning System (GPS)) in the form of trajectories (e.g., [7], [8], [9], [10]). Great applications [1], [2], [11], [12] allow users to maintain, keep and share trajectories in the form of GPS logs, and find travel routes, attractive places, etc.
Today’s smartphone not only serves as the key computing and communication mobile device of choice, but it also comes with a rich set of embedded sensors, such as an accelerometer, digital compass, gyroscope, GPS, microphone, camera, etc. Data is collected using smartphones including smartphone sensors’ reading, texts written by users, the network connection log, etc. User may write the name of any location which includes familiar or unfamiliar, popular or unpopular places, billboards, road side advertisements, wall writing etc.

1.1 Research Challenges

The problem of trajectory-based POI identification is challenging for two primary reasons. The first challenge is the imprecision of crowdsourced geo spatial trajectory data. The data collected are not from exact location of the POI. So identifying the location of a POI by investigating features of the trajectory data like speed, distance covered, bearing, time etc. is very difficult. To overcome the speed and distance, we have collected data repeatedly twice, once the user starts typing and the other is when the data is submitted. By collecting starting and ending longitude and latitude for each spatio-textual log we can get a bounding region for the POI which is minimized to an approximate region for the location of the POI.

The second challenge is that the identification queries need to be answered in real time. It is easy to compute the locations of POIs offline and recommend them later when a query comes in. However, as the goal is to build a system capable of executing identification queries in mere seconds on a mobile device, we need to make sure that queries can be answered efficiently even when the results are based on a large spatio-temporal database.

Moreover the trajectory-based POI identification is a new problem and there are no previously published requirements for how a trajectory-based POI identification system should behave. Previous research mines individual or multiple users’ trajectory while we aim to mine location of point of interests(POIs). Furthermore, existing systems tend to be capable of recommending only nearby or popular places, but we aim to locate any point of interest in geometric space. Both of these limitations are addressed by this thesis.
1.2 Objectives of the Thesis

The first objective is to digitize the web map services by automatically identifying locations of POI from crowdsourced spatio-textual trajectory data. From our comprehensive analysis of the trajectory data we identify probable region for the location of unknown and unfamiliar POIs.

Our second objective is to convert the idle time of commuters into productive time. We collect data of their daily trails which is actually the GPS trajectory data. Investigating features of the trajectory data like distance, direction, time, velocity etc. is another objective of our research.

Finally, we formulate and devise an efficient method utilizing fact finding algorithms for investigation of different features and related facts from crowdsourced spatio-textual trajectory data and identify the location of POIs.

1.3 Users Motivations to Participate

Generally, in crowdsourcing platforms, best solvers or a group of solvers are paid for their contribution. On the other hand, a large number of participants do not receive any payment despite their competent efforts. However, in numerous crowdsourcing platforms, participants contribute voluntarily. Innocentive is a platform where very complex scientific challenges are posted. Members compete to solve and best solution is awarded up to US$ 100 000. Another well-known example is the Wikipedia where volunteers upload information with no reward in return. It has become one of the most visited platforms for the information seekers. In some cases, the reliability of the Wikipedia information is considered higher than the information provided by the other highly established encyclopedias.

1.3.1 Games with a purpose

Games with a Purpose (GWAPs) are games played online, ostensibly for fun, but the output of the gameplay is used to solve real-world problems. The first GWAP developed by von Ahn [13], the ESP
Game displays images to two players who each try to guess words that the other player would use to describe the image. The game improves web image searches by generating descriptions of uncaptioned images. Google Inc. has licensed the game, which the company calls Google Image Labeler. The online game Peekaboom (www.peekaboom.org) improves on the data collected by the ESP Game by obtaining precise location information for each object in a given image. More specifically, it identifies which pixels belong to which object in the image.

1.3.2 Micro-task markets

Micro-task markets are systems in which workers complete short jobs in exchange for monetary compensation. To date, the largest and most popular micro-task market is Amazon's Mechanical Turk service (MTurk). MTurk hosts hundreds of thousands of tasks, ranging from image labeling to text transcription to spam filtering. For longer and more complex tasks, researchers might consider using other employment markets, such as oDesk, which tend to attract more specialized workers than those found on MTurk.

1.3.3 Open innovation contests

In addition to micro-task markets, open innovation contests provide yet another platform for conducting crowdsourced work. Unlike markets, in which compensation is given for all work, open innovation contests only compensate top performers. Challenges are posted online, either through company-wide initiatives such as Nokia's Ideas Project, Dell’s Idea Storm, and OpenIDEO or through innovation hubs like Innocentive. On the other hand, in TopCoder, a CS platform which hosts regular contests relevant to design and development, the reputation score of contestants is calculated with a more sophisticated algorithm that takes into account their prior history, their expected performance, as well as their performance as compared to that of other contestants [14].

1.3.4 Social identification

Social motives often play a major role for participating in crowdsourcing platforms. Having a good social image is very important for participants in online communities, who want to be perceived as intelligent, fair, wealthy and “good” in general. In order to trigger these social motives, many social incentive mechanics that act as enablers of social interactions, giving users the chance to showcase
their skills and gain social status in the community can be implemented (e.g. specialized mailing lists, provision of feedback/compliments functionalities, invitations to events, etc.). Yelp members care about presenting a good social image to friends and other Yelp community members by being active and contributing many quality reviews [15].

Games with a Purpose (GWAPs) also referred as Gamification is frequently encountered in successful crowdsourcing platforms and applications. In the mobile app TrashOut, by delivering badges, they try to motivate the crowd to denounce the location of illegal dumping. Also by using points and badges, the Old Weather Project, developed in the crowdsourcing platform Zooinverse, encourages the crowd to retrieve weather data from the Arctic and the world by transcribing ships’ logs. In our case we can offer points, achievement badges to users in popular games if they contribute data to our application. To ensure that the app users want to contribute to our application, our main goal is to reward users with coins, gems or levels in which the user would be interested in.

1.4 Contributions

To the best of our knowledge, we propose the first approach for identifying location of POIs from crowdsourced spatio-textual trajectory data. In summary, the contributions of this thesis are as follows:

- We put forward a new problem of identifying any POI, based on the GPS history data collected from crowdsourcing, so that we can provide more specific recommendation for any location.

- We propose to exploit crowdsourced spatio-temporal trajectories for location filtering, so as to address the improvement of geocoded map markers. We also show how to well incorporate this additional information with the GPS-histories for correcting a map marker that has geocoded incorrectly.

- We analyze our application using a large GPS dataset, which was collected by 11 users over a period of three months in the real world. The number of GPS points is 10,700.
1.5 Outline

The remaining part of the thesis is organized as follows:

In Chapter 2, we describe the terms Spatio-textual log and Spatio-textual trajectory, formulate POI identification problem and some recommendation system requirements.

In Chapter 3, we outline the research work related to this problem. To make it more convenience we split the chapter into few sections as trajectory mining, spatial crowdsourcing, extraction of points of interest, mining location history, mining location histories of multiple users, location recommenders, location history based recommenders.

In Chapter 4, we show the system architecture and propose and explain our POI identification query and description of the specific methods constructed to identify location of POIs.

In Chapter 5, we present our algorithm and also descriptions of the functions used in our algorithm to evaluate POI identification queries and its variant in spatial databases.

In Chapter 6, we present descriptions and visualizations of the datasets used for experimentation, and experimental results using real datasets demonstrating the effectiveness of the methods presented in this thesis.

In Chapter 7, we conclude the thesis with possible directions for future work.
Chapter 2

Problem Formulation

In this chapter, we first describe some terms used including Spatio-textual log \((L)\) and Spatio-textual trajectory \((Traj)\) and define the Point of interest(POI) identification problem. Then we give some properties of our POI identification system.

2.1 Definitions

2.1.1 Spatio-textual log

Basically, a spatio-textual log is a collection of Spatio-textual points \(L = l_1, l_2, \ldots, l_n\). Each spatio-textual point \(l_i \in L\) mainly contains a starting latitude \(l_i.Slat\), a starting longitude \(l_i.Slon\), a starting time-stamp \(l_i.Stime\), an ending latitude \(l_i.Elat\), an ending longitude \(l_i.Elon\), and an ending time-stamp \(l_i.Etime\). Particularly for our problem each point additionally contains a user defined text \(l_i.Txt\), corresponding starting speed \(l_i.Sspd\), ending speed \(l_i.Espd\), starting bearing \(l_i.Sbear\), ending bearing \(l_i.Ebear\). The speed and bearing are taken from GPS. The starting latitude, longitude and time-stamp is recorded when user starts typing and the ending attributes are taken when user submits.

2.1.2 Spatio-textual trajectory

We can sequentially connect these spatio-textual points into a curve based on their time serials until the time interval between consecutive spatio-textual points exceeds a certain threshold \(\Delta T\). Thus trajectory \((Traj)\) is a sequence of spatio-textual points \(Traj = l_1 \rightarrow l_2 \rightarrow \ldots \rightarrow l_n\), where \(l_i \in L, l_{i+1}.T\)
> l_i.T and \( l_{i+1}.T - l_i.T < \Delta T \) (1 \( \leq \) i < n).

### 2.1.3 Problem Description

The primary goal of this thesis is to analyse Spatio-textual trajectories and provide a realistic model for outlining the problem of identifying POI and then to describe an efficient and scalable method for answering POI locations.

To begin, we assume that the following information is available:

- a Spatio-textual log
- Spatio-textual trajectories

Like a traditional search engine, the problem of POI identification is a query-answering problem. However, unlike a traditional search engine where a query consists of a series of words, here a query is a single point of interest which might not be a common place. The goal is then to return the most approximate location of the point.

### 2.1.4 POI Identification Problem

A Spatio-textual log and Spatio-textual trajectories are given to find the location of point of interests users pointed in their trajectory. These locations are the main outputs of our system.

This is a very general definition of the problem, and one interesting ambiguity in its statement is that it does not state which location of the points needed to be identified. By varying the number of
CHAPTER 2. PROBLEM FORMULATION

POIs, we can in fact construct an online and an offline model of the problem. For example, we could perform the searching offline for all the POIs user have mentioned in the Spatio-textual log and save them for future recommendation. Again we can perform the searching online only when any user searches for a specific POI.

Figure 2.1 illustrates the problem proposed in this paper, which is already described in this section. Firstly we introduce the spatio-textual log (Fig. 2.1(a)), which will be used in our GPS data analysis. The second step corresponds to the data modelling technique getting spatio-textual Trajectories (Fig. 2.1(b)), where a method based on coordinates speed will be introduced will be introduced to eliminate noisy data. The Step 3 of our problem is to finally identify the locations of POIs (Fig. 2.1(c)).

2.2 Recommendation System Requirements

In chapter 4 we will describe two particular instances of the trajectory-based POI identification problem that are tackled by this thesis. However, before doing so, we want to first motivate and describe some properties that are desired for a useful POI identification system. The two instances of the identification problems solved by this thesis will incorporate first three requirements and only the online version will incorporate the last requirement. These requirements are:

1. (Efficiency) A POI identification system should take care of all the factors and come up with the most efficient solution for the users.

2. (Accuracy) The location recommended by the system should be accurate.

3. (Scalability) The trajectory-based recommendation system must be scalable in order to handle an arbitrarily large historical trajectory database, as well as any number of simultaneous requests.

4. (On-line Recommendation Capability) Recommendation queries must execute in realtime. However, there is no limitation on the amount of pre-processing time.

2.2.1 Efficiency

One highly desirable requirement for any trajectory-based POI recommendation system is an efficient and reliable solution providing exact location information for its users. It should have
the ability to check possible location of the specific querying POI and give output in the fastest
time. This can make users journey easier and help them locate lots of Point of interests (POIs)
they are not familiar with.

### 2.2.2 Accuracy

Location information provided by the POI identification system should be accurate and should
not have any error. Ambiguities and flaws in recommended information may produce a location
that doesn’t take you to the destination you expect. When the system will become popular by
the increasing uses of the users it should have up-to-the-minute information, such as change of
any business location or opening any new business.

### 2.2.3 Scalability

If a trajectory-based recommendation system were to be put into use, the size of the historical
trajectory database could be expected to grow rapidly as the system became increasingly
popular, and it is important that the system be able to scale. There are two dimensions of
scalability to handle. The first is the number of incoming requests, but this is essentially solved
by scaling the hardware used to process requests and will not be mentioned further in this
thesis. The second is the size of the historical trajectory database. It is important that the time
required to answer a recommendation query grows sub-linearly with respect to the size of the
historical database.

If we consider the offline version it would be possible for queries to be answered in expected
constant time by pre-computing the locations of the POIs using a hash table to store and retrieve
their results. However, for the online version we will start computing the location from the re-
computed results of the raw spatio-temporal trajectory data which will make the searching faster
in real-time.
2.2.4 On-line Recommendation Capability

To be useful in the real world, a trajectory-based recommendation system must be able to execute recommendation queries in real time. As the envisioned use of the system is for people on the move, it is important that queries be satisfied quick enough so that it is possible for a user to act based on the returned set of recommendations. On the other hand, like a normal search engine, we can allow for large amount of pre-processing time and computational resources. It is desirable to minimize the resources required for preprocessing the historical trajectory data. But this is of much less importance than ensuring that queries can be executed very quickly. Even if the choices were made for queries to be executed locally on a mobile device rather than a backend server, any amount of work could be performed prior to loading a processed dataset onto the mobile device.
Chapter 3

Related Work

In this chapter, we discuss the work related to our research problem. We first discuss related works for Trajectory Mining in 3.5.2, methods of spatial crowdsourcing and task assignment algorithms in 3.2 and Extraction of places (or regions) of interest in 3.3. We then categorize other related researches into two parts: Mining Location History and Location Recommenders. Mining Location History exist for both single user and multiple user in the literature. In Section 3.4.1, we discuss existing approaches for mining location history of individuals. In Section 3.4.2, we discuss existing works for mining location history of multiple users. In Section 3.5.1, we discuss the Real-time location based recommenders. In Section 3.5.2, we show how popular locations are recommended from location histories.

3.1 Trajectory Mining

The goal of this thesis is to devise a framework for a spatio-temporal trajectory-aware point of interest identification system. In addition to being built upon research into recommendation systems, the other field closely related to the content of this thesis is trajectory mining. Trajectory mining is a very new field of research. There was a modicum of related research performed in the 1990s, tackling problems such as vehicle classification [16] and trajectory clustering using regression models [17]. However, these research studies tackle the problem of trajectory mining from highly mathematical and statistical stances respectively. Trajectory mining as a topic of data mining is very new and has only been the subject of intensive research in the past several years. Like data mining, research into trajectory mining has tended to focus on the traditional three pillars of clustering, classification and
pattern mining. Regarding trajectory classification, Lee et al. [18] presented the "TraClass" algorithm to classify trajectories. The features for the classifier are discovered by performing a region-based clustering of the trajectories, followed by a trajectory similarity-based clustering step. Among the applications in mind for this direction of research is to classify whether a boat is an oil-tanker, a tugboat, a fishing-boat and so on, and another application is to classify an animal given its historical trajectories. While interesting, this research is not particularly relevant to the problem and methods in this thesis. Giannotti et al. [19] addressed the problem of trajectory pattern mining by using a "region-of-interest" approach to find trajectories moving between regions of interest. A spatial approach to trajectory pattern detection employs a spatial approach to trajectory pattern detection, spatial information is considered in a pre-processing phase that reduces trajectories to a sequence of regions of interest. In their approach, temporal differences between visits matter, but the exact time of a visit does not, and it is not possible for the order of points in trajectories to be swapped. Similarly, spatial regions matter, but not specific locations. [19] is relevant to the research contained in this thesis, but the problem tackled is different and for our purpose suffers from the serious limitation of only considering regions and not specific locations. Finally, Gidofalvi and Pedersen [20] mined long trajectories of moving objects and showed how to identify trips using an SQL-based implementation.

Highly relevant to the problem addressed by this thesis is the research done by Zheng et al. [6] on mining interesting locations and travel sequences from GPS trajectories. Using ideas from the HITS (hypertext induced topic search) model developed by Zheng et al. [6] used a HITS-based inference model to find locations and trajectories that could be recommended. In particular, they treated users as hubs, and locations as authorities, and this is used to compute the interest of each location. A very useful application of this research would be to devise tour plans for cities, as the methods described could determine popular tour routings from GPS trajectory data. However, this method does not allow for queries to be executed of the form "given a name of a location Q, where is this place situated?" which is the main focus of this thesis.

### 3.2 Spatial Crowdsourcing

Crowdsourcing is now becoming a new effective method to handle computer-hard tasks. As many of those tasks contain spatial information (e.g. taking a photo in a location), spatial crowdsourcing also draws attention from both industry and research community [21],[22],[23]. A common constraint of
those spatial tasks is that, they require workers to finish the tasks by traveling to the marked locations specified in the task. Thus, the spatial distance between workers and tasks is treated as the travel cost, which needs to be considered in the general task objective. Task assignment algorithms are then proposed to optimize those objectives [22],[23] and a platform [21] is developed to specifically support these spatial tasks. Recently, some approaches [24],[25],[26],[27] have been proposed to study the task assignment problem. Liu et al. [24] uses a quality-sensitive method to compute the uncertainty of each task and adopts an entropy-like method to select the tasks with maximum uncertainty for the worker. Zheng et al. [25] proposes to maximize the evaluation metric-driven quality improvement in the assignment. Fan et al. [25] models diverse accuracies of workers on tasks and assigns tasks to the workers who have high accuracies in answering the tasks. Some other works [26],[27] leverage machine learning techniques to decide the assigned tasks under different settings. Our work is different from these spatial crowdsourcing tasks. First, our approach does not request workers to travel to specified locations to answer any task. Rather we propose a framework where users will log their locations on the way to their own destinations. Second, the identification and optimization goal is different. These works focus on minimizing the travel cost and how to select the tasks for each user while we don’t require any dedicated workers to collect our data. Different from these studies, we consider the most cost-effective data collection technique and we only focus on identifying the location of a POI rather than to propose any technique for optimizing the task assignment problem.

3.3 Extraction of Points (or Regions) of Interest

In the following are reported some recent works in which the extraction of places (or regions) of interest is a fundamental point. Each one of them explains its own extraction method starting from different types of data. In [28] the authors propose a visual analytic procedure for studying mobility data. Their procedure extracts relevant places from movement data because, for their aim, there is not a predefined set of places (e.g. compartments of a territory division) from which the analyst can select places of interest. In [19], [29], the authors generate regions of interest with the purpose of predicting human movements using mobility pattern mining. The regions of interest are obtained by discretizing the working space in a regular grid with cells of small size. Then, the cells not visited are discarded and, by following a density based principle, the cells conceptually belonging to the same
points are merged. In [30] is proposed an approach based on supervised learning to infer people’s motion models from their GPS logs. The authors, first analyze different features to understand the kind of movement performed by the users (car, bus, bike etc.), and then use clustering algorithms to detect stopping points areas. To estimate the physical location of users from traces of mobile devices associated with access points in a wireless network, the authors in [31] characterize popular regions evaluating access points paths with GPS traces. Finally, in [32] it is proposed an approach capable of uncovering semantically relevant keywords for describing a location. Also in this case the locations correspond to the access points areas.

3.4 Mining Location History

3.4.1 Mining location history of individuals’

Within the past years, a number of researches [33],[34],[35],[36], motivated by the advancement of data collection, have been performed based on location history of individual users from GPS trajectories. The works in [33],[34], detects locations of significant users, [33],[36], predicts movements among users locations and [37] recognizes activities of each user on each location. In contrast to these works, we aim to model crowd-sourced trajectory histories and mine locations of POIs rather than mining user’s individual activity.

3.4.2 Mining Location Histories of Multiple Users

Similar sequences were mined in Gonotti et al. [19] from moving users trajectories and a framework was proposed by Mamoulis et al. [38] to retrieve a maximum number of periodic patterns in spatio-temporal data. History of a drivers destinations were used in MSMLS [35], including with data containing behavior of drivers which was mined from multiple users location histories to determine where a driver might be going as a trip progresses. As opposed to these works, we extend the paradigm of mining multiple users location histories from exploring users trajectory data and understanding locations as well as pointing locations.

Some recent works report that the mining of places (or regions) of interest is a significant issue. Each research shows its own model to extract places containing different types of data. A visual analytic procedure to study the mobility of data is proposed in [28]. There is not a predefined set of places,
their method extracts places relevant to movement of data, from which places of interest are selected by the analyst. [30] proposed a method to infer people’s motion models based on supervised learning from their GPS logs. The authors first understands the transportation of users (car, bus, bike etc.) by analyzing different features and then detects stopping points areas using clustering algorithms. The authors in [31] estimates the physical location of users by tracing mobile devices connected with access points in a wireless zone and extract popular regions evaluating GPS traces associated with point paths. A model is proposed in [32] which is able to extract semantically relevant keywords for describing a location. And in this case the access point areas are also correspond to the locations.

3.5 Location Recommenders

3.5.1 Real-time location based recommenders

Based on a users real-time location [39],[40],[41],[42] typically recommends and provides navigation services with mobile tourist guide systems. Formerly, these types of systems simply provided individual’s closest and nearby locations without considering the actual interest of the individual. Newly, a few researchers [40],[42] aim to refine the provided results with the locations obstructed by the closest buildings. Concurrently [39],[41] recommend the users a more personalized recommendation using their location histories. Opposing these approaches, our aim is to recommend locations of users place of interest around them with the knowledge mined from multiple users location histories.

3.5.2 Location History based Recommenders

Some recommenders, such as Geowhiz [43]] and CityVoyager [44] have given approaches to recommend geospatial locations like shopping-malls or restaurants using multiple user’s location histories. Horozov et al. [43] proposed a magnified collective approach to filter and recommend a restaurant. In [44] authors proposed a method to recommend shops to individual based on their choices extracted by analyzing their past location histories. Some focuses on recommending locations that are hot spots for tourism [6]. A HITS-based model is proposed which analyzes users travel experience and the interest of the location, so that only popular locations are recommended by the experienced users. In contrast to these limited systems which only recommends popular locations, our system can mine location of any POI in geometric space. That is, we can recommend the location of not only the popular places
in the area but also other places people have interest in.

In this thesis, for the first time, we present a new algorithm to analyse spatio-trajectory database and identify location of point of interests. The algorithms returns the most probable region for a point of interest which ensure that the point of interest (POI) is exactly located inside that region.
Chapter 4

Our Solution

In this chapter, we present the steps of our hierarchical exploration model and their explanations. Our proposed model works with multiple steps for processing and analyzing large volume of spatio-temporal data collected from crowdsourcing for progressive identification of location of places. The proposed steps employ supervised and unsupervised learning methods mostly consistent with a regression, statistical classifiers and clustering techniques for predicting social groups and relationships. In this hierarchical exploration model, the module in every step works as an autonomous dataset processor capable of handling the expected input feature vector and produce output independently. Collectively, all the modules work as the complete hierarchical exploration model producing output in different steps. In our work, we have used the probability model for both online and offline versions.

The chapter is organized as follows. Section 4.1, briefly explains some preliminary topics related to our problem and proposed solution. In Section 4.2, we show the system architecture of our developed approach for processing POI identification queries.

4.1 Preliminaries

4.1.1 Spatial and Spatiotemporal Data Attributes

There are three distinct types of data attributes for spatiotemporal data: non-spatiotemporal attributes, spatial attributes and temporal attributes. Non-spatiotemporal attributes are used to characterize non-contextual features of objects, such as name, population and unemployment rate for a
city. They are the same as the attributes used in the data inputs of classical data mining. Spatial attributes are used to define the spatial location (e.g., longitude and latitude), spatial extent (e.g., area, perimeter), shape, as well as elevation defined in a spatial reference frame. Temporal attributes include the timestamp of a spatial object, a raster layer, or a spatial network snapshot, as well as the duration of a process.

4.1.2 Machine Learning for Spatio-temporal Prediction

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search and a vastly improved understanding of the human genome. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning “signal” or “feedback” available to a learning system. These are,

**Supervised learning:** The computer is presented with example inputs and their desired outputs, given by a “teacher”, and the goal is to learn a general rule that maps inputs to outputs.

**Unsupervised learning:** No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end.

Between supervised and unsupervised learning is semi-supervised learning, where the teacher gives an incomplete training signal: a training set with some (often many) of the target outputs missing. Transduction is a special case of this principle where the entire set of problem instances is known at learning time, except the missing part of the targets.
4.1.3 Techniques for finding facts from Big Data

The process of finding and predicting facts or data mining is the computational process of discovering patterns in large data sets ("big data") involving methods at the intersection of artificial intelligence, machine learning, statistics and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Aside from the raw analysis step, it involves database and data management aspects, data pre-processing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization and online updating.

The actual data mining task is the automatic or semi-automatic analysis of large quantities of data to extract previously unknown, interesting patterns such as groups of data records (cluster analysis), unusual records (anomaly detection) and dependencies (association rule mining). These patterns can then be seen as a kind of summary of the input data, and may be used in further analysis or, for example, in machine learning and predictive analytics. Neither the data collection, data preparation and preprocessing, nor result interpretation, reporting and visualization is part of the data mining step, but do belong to the overall knowledge discovery in databases (KDD) process as additional steps.

Data mining involves six common classes of tasks. They are: Anomaly detection, Association rule learning, Clustering, Classification, Regression and Summarization. In our work, we have mainly used Classification and Clustering for finding different travel modes and multiple branches of individual POIs.

Figure 4.1: Process of Knowledge Discovery in Databases (KDD)
4.1.4 Classification

In machine learning and statistics, classification is the problem of identifying which of a set of categories a new observation belongs, on the basis of a training set of data containing observations or instances whose category membership is known. In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

4.1.4.1 Linear classifiers

A large number of algorithms for classification can be phrased in terms of a linear function that assigns a score to each possible category $k$ by combining the feature vector of an instance with a vector of weights, using a dot product. The predicted category is the one with the highest score. This type of score function is known as a linear predictor function and has the following general form:

\[
    \text{score}(X_i, k) = \beta_k \cdot X_i
\]

where $X_i$ is the feature vector for instance $i$, $\beta_k$ is the vector of weights corresponding to category $k$, and score $(X_i, k)$ is the score associated with assigning instance $i$ to category $k$. In discrete choice theory, where instances represent people and categories represent choices, the score is considered the utility associated with person $i$ choosing category $k$.

Algorithms with this basic setup are known as linear classifiers. What distinguishes them is the procedure for determining (training) the optimal weights/coefficients and the way that the score is interpreted.

4.1.5 Naive Bayes classifier

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes’ theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number
of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $X = (x_1, ..., x_n)$ representing some $n$ features (independent variables), it assigns to this instance probabilities

$$p(C_k|x_1, ..., x_n)$$ (4.2)

for each of $K$ possible outcomes or classes.

### 4.1.6 Support Vector Machines

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. When data is not labeled, a supervised learning is not possible, and an unsupervised learning is required, that would find natural clustering of the data to groups, and map new data to these formed groups. The clustering algorithm which provides an improvement to the support vector machines is
called support vector clustering highly used in industrial applications either when data is not labeled or when only some data is labeled as a preprocessing for a classification pass.

4.1.7 Clustering

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some predesignated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Other methods are based on estimated density and graph connectivity. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It will often be necessary to modify data preprocessing and model parameters until the result achieves the desired properties.

Here we have discussed some of the popular clustering algorithms used in our work.

4.1.7.1 K-means clustering

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean,
Algorithm 1: KMEANS($X, k$)

1: $X \leftarrow \{ x_1, x_2, ..., x_n \}$
2: $k \leftarrow$ (number of clusters)
3: $MaxIters \leftarrow$ (limit of iterations)
4: $S \leftarrow$ some initial candidate solution
5: for each $c_i \in C$ do
6: $c_i \leftarrow e_j \in E$ (e.g. random selection)
7: end for
8: for each $e_i \in E$ do
9: $l(e_i) \leftarrow \text{argminDistance}(e_i, e_j) j \in 1...k$
10: end for
11: changed $\leftarrow$ false
12: iter $\leftarrow$ 0
13: repeat
14: for each $c_i \in C$ do
15: UpdateCluster($c_i$)
16: end for
17: for each $e_i \in E$ do
18: $\text{minDist} \leftarrow \text{argminDistance}(e_i, e_j) j \in 1...k$
19: if $\text{minDist} \neq l(e_i)$ then
20: $l(e_i) \leftarrow \text{minDist}$
21: changed $\leftarrow$ true
22: end if
23: end for
24: until changed = true and iter $\leq$ maxIters
25: RETURN
serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

4.1.7.2 DBSCAN clustering

DBSCAN is a density based clustering algorithm, where the number of clusters are decided depending on the data provided. This is unlike k-means clustering, a method for clustering with predefined ‘K’, the number of clusters. The main reason why we recognize the clusters is that within each cluster we have a typical density of points which is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of the clusters. Again, this is unlike k-means clustering where all the points are assumed to be belonging to some cluster.

4.1.8 String Searching Algorithms

String searching is an important component of many problems, including text editing, data retrieval, and symbol manipulation. Despite the use of indices for searching large amounts of text, string searching may help in an information retrieval system. For example, it may be used for filtering of potential matches or for searching retrieval terms that will be highlighted in the output. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

The fundamental string searching (matching) problem is defined as follows: given two strings - a text and a pattern, determine whether the pattern appears in the text. The problem is also known as “the needle in a haystack problem”.

**The Problem:** Given a piece of text, find if a smaller string occurs in it. Let \( T[1..n] \) be an array
which holds the text. Call the smaller string to be searched pattern: \( p[1..m] \).

### 4.1.8.1 The Knuth-Morris-Pratt Algorithm

**Algorithm 2: KMP-Matcher**

\[
\begin{align*}
1: & \quad q = 0 \quad \text{(number of characters matched)} \\
2: & \quad \text{for each } i \in n \text{ do} \\
3: & \quad \quad \text{if } P[q + 1] = T[i] \text{ then} \\
4: & \quad \quad \quad q = q + 1 \\
5: & \quad \quad \text{else} \\
6: & \quad \quad \quad \text{if } q > 0 \text{ then} \\
7: & \quad \quad \quad \quad q = f[q] \\
8: & \quad \quad \quad \quad \text{goto line 2} \\
9: & \quad \quad \text{end if} \\
10: & \quad \text{end if} \\
11: & \quad \text{end for}
\end{align*}
\]

The classic Knuth, Morris, and Pratt (1977) algorithm, discovered in 1970, is the first algorithm for which the constant factor in the linear term, in the worst case, does not depend on the length of the pattern.

The KMP matching algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to \( O(n) \). The basic idea behind KMPs algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text of next window. We take advantage of this information to avoid matching the characters that we know will anyway match.

**Computing \( f \):** \( f[i] \) denotes the longest proper suffix of \( P[1..i] \) which is a prefix of \( P[1..n] \).

**Proper prefix:** All the characters in a string, with one or more cut off the end. “S”, “Sn”, “Sna”, and “Snap” are all the proper prefixes of “Snape”.

Proper suffix: All the characters in a string, with one or more cut off the beginning. “agrid”, “grid”, “rid”, “id”, and “d” are all proper suffixes of “Hagrid”.

4.2 System Architecture

Figure 4.2: Architecture of our System

Figure 4.2 shows our systems’ architecture, which is comprised of the following five parts; Identifying individual keywords, Clustering, Identifying dense regions, Calculating Centroids of the regions and Finding locations of POIs. For online version we will first check our database server for any previously stored location recommendation. If any previously known location is not found then these steps will be performed in real time and for offline version these steps could be performed off-line and location of POIs could be saved for future recommendations.
4.2.1 Identifying individual keywords

The GPS trajectory (\(Traj\)) contains all the POIs information. First, we need to find the keywords from User Defined Text (\(p_i.Txt\)) of GPS logs. All the distinct \(p_i.Txt\) will be considered as a keyword. For each keyword, we then need to find all the POIs containing the keyword in \(p_i.Txt\). The set of all the GPS Logs containing the keyword is called individual POIs.

4.2.2 Clustering

Popular businesses increasingly feel the need to expand their reach into new markets - both domestically and internationally from a very early age. Individual POIs will contain all the points of the keyword including all the branches. It is important to cluster the individual POIs to get smaller regions for each branch. Using scikit-learns implementation of the DBSCAN algorithm we will cluster the data. DBSCAN clusters a spatial data set based on two parameters: each point’s physical distance and a minimum cluster size. This works much better for spatial lat-long data.

4.2.3 Identifying dense regions

After clustering the points with individual keywords we will consider regions around each point of the clusters in circular and elliptical form. We will decide the shape of the regions by the speed of the points collected by users. We are considering the speed of the points because our idea of collecting data is to tag a location only when the location is visible to a user. So for lower speed points it is obvious that the point is closer to the actual location and with higher speeds it indicates that the user must have left the actual location behind them. We will consider circular regions around the points having lower speeds and elliptical regions around the points having higher speeds.

4.2.4 Calculating Centroids of the regions

To calculate the centroids and the intersection areas we first need to reform the shapes of the regions. It is not possible to find intersection area of \(n\) ellipses and circles. So, first we will reform the shapes of the regions. It is possible to find out the intersection of \(n\) rectangles. So we reform the ellipses into rectangles and circles into squares and then compute the intersection area. Before we can assign probabilities to the intersected areas we need to calculate the centroids of the squares and the rectangles.
Then we will compute the average centroid to assign the highest probability nearer to the average centroid.

4.2.5 Finding locations of POIs

We will use a sweep line algorithm to find the intersected areas of the shapes. After getting all the intersection areas of the rectangles we will assign probabilities to the computed intersected rectangles. We will give higher probabilities to the rectangles at the average centroid and rectangles far from the average centroid of the intersected rectangles we get lower probabilities. The highest probable rectangles will be saved tagged with the keyword and whenever any user searches for the location of a POI in our system it will recommend the areas saved with the keyword.

4.3 Identifying individual keywords

This is the first step of our architecture, which mainly deals with the raw trajectory dataset. So, procedures in this step focus on collecting and preparing the data for using in the subsequent steps, rather than finding major facts. Here, the objective is to find simpler facts using statistical classifiers from the raw trajectory data and integrating them to the knowledge base. We perform the basic analysis of our raw data and obtain information of the individual POIs.

4.3.1 Data Collection

Data collection is the process of using the smartphone to record data. These data can be the smartphone sensors’ reading, texts written by user’s phone, the network connection log from certain APs or Cellular towers, etc. The users’ written text means name of any location which includes familiar or unfamiliar, popular or unpopular places, billboards, road side advertisements, wall writing etc. The data collection screen of our andorid apps is shown in figure 4.3.

The connection log provides the information in the time and space perspective and is helpful in localization and other related scenarios. Smartphone sensors are classified into four categories: motion sensors, environmental sensors, position sensors and connection sensors.

- **Motion Sensors**: These sensors measure acceleration forces and rotational forces along three
CHAPTER 4. OUR SOLUTION

Figure 4.3: Data Collection

axes of the phone’s coordinates. This category includes accelerometers, gravity sensors, gyroscopes and rotational vector sensors.

- **Environmental Sensors:** These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination and humidity. This category includes barometers, photometers and thermometers.

- **Position Sensors:** These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

- **Connection Sensors:** These sensors provide the solution for smartphones to connect and interact with other devices with various protocols. This category includes Bluetooth, GPS sensors, Wireless sensors, standard cellular connection modulars.

A summary of the smartphone sensors and their sensing data is shown in Table 4.1. The data range and resolution are different from phone to phone. Some have very high resolution sensors and cost more. Some others return a estimated result indicating the level and the exact number (e.g. some
proximity sensor returns binary results indicating far or near to user’s face).

### Table 4.1: Sensors’ Data Details

<table>
<thead>
<tr>
<th>Sensor Name</th>
<th>Data Collected</th>
<th>Dimensions</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>Acceleration</td>
<td>x, y, z</td>
<td>g-force</td>
</tr>
<tr>
<td>Gravity Sensor</td>
<td>Gravity</td>
<td>x, y, z</td>
<td>m/s²</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>Rotation Rate</td>
<td>x, y, z, x calibration, y calibration, z calibration</td>
<td>rad/s</td>
</tr>
<tr>
<td>Magnetometer</td>
<td>Magnetic Field</td>
<td>x, y, z, x calibration, y calibration, z calibration</td>
<td>µ</td>
</tr>
<tr>
<td>Barometer</td>
<td>Ambient Air Pressure</td>
<td>1</td>
<td>hPa</td>
</tr>
<tr>
<td>Rotation Sensor</td>
<td>Rotation Degree (y axis pointing to magnetic north as the default)</td>
<td>Azimuth: Rotation around z Axis, Pitch: Rotation around x Axis, Roll: Rotation around y Axis</td>
<td>Degree</td>
</tr>
<tr>
<td>Proximity Sensor</td>
<td>Relative distance from an object to the view screen of a device</td>
<td>1</td>
<td>cm</td>
</tr>
<tr>
<td>Light Sensor</td>
<td>the ambient light level</td>
<td>1</td>
<td>lx</td>
</tr>
<tr>
<td>Humidity Sensor</td>
<td>the relative ambient humidity</td>
<td>1</td>
<td>Percentage</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>Ambient temperature</td>
<td>1</td>
<td>Celcuis</td>
</tr>
<tr>
<td>GPS Sensor</td>
<td>Geographical description of current location and estimated speed</td>
<td>Latitude, Longitude, Speed</td>
<td>Degree, m/s</td>
</tr>
</tbody>
</table>

As shown in Table 4.2, multiple location sensing sources exist for mobile smart devices, including GPS, cell towers, and WiFi. Location sensing based on cell towers and WiFi typically calculate velocity using consecutive location points. On the other hand, GPS-based location sensing can directly determine velocity. Because Bluetooth is typically used in indoor location systems, we do not consider it.
Table 4.2: Location sources

<table>
<thead>
<tr>
<th>Location</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>Highly accurate, directly provides velocity, only functions outdoors, quickly consumes battery power, and does not return the current location as quickly as some users may desire.</td>
</tr>
<tr>
<td>Cell towers</td>
<td>Works indoors and outdoors, quickly responds, uses less battery power than GPS, and is accurate up to 100 to 300 m, depending on the service provider. Cannot directly provide velocity.</td>
</tr>
<tr>
<td>WiFi</td>
<td>Works indoors and outdoors, quickly responds, uses less battery power than GPS, and is accurate up to 30 to 200 m, depending on the service provider. Cannot directly provide velocity.</td>
</tr>
</tbody>
</table>

Given the low accuracy of cell towers and WiFi-based location sensing, neither of which can directly determine velocity, we rely on GPS to determine velocity in this study.

In data capturing, the main question to answer is how to collect data and what to collect. We record measurements from the acceleration sensor and velocity, latitude, and longitude readings from GPS. The timestamp and a text from the user are also recorded. Additionally we collect bearing information of the user’s mobile phone. However, the raw data collected from such sensors often accompanies with various noises, due to a number of reasons. So, before using these data preprocesing is required to get useful information. A smartphone application was developed for the purpose of data collection. The application stores the data coming from smartphone sensors including GPS, Accelerometer, and Magnetometer. Carrying these sensor enabled devices with the application installed on it, 11 users recorded their outdoor movements with GPS logs from Jun 2015 to Nov 2015. There were two female and nine male participants, all ranging in age from 25 - 35. A significant number of POIs must be visited by at least two users otherwise they will be useful only as an individual information in any place. All points of this dataset were collected from Dhaka, Bangladesh. The workers voluntarily logged their outdoor movements as they are friends, colleagues, family members. The total number of GPS points reached 10700. The screenshot of all the data collected in a notepad is shown in figure 4.4.
CHAPTER 4. OUR SOLUTION

Figure 4.4: Screenshot of all the data collected in notepad

4.3.2 Preprocessing

Table 4.3: Notations used in this text and their definition

<table>
<thead>
<tr>
<th>Notation/Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>A GPS log which is a collection of GPS points</td>
</tr>
<tr>
<td>$l_i$</td>
<td>GPS point</td>
</tr>
<tr>
<td>$l_i.S.lat$</td>
<td>Starting latitude of a GPS point</td>
</tr>
<tr>
<td>$l_i.S.lon$</td>
<td>Starting longitude of a GPS point</td>
</tr>
<tr>
<td>$l_i.E.lat$</td>
<td>Ending latitude of a GPS point</td>
</tr>
<tr>
<td>$l_i.E.lon$</td>
<td>Ending longitude of a GPS point</td>
</tr>
<tr>
<td>$l_i.Txt$</td>
<td>User defined text of a GPS point</td>
</tr>
</tbody>
</table>

4.3.2.1 Data Cleaning

For a better performance, two data preprocessing techniques were employed in this study. First, we removed the duplicate data in the dataset as some GPS points were recorded more than once due to recording errors on the GPS device. Second, according to common sense, we removed some outlier trajectories, which were deemed abnormal. For instance, if the speed of a GPS Log ($L$) is greater than $40 \, \text{m/s}$ than the vehicle is moving at a very high speed which means the actual location is really far from the logged location. We identified them as abnormal trajectories and removed them from the dataset.
4.3.2.2 Feature Extraction

Feature engineering plays an important role in recommending POI locations. Feature engineering is the process of using domain knowledge to create features ensuring machine learning methods to work well, to achieve accurate results in the recommendation of location information and to extract a large number of features from the processed GPS trajectories. These features can then be categorized into individual POIs and travel modes of the logs of the dataset.

At first we identify all the different texts \((l_i, \text{Txt})\) of GPS logs from user’s collected data and these texts are marked as keywords. These keywords are mainly the name of the individual POIs which are used as the input query for the POI Identification System. We iterate through all the GPS logs collected. We add the first text \((l_i, \text{Txt})\) into an array. We then match the second text using the KMP algorithm 2. If the second text is matched with any of the text in the array, then it is discarded or else it is added in the array. After identifying name of the individual POIs we start with each POI and find the location for that POI.

Another feature is the travel modes of logs of the dataset using optimized Support Vector Classification (SVC) for further use. SVC is one of the SVM (i.e., machine learning) methods analyzing data and recognizing patterns. Given a set of input-output data pairs \((x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l) (x_i \in X \subseteq \mathbb{R}^m, y_i \in Y \subseteq \mathbb{R}^n, l \text{ is the number of training samples})\) that are randomly and independently generated from an unknown function, SVM estimates the function using the following Equation:

\[ f(x) = w.\Phi(x) + b \]  \hspace{1cm}  \begin{align*} w, x & \in \mathbb{R}^m, b \in \mathbb{R}^n \end{align*} \hspace{1cm} (4.3)

where \(\Phi(x)\) represents the high-dimensional feature spaces that are nonlinearly mapped from the input space \(x\). \(w\) denotes a parameter vector and \(b\) is the threshold. If the interpretation \(y\) only takes category values, i.e., 1 and +1, it denotes SVC. Otherwise, if the domain of output space \(y\) contains continuous real values, the learning problem then refers to Support Vector Regression (SVR).

The travel modes of walking, bicycle, subway, bus, and car are recognized in this model. We have considered all the modes except for the subway and take up bicycle as the rickshaw mode. Actually these information is used in further analysis and also in experimental results to produce different
datasets of different travel modes. All the occurrences for a single keyword including all travel modes is shown in figure 4.5.

4.4 Clustering

Companies increasingly feel the need to expand their reach into new markets-both domestically and internationally - from a very early age and over the past years we have seen chain stores are expanding their branches at a higher rate as they are always looking to improve their position and strengthen their brand identity in the marketplace. A chain store is one of a group of stores engaged in the same kind of business in different locations and under the same ownership and management. Companies use geographic information systems (GIS) to collect geographic and demographic data to determine where to build new outlets. For chains, using GIS and other data-centric services follows simple logic: It helps the company save money, and prevents them from losing money by opening branches that will just underperform later. As shop owners are opening numerous chains everyday in different places it is important to identify these locations of places individually from one another. The user defined text \((l_i.Txt)\) of GPS log can be similar for two different branches of the same restaurant. If \(l_i.Txt\) is same for two different branches than identifying individual POIs in 4.3.2.2 will identify it as an individual POI. To identify the location of these two branches accurately we will cluster the individual POIs for multiples branches. To choose any clustering algorithm the requirements that are needed:
• Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases. In our case we don’t know the number of cluster in advance.

• Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.

• Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

From the various clustering algorithms, we have selected DBSCAN [45] algorithm for clustering the branches. DBSCAN is a well-known data clustering algorithm that is commonly used in data mining and machine learning. Based on a set of points, it groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions. It requires only one input parameter and supports the user in determining an appropriate value for it. It discovers clusters of arbitrary shape. Finally, DBSCAN is efficient even for large spatial databases. The key idea of the algorithm is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, i.e. the density in the neighborhood has to exceed some threshold. The shape of a neighborhood is determined by the choice of a distance function for two points p and q, denoted by $\text{dist}(p,q)$. We have used the Euclidean distance and considered any two points into two different cluster if they have distance equal to or greater than 1km. We can apply the algorithm to our data set and find clusters based on the minimum distance among the places that the users have logged.

Figure 4.6 shows the implementation of the clustering algorithm with classified sets of the coordinates. In the Figure we gwt three clusters $C_1$, $C_2$ and $C_3$ by applying the clustering algorithm in sub section 4.1.7.2. Points $p_1$ and $p_2$ are in one cluster because the distance calculated from $\text{dist}(p_1, p_2)$ is less than the threshold value for a single cluster. Points ($p_1$ and $p_9$) and ($p_{11}$ and $p_{19}$) are in different clusters because the result of the function $\text{dist}(p_1, p_9)$ and $\text{dist}(p_{11}, p_{19})$ are greater than the threshold value for a single cluster.
4.5 Identifying dense regions

The key idea of our search region refinement techniques is based on circular and elliptical properties. A smaller search region increases the probability of identifying the accurate location of POIs, avoids unnecessary probability computation, and reduces I/O access and computational overhead significantly. We present a novel technique to refine search region using multiple circles and ellipses for different travelling speeds and for having different user defined constraints. Using the following steps, we present the circular search region refinement technique for points having lower speeds (traveling mode: walk and rickshaw) and the elliptical search region refinement technique for points having higher speeds (travelling mode: car and bus). Using the steps we consider circular regions for lower speed points and elliptical regions for higher speed points and zero speed points. We are considering the speed of the points because our idea of collecting data is to tag a location only when the location is visible to a user. Finally our refined search regions consists of union of the multiple circles and ellipses where each circle and ellipse corresponds to one POI logged by users. Based on these refinement techniques, we develop our algorithm to process POI location identification queries in Chapter 5.

- **Lower Speed Points:** As we have two points for each keyword so the center of the lower speed points will be the midpoint of the two coordinates and the radius of the circle will be the distance of the two points.

- **Higher Speed Points:** For higher speed points the radius of the ellipse along the road side
will be the distance of the two points and the other radius will be the half of the distance.

• **Zero Speed Points:** For the points having zero speed we will consider elliptical region covering both side of the roads.

As we tag a location only when the location is visible to the user, so the location must be within visible range of human eyesight. We are also considering speed of vehicles because speed refers to the rate at which an object covers distance. A fast-moving object has a high speed and covers a relatively large distance in a short amount of time. A slow-moving object has a low speed and it covers a relatively small amount of distance in the same amount of time. An object with no movement at all has a zero speed.

The Earth’s surface curve is out of sight at a distance of 3.1 miles or 5 kilometers. So any object further than this would be below the horizon. Again obstructions such as trees, buildings and other light sources limit human vision. Again, travel times of motor vehicles may vary because of different physical factors (curvature, grade, sight distance, frequency of intersections, and roadside development), different traffic factors (volume of traffic, percent of thorough traffic, turning traffic, proportion of trucks, parking conditions and volume of pedestrians), and different environmental factors (section of the country, driver characteristics, weather, season, visibility, enforcement practices, speed limits and other traffic controls).

The speed characteristics at any one location may change from time to time because of the effect of one or more factors such as changes in traffic volumes, weather, or visibility. Speeds may also change as the result of changes or a combination of changes in speed limits, parking regulations, enforcement, or other traffic control measures.

**Lower Speed Points:** We have two spatial data points \((l_{i}.S_{lat}, l_{i}.S_{lon})\) and \((l_{i}.E_{lat}, l_{i}.E_{lon})\) corresponding to each keyword \(l_{i}.Txt\). To compute the center of a circle we first need to find the midpoint of these two points. The midpoint is the latitude and longitude of half-way point along a great circle path between the two points. To calculate the midpoint the equations are given below:

\[
B_{x} = cos(l_{i}.E_{lat}) * cos(l_{i}.E_{lon} - l_{i}.S_{lon})
\] (4.4)
### CHAPTER 4. OUR SOLUTION

\[ B_y = \cos(l_i.E_{lat}) \ast \sin(l_i.E_{lon} - l_i.S_{lon}) \]  
(4.5)

\[ l_i.M_{lat} = \text{atan2} \left( \sin(l_i.S_{lat}) + \sin(l_i.E_{lat}), \sqrt{\cos(l_i.E_{lat}) + B_x} \right)^2 + B_y^2 \]  
(4.6)

\[ l_i.M_{lon} = l_i.S_{lon} + \text{atan2} \left( B_y, \cos(l_i.S_{lat}) + B_x \right) \]  
(4.7)

Using the equations 4.6 and 4.7 we calculate the center for the circles as \((l_i.M_{lat}, l_i.M_{lon})\). Now we need to compute the distance between the two points for the radius of the circle. We use the 'haversine' formula to calculate the great-circle distance between two points that is, the shortest distance over the earth’s surface.

\[ a = \sin^2 \left( \frac{l_i.E_{lat} - l_i.S_{lat}}{2} \right) + \cos(l_i.S_{lat}) \ast \cos(l_i.E_{lat}) \ast \sin^2 \left( \frac{l_i.E_{lon} - l_i.S_{lon}}{2} \right) \]  
(4.8)

\[ c = 2 \ast \text{atan2} \left( \sqrt{a}, \sqrt{1 - a} \right) \]  
(4.9)

\[ d = R \ast c \]  
(4.10)

where, \( R \) is earth’s radius (mean radius = 6,371km). Using the formula in equation 4.10 we can calculate the distance between \((l_i.S_{lat}, l_i.S_{lon})\) and \((l_i.E_{lat}, l_i.E_{lon})\) which will be the radius for the circle.

In Figure 4.7(a), \(p_1, p_2, \ldots, p_{18}\) are the occurrences for a single keyword. The red dots represent the lower speed points which are points having lower speed of the user when collected. The green dots represent the higher speed points. The dots in blue represent the points which have zero speeds, these points are logged while walking or vehicles stuck in traffic jam. In Figure 4.7(b) circular regions are outlined with the lower speed points. For the circle \(C_1\), the center is \(m_1\) which is the midpoint of \(p_3\) and \(p_4\) and the radius \( r_1 = \text{dist}(p_3, p_4) \). Similarly the center for circles \(C_2, C_3, C_4\) are \(m_2, m_3, m_4\) respectively. The radius \( r_2 = \text{dist}(p_7, p_8) \), \( r_3 = \text{dist}(p_{11}, p_{12}) \) and \( r_4 = \text{dist}(p_{13}, p_{14}) \). In Figure 4.7(c) Elliptical regions are outlined with higher speed points. The major axis of ellipse \(E_1\) is \(a_1 = \text{dist}(p_5, p_6)\) and minor axis is \(b_1 = \text{dist}(p_5, p_6)/2\). The major axis of ellipse \(E_2\) is \(a_2 = \text{dist}(p_9, p_{10})\) and minor axis is \(b_2 = \text{dist}(p_9, p_{10})/2\). Figure 4.7(d) shows the elliptical regions for zero speed points.
CHAPTER 4. OUR SOLUTION

Figure 4.7: Steps of outlining regions for a single keyword
Figure 4.8 shows all the circular and elliptical search regions for the points retrieved for a single keyword. We will now search inside these regions for the location of the keyword.

### 4.6 Calculating Centroids of the regions

In this section we will calculate the centroids and intersection areas of the circular and elliptical regions outlined in the section 4.5. As it is not possible to find the intersection area of \( n \) ellipses and circles, we need to first reshape the circles into squares and ellipses into rectangles.

Figure 4.9 shows the steps of transforming the circular and elliptical regions. To transform the circle into a square we simply draw a square with the circle inscribed in it where each side length of the square is equal to the diameter of the circle. In Figure 4.9(a) the side of square \( S_1 \) is \( a = d \), diameter of circle \( C_1 \). So the radius of the circle is exactly half the length of a side of the square. If the center of the circle is \((l_i.M_{lat}, l_i.M_{lon})\) and the radius of the circle is \( r \), then the corners of the square are located at the following points:

\[
\text{upper left corner} = l_i.M_{lat} - r, l_i.M_{lon} + r
\]

\[
\text{upper right corner} = l_i.M_{lat} + r, l_i.M_{lon} + r
\]
a) Reshape circles into squares

b) Reshape ellipses into rectangles for higher speed points

c) Reshape ellipses into rectangles for zero speed points

d) All square and rectangle regions for the POI location identification

Figure 4.9: Steps of reshaping circular and elliptical regions into squares and rectangles
\begin{equation}
lower \text{right corner} = l_1.M_{lat} + r, l_1.M_{lon} - r
\end{equation}

\begin{equation}
lower \text{left corner} = l_1.M_{lat} - r, l_1.M_{lon} - r
\end{equation}

Figure 4.10: Steps of calculating centroids of squares and rectangles

To reshape the ellipses into rectangles we draw a rectangle with the ellipse inscribed in it where length of the rectangle is equal to the major axis of the ellipse and width of the rectangle is equal to the minor axis of the ellipses. We will first take the minor axis and draw two parallel lines touching the major axis coordinates and the two lines parallel to major axis touching the minor axis coordinates. In Figure 4.9(b) length of rectangle $R_1$ is $x_1 = a_1$, major axis of ellipse $E_1$ and width $y_1 = b_1$, minor axis of the same ellipse.

Now we need to find the centroids of the squares and rectangles. Square is a shape whose all dimensions are same. Centroid of square lies where, diagonals intersects each other. As shown in the Figure 4.10(a). Centroid of rectangle lies at intersection of two diagonals. Diagonals intersect at width $(b/2)$ from reference $x - axis$ and at height $(h/2)$ from reference $y - axis$, as shown in Figure 4.10(b).
To assign probability we need to calculate average centroid of all the centroids of squares and rectangles. Square is a special kind of rectangle, it is one where all the sides have the same length. If $A_i$ is the area of rectangle $i$, and $C_i$ is the centroid of rectangle $i$, then the centroid of all the rectangles taken together is just:

$$\frac{\sum_{i=1}^{n} A_i * C_i}{\sum_{i=1}^{n} A_i}$$  \hspace{1cm} (4.15)

### 4.7 Finding locations of POIs

As we have the average centroid point, so we will now assign probabilities to the intersected rectangles. Rectangles nearer to the centroid will be given higher probabilities and rectangles that are far from the centroid will gradually get lower probabilities. After assigning probabilities we will save the highest probable rectangles for future recommendation tagged with the keyword.

In Figure 4.11(a), all the centroids are shown. For $p_1$ and $p_2$ points the corresponding centroid is $C_8$. Centroid $C_1$ corresponds to points $p_3$ and $p_4$. Points $p_5$ and $p_6$ have centroid $C_5$. For $p_7$ and $p_8$ points the corresponding centroid is $C_2$. Centroid $C_6$ corresponds to points $p_9$ and $p_{10}$. Points $p_{11}$ and $p_{12}$ have centroid $C_3$. Points having the centroid $C_4$ are $p_{13}$ and $p_{14}$. Points $p_{15}$ and $p_{16}$, $p_{17}$ and $p_{18}$ have the centroids $C_7$ and $C_9$ respectively. Figure 4.11(b) shows the average centroid of all the centroids shown in Figure 4.11(a) calculated using Equation 4.15. The average centroid with all the search regions is shown in Figure 4.11(c). We assign heighest probabilities to the regions nearer to the average centroid which is shown in Figure 4.11(d).
a) All the centroids

b) Average Centroid

c) Average centroid with search regions

d) Assigned probabilities according to average centroid

Figure 4.11: Recommendation
Chapter 5

Algorithms

In this chapter we present algorithms for POI identification queries based on our solution described at Chapter 4 and discuss, how we can extend the algorithms for POI identification queries with different types of constraints.

The organization of this chapter is as follows. We present and elaborate the algorithms for POI identification queries in Sections 5.1. In Section 5.2 and 5.3 we discuss algorithms to outline circular and elliptical regions of our proposed algorithm. Section 5.4 shows the algorithms to find the intersections of the search regions to assign probabilities for recommendations.

5.1 POI Identification

The key idea of our algorithm is to retrieve regions nearest to the geometric centroid of all the circular and elliptical regions of the POIs logged by the users. Our algorithm uses KMP algorithm to retrieve all the keywords from the GPS Log($L$). Our algorithm retrieves all the occurrences of each keyword and iterates for each keyword. Finally, it identifies the approximate locations for all the keywords and save it tagged with respective keyword for future recommendations.

Algorithm 3 shows the pseudocode of our approach to evaluate POI identification queries for both Euclidean space and road networks. It takes a collection of GPS points which are GPS logs ($L$) logged
Algorithm 3: $POI\_Extraction(L)$

**Input:** A GPS Log ($L$), a Collection of GPS Points

**Output:** A Set of POIs ($P$) with approximate Locations

1: $Initialize()$
2: $K \leftarrow$ set of all distinct keywords from $L_i.Txt$
3: for each $i \in K$ do
4: $L_{si} \in L \leftarrow SearchString(K_i, L)$
5: $C = Cluster(L_i)$
6: for each $j \in C$ do
7: $C_j^{C} = CircularRegion(C_j)$
8: $C_j^{E} = EllipticalRegion(C_j)$
9: $C_j = ReformShapes(C_j^{C}, C_j^{E})$
10: $AvgCen = CalculateAvgCentroid(C_j)$
11: $IntArea = IntersectionArea(C_j)$
12: $R = AssignProbability(IntArea, AvgCen)$
13: $Add(R)$
14: end for
15: end for
16: Return $R$
by the users as input. The output is a set of POIs (P) tagged with their approximate Locations. As the first step, using function Initialize(), Algorithm 3 initializes K with all the distinct keywords from L_i.Txt. The algorithm runs for each keyword stored in K.

The function SearchString(K_i, L) searches for all the logs containing the keyword K_i using the Algorithm 2 discussed in the subsection 4.1.8 in chapter 4. Function Cluster(L_i) then clusters the logs using the Algorithm 2

### 5.2 Circular Regions

Identification of location of POIs is more accurate with smaller search regions. To refine search regions as discussed in section 4.5 we will have circular regions for lower speed points. We use the algorithm 4 for circular regions which uses the Bresenham’s circle drawing algorithm.

**Algorithm 4: Circular Region(C_j)**

**Input:** A Cluster (C_j), a Collection of GPS Points

**Output:** A Set of circles (C_j) for each POIs in the cluster

1: for each m ∈ C_j do
2: \(\text{CorE} = \text{CircleOrEllipse}(P_{m}^{C})\)
3: if CorE = Circle then
4: \(c = \text{mid}(P_{m}^{C}.S, P_{m}^{C}.E)\)
5: \(r = \text{dist}(P_{m}^{C}.S, P_{m}^{C}.E)\)
6: \(\text{Draw_Circle}(c, r)\)
7: end if
8: end for

#### 5.2.1 Bresenham’s Algorithm

Bresenham’s circle drawing algorithm uses the key feature of circle that it is highly symmetric. So, for whole 360 degree of circle we will divide it in 8-parts each octant of 45 degree shown in Figure 5.1a. In order to that we will use Bresenham’s Circle Algorithm for calculation of the locations of the pixels
5.3 Elliptical Regions

As we have discussed in section 4.5 that smaller search regions give more accurate results while identifying locations of POIs. We will have elliptical search regions for higher speed points as we need extra space to cover regarding that the user may have already crossed the actual location with high speed. We will use Midpoint Ellipse Algorithm discussed in subsection 5.3.1.

5.3.1 Midpoint Ellipse Algorithm

Midpoint ellipse algorithm is modified from Bresenham’s circle algorithm. The advantage of this modified method is that only addition operations are required in the program loops. This leads to simple and fast implementation in all processors. The algorithm uses symmetry of ellipse and midpoint algorithm to implement one quadrant only. We divide the quadrant into two regions and the boundary of two regions is the point at which the curve has a slope of -1. We proceed by taking unit steps in the x direction to the point P(where curve has a slope of -1), then taking unit steps in the y direction and applying midpoint algorithm at every step.
Algorithm 5: \textit{Draw\_Circle}(c, r)

\textbf{Input:} A Center $c$ and A Radius $r$

\textbf{Output:} A Circle with center $c$ and radius $r$

1: \textit{Initialize}()
2: $x = 0$
3: $y = r$
4: $x_{\text{center}} = c.x$
5: $y_{\text{center}} = c.y$
6: $d = 3 - 2 \times r$
7: \textbf{while} $y \geq x$ \textbf{do}
8: \hspace{1em} \textit{Plot\_Circle}(x_{\text{center}}, y_{\text{center}}, x, y)
9: \hspace{1em} $x += 1$
10: \hspace{1em} \textbf{if} $d > 0$ \textbf{then}
11: \hspace{2em} $y -= 1$
12: \hspace{2em} $d = d + 4 \times (x - y) + 10$
13: \hspace{1em} \textbf{else}
14: \hspace{2em} $d = d + 4 \times x + 6$
15: \hspace{1em} \textbf{end if}
16: \hspace{1em} \textit{Plot\_Circle}(x_{\text{center}}, y_{\text{center}}, x, y)
17: \textbf{end while}
5.4 Intersection Area

For finding intersection areas we will use sweep line algorithm. Since the rectangles must not be parallel to the pivot, it is easier to change the problem to an effectively solved one: compute the intersections of the edges of the rectangles. A rectangle is represented by two points, one lower-left point and one upper-right point.

A sweep line is an imaginary vertical line which is swept across the plane rightwards. That’s why, the algorithms based on this concept are sometimes also called plane sweep algorithms. We sweep the line based on some events, in order to discretize the sweep. The events are based on the problem we are considering, for this problem which are the vertical edges. When we encounter a left edge, we do some action and when we encounter a right edge, we do some other action. Left edge is represented by lower-left point and right edge by upper-right point.

We start our algorithm by sorting the events by x coordinates. When a lower left point of a rectangle is hit (i.e., we encounter left edge of rectangle), we insert the rectangle into the set. When we hit an upper right point of a rectangle (we encounter right edge of rectangle), we remove the rectangle from the set. At any instance, the set contains only the rectangles which intersect the sweep line (rectangles whose left edges are visited but right edges are not).

The area swept at any instance is \( \Delta y \ast \Delta x \) where \( \Delta y \) is the length of the sweep line which is actually cut by the rectangle(s) and \( \Delta x \) is the distance between two events of this sweep line.

Now to find the length of the sweep line cut by the rectangles, We use the line sweep technique and this time we apply it 90 degrees rotated, i.e., we sweep a horizontal line from bottom to up. The events for this sweep line would be the horizontal edges of the active rectangles(rectangles cut by vertical sweep line). When we encounter a bottom horizontal edge of an active rectangle, we increment the counter (counter here maintains the number of rectangles that overlap at current time) and we decrement it on top horizontal edge of active rectangle. When the counter becomes zero from some non zero value, we have found cut length of the vertical sweep line, so we add the area to our final answer.
5.4.1 Sweep Line Algorithm

- build a set $E$ containing all edges, and the id with the rectangle they belong to; we will get an arrangement of tuples of the form $((x_{\text{start}}, y_{\text{start}}), (x_{\text{end}}, y_{\text{end}}), r_{\text{id}})$, where $r_{\text{id}}$ is the ID of the corresponding rectangle

- use a sweep line algorithm to discover the crossing points of those lines

The sweep line stops at each $x$-coordinate in $E$, i.e. all start and end coordinate values. For each new start coordinate, put the relating line in a transitory set $T$. For every new end-coordinate, remove the relating line from $T$. Moreover to adding new lines to $T$, we check for each new line whether it crosses with one of the lines present in $T$. If they do, the corresponding rectangles do, too.

Algorithm 6: \textit{Find Intersections}(E)

\begin{enumerate}
\item $Q \leftarrow \emptyset$
\item Insert the segment endpoints into $Q$
\item $T \leftarrow \emptyset$
\item \textbf{while} $Q \neq \emptyset$ \textbf{do}
\item extract the next event point $p$
\item $Q \leftarrow Q - p$
\item \textbf{end while}
\end{enumerate}

Algorithm 6 shows the pseudocode for finding the intersection points of the squares and rectangles. It takes the set of edge lines $E$ of the squares and rectangles of whose intersection points are need to be determined. The output is the set of intersection points among the edge lines in $E$. At the first step, using an empty queue, Algorithm 6 initializes an event queue $Q$ and inserts the segment endpoints into the queue $Q$. An status structure $T$ is initiazed as \textit{null}. The algorithms runs until the event queue $Q$ is not \textit{null}. It calculates the next event point $p$ in the event queue $Q$ and deletes it. It then calls the function \textit{Handle Event Point}(p) whose algorithm is discussed in Algorithm 7.

Algorithm 7 shows the pseudocode for handling the event points $p$ found in the event queue $Q$. At the first step, the algorithm initializes $S(p)$ with the segments containing $p$, $L(p)$ with the segments with
Algorithm 7: \textit{Handle\_Event\_Point}(p)

1: $S(p) \leftarrow$ segments containing $p$
2: $L(p) \leftarrow$ segments with left endpoint $p$ $L(p) \subseteq S(p)$
3: $R(p) \leftarrow$ segments with right endpoint $p$
4: $C(p) \leftarrow$ segments with $p$ in interior $C(p) \subseteq S(p)$
5: if $|L \cup R \cup C| > 1$ then
6: report $p$ as an intersection along with $L$, $U$, $C$
7: end if
8: $T \leftarrow T - (R \cup C)$
9: $T \leftarrow T \cup (L \cup C)$
10: if $L \cup C = \emptyset$ then
11: $s_b \leftarrow$ neighbors right below $p$ in $T$
12: $s_a \leftarrow$ neighbors right above $p$ in $T$
13: $Find\_New\_Event(s_b, s_a, p)$
14: else
15: $s' \leftarrow$ lowest segment in $L \cup C$
16: $s_b \leftarrow$ segment right below $s'$
17: $Find\_New\_Event(s_b, s', p)$
18: $s'' \leftarrow$ highest segment in $L \cup C$
19: $s_a \leftarrow$ segment right above $s''$
20: $Find\_New\_Event(s'', s_a, p)$
21: end if
left endpoint \( p \), \( L(p) \) is a subset of \( S(p) \) \((L(p) \subseteq S(p))\), \( R(p) \) with the segments with right endpoint \( p \) and \( C(p) \) with the segments with \( p \) in interior \((C(p) \subseteq S(p))\). The algorithm checks if \(|L \cup R \cup C|\) is greater than 1 and if it is true then it reports \( p \) as an intersection along with \( L \), \( U \), \( C \). The algorithm then removes \((R \cup C)\) from \( T \) and adds \((L \cup C)\) into \( T \). If \( L \cup C \) becomes \( \emptyset \) then the algorithm initializes \( s_b \) with the neighbors right below \( p \) in \( T \), \( s_a \) with the neighbors right above \( p \) in \( T \) and calls the \( \text{Find}_\text{New}_\text{Event}(s_b, s_a, p) \) function with the parameters \( s_b, s_a \) and \( p \). If \( L \cup C \) is not \( \emptyset \) then the algorithm initializes \( s' \) with the lowest segment in \( L \cup C \), \( s_b \) with the segment right below \( s' \) and calls the \( \text{Find}_\text{New}_\text{Event}(s_b, s', p) \) function with the parameters \( s_b, s' \) and \( p \). The algorithm then initializes \( s'' \) with the highest segment in \( L \cup C \), \( s_a \) with the segment right above \( s'' \) and calls the \( \text{Find}_\text{New}_\text{Event}(s'', s_a, p) \) function with the parameters \( s'', s_a \) and \( p \).

\[
\text{Algorithm 8: } \text{Find}_\text{New}_\text{Event}(s_l, s_r, p)
\]

1. \( q = \text{Intersect}(s_l, s_r) \)
2. \( \text{if } q \&\& ((q.y < p.y) || (q.y == p.y) \&\& (q.x > p.x)) && \text{Q.contains}(q) \) then
3. \( \text{Q.insert}(q) \)
4. \( \text{end if} \)

The pseudocode for the function \( \text{Find}_\text{New}_\text{Event}(s_l, s_r, p) \) is shown in the Algorithm 8 for finding new events. If \( s_l \) and \( s_r \) intersect to the right of the sweepline, or on it and below the current event point \( p \), and the intersection is not yet present as an event in the event queue \( Q \) and If \( s_l \) and \( s_r \) is on the sweepline, or intersect to the right of it and below the current event point \( p \), and the intersection is not yet exist as an event in the event queue \( Q \) then the algorithm adds the intersection point as an event to the event queue \( Q \).

Let consider the sweep line to be a vertical line. We have to somehow maintain the intersection of the the rectangles with the sweep line. Let \( y \) be the sum of all the intercepts. If we consider a sweep of distance \( \Delta x \), in which the intercepts remain unchanged, then area swiped by the sweep line is

\[
\Delta A = y \star \Delta x
\]  \hspace{1cm} (5.1)

Total area would then be nothing but the sum of these quantities

\[
A = \sum y \star \Delta x
\]  \hspace{1cm} (5.2)

Thus the problem at hand becomes to maintain the intercepts. The \( y \) can change only at
CHAPTER 5. ALGORITHMS

1. The beginning of a rectangle.

2. The end of the rectangle.

These become the event point of the algorithm. Now, we will look into various methods of keeping track of these $y$’s. At any moment of the sweep all intercepts are on the sweep line. The total length of the intercept on the line is the our required $y$.

5.4.2 $R^*$-Tree Algorithm

$R$-trees are tree data structures used for spatial access methods, i.e., for indexing multi-dimensional information such as geographical coordinates, rectangles or polygons. A common real-world usage for an $R$-tree might be to store spatial objects such as restaurant locations or the polygons that typical maps are made of: streets, buildings, outlines of lakes, coastlines, etc. and then find answers quickly to queries for finding locations of POIs. $R^*$-tree is a variant of $R$-trees used for indexing spatial information. $R^*$-trees have slightly higher construction cost than standard $R$-trees, as the data may need to be reinserted; but the resulting tree will usually have a better query performance. As we will insert the results offline but perform query online so we will use the $R^*$-tree to store locations identified from offline version. The $R^*$-tree uses the same algorithm as the regular $R$-tree for query and delete operations. When inserting, the $R^*$-tree uses a combined strategy. For leaf nodes, overlap is minimized, while for inner nodes, enlargement and area are minimized. So first we will discuss query and delete operations of $R$-tree.

$R$-trees are hierarchical data structures and they are used for the dynamic organization of a set of $d$-dimensional geometric objects representing them by the minimum bounding $d$-dimensional rectangles (for simplicity, MBRs). Each node of the $R$-tree corresponds to the MBR that bounds its children. The leaves of the tree contain pointers to the database objects instead of pointers to children nodes. The nodes are implemented as disk pages. An $R$-tree of order $(m, M)$ has the following characteristics:

- Each leaf node (unless it is the root) can host up to $M$ entries, whereas the minimum allowed number of entries is $m \leq M/2$. Each entry is of the form $(mbr, oid)$, such that $mbr$ is the MBR that spatially contains the object and $oid$ is the object’s identifier.
• The number of entries that each internal node can store is again between \( m \leq M/2 \) and \( M \). Each entry is of the form \((\text{mbr}, p)\), where \( p \) is a pointer to a child of the node and \( \text{mbr} \) is the MBR that spatially contains the MBRs contained in this child.

• The minimum allowed number of entries in the root node is 2, unless it is a leaf (in this case, it may contain zero or a single entry).

• All leaves of the R-tree are at the same level.

In all R-tree variants that have appeared in the literature, tree traversals for any kind of operations are executed in exactly the same way as in the original R-tree. Basically, the variations of R-trees differ in how they perform splits during insertion by considering different minimization criteria instead of the sum of the areas of the two resulting nodes.
Chapter 6

Experiments

In this chapter, we evaluate the performance of our approach for processing POI identification queries through extensive experiments. Since there is no existing work for POI identification queries in the literature, we compare the locations retrieved from our proposed POI extraction approaches with the actual ones by manually collecting the actual location of some of the POIs.

We evaluate our approach in Euclidean dataspaces using real world datasets collected by friends, colleagues and families which contains 10700 POIs of 11 users. The users collected information of 78 POIs for a total of 33 days within a duration of three month from Jun 2015 to Nov 2015. 2 female and 9 male voluntarily participated in collecting the data all ranging in age from 25 – 35.

As we have already mentioned, for each record the dataset has the following parameters: a starting latitude \( l_i.S_{lat} \), a starting longitude \( l_i.S_{lon} \), a starting time-stamp \( l_i.S_{time} \), an ending latitude \( l_i.E_{lat} \), an ending longitude \( l_i.E_{lon} \), and an ending time-stamp \( l_i.E_{time} \). Particularly for our problem each point additionally contains a user defined text \( l_i.Txt \), corresponding starting speed \( l_i.S_{spd} \), ending speed \( l_i.E_{spd} \), starting bearing \( l_i.S_{bear} \), and ending bearing \( l_i.E_{bear} \).

Geographical information was obtained from the geographical coordinates (latitude and longitude) of network antennas. The precision of the spatial accuracy of the mobile device activity corresponds to the coverage of a network antenna. The coverage area is not spatially fixed and varies according to population density. As Dhaka is one of the most densely populated cities of the world, the analysis on the sample collected from here gives us some unique results because of the uniqueness of the city.
### Table 6.1: Example of spatial trajectory data collected by crowdsourcing

<table>
<thead>
<tr>
<th>Date</th>
<th>Text</th>
<th>St. Time</th>
<th>St. Latitude</th>
<th>St. Longitude</th>
<th>St. Speed</th>
<th>St. Bearing</th>
<th>En. Time</th>
<th>En. Latitude</th>
<th>En. Longitude</th>
<th>En. Speed</th>
<th>En. Bearing</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-12-09</td>
<td>four season</td>
<td>16:01:14</td>
<td>23.751445</td>
<td>90.36783</td>
<td>0.926</td>
<td>151.222</td>
<td>16:01:20</td>
<td>23.751416</td>
<td>90.367901</td>
<td>1.002</td>
<td>167.615</td>
</tr>
<tr>
<td>2016-12-09</td>
<td>bank al falah</td>
<td>16:07:51</td>
<td>23.743973</td>
<td>90.372927</td>
<td>2.466</td>
<td>139.859</td>
<td>16:07:56</td>
<td>23.743806</td>
<td>90.373052</td>
<td>3.044</td>
<td>147.061</td>
</tr>
<tr>
<td>2016-12-09</td>
<td>kfc</td>
<td>16:08:03</td>
<td>23.743601</td>
<td>90.373212</td>
<td>3.537</td>
<td>137.416</td>
<td>16:08:05</td>
<td>23.743545</td>
<td>90.373267</td>
<td>3.668</td>
<td>137.849</td>
</tr>
<tr>
<td>2016-12-09</td>
<td>pizza hut</td>
<td>16:05:27</td>
<td>23.747903</td>
<td>90.370274</td>
<td>3.922</td>
<td>149.588</td>
<td>16:05:32</td>
<td>23.747756</td>
<td>90.370354</td>
<td>3.883</td>
<td>162.529</td>
</tr>
</tbody>
</table>

We use an Intel Core i3 machine with 2.30 GHz CPU and 2GB RAM to run the experiments. For each set of experiments, we measure two performance metrics: the average processing time and average I/O overhead (I/O access in $R^*$-tree). The metrics are measured by running 78 independent POI identification queries having random number of data in the datasets, and then taking the average of processing time and I/O access.

The raw spatial trajectory data are contained in simple but large text files. We have developed an android application to collect these data and for processing this data using our algorithm we developed a program based on our proposed prediction classifiers and clustering algorithms using JAVA. We have obtained most of the the outputs in two types, output files and visualizations For better understanding of the results discovered, the visualization of the location data is necessary. We have used Google Maps API in our android and JAVA programs, for visualization the locations on map.

### 6.1 Answer Accuracy

We evaluated the accuracy of the POI identification queries using our proposed algorithm. The accuracy w.r.t. the centroid with the actual location is shown in Figure 6.1. After generation of locations of POIs using our method, we manually collected the actual location of 30 keywords. We then computed the accuracy comparing with the actual location and the generated location of the POI. The generated locations dont match exactly with the actual ones'. Our POIs are restaurants, malls, hospitals, hotels which are actually a building and any building covers at least 2,000 square feet of area. So, we have checked if the centroid coordinates of the identified resides inside any of the rectangles computed and tagged with each POI.
If the centroid coordinates of the highest probable region is inside probable rectangles, we take that as a correct generated POI. We have investigated the possibility of the points residing inside the rectangles with lower probabilities. Considering rectangles having lower probabilities almost all the POIs resides inside either higher probable rectangle or lower probable rectangles. We have conducted a multiple number of tests by taking different points as inputs.

### 6.1.1 Offline Version

For identifying the answer accuracy of POI identification queries for offline version, we performed several set of experiments by varying the following parameters:

1. Travel mode and speed range
2. User and POI direction, and
3. Time of location tag

![Figure 6.1: Average centroid and actual location of POIs](image)

**6.1.2 Travel mode and speed range**

At our first step we have removed the POIs having speed more than 40kmph as noisy data. We have categorized the dataset with travel modes and speed range also. And then we have implemented tests with datasets having single travel mode and mixed travel modes. Figure 6.2 shows the accuracy graph for different travel modes. It is seen from our test results that we get more accurate results with the datasets having multiple travel modes rather than datasets having single travel mode. We have also categorized the data sets with speed ranges and mix speeds. Then we have conducted some tests with these data sets. And with mix speed data set we get better performance. For 10700 no of POIs, mix
Figure 6.2: Accuracy Graph for Different Travel Mode

Table 6.2: Accuracy with different travel mode data sets

<table>
<thead>
<tr>
<th>Travel Mode</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>47%</td>
</tr>
<tr>
<td>Bus</td>
<td>59%</td>
</tr>
<tr>
<td>Rickshaw</td>
<td>67%</td>
</tr>
<tr>
<td>Pedestrian</td>
<td>78%</td>
</tr>
<tr>
<td>Mix Travel Modes</td>
<td>95%</td>
</tr>
</tbody>
</table>

travel mode achieved an overall accuracy of 95%, outperforming other single travel modes.

6.1.3 User and POI direction

After computing the result we have categorized the data sets again with the direction of users and the direction of the actual POIs. We have classified the datases with points having user’s direction and location of the POI at same sides and points having user’s direction and the location of the POI at opposite direction. And then we have conducted tests with these two datasets. It is seen that the dataset having points having user’s direction and the location of the POI at same side provides better results than the other dataset. The dataset size for both the cases are shown in Table ?? and the accuracy graph is given in Figure 6.3. We could observe that the more points we get as input the more
CHAPTER 6. EXPERIMENTS

Figure 6.3: Accuracy Graph for User and POI direction

accurate the result becomes. In future we propose to add the direction parameter to our experiment to get exact location of the POIs.

Table 6.3: Accuracy with different user and POI direction

<table>
<thead>
<tr>
<th>Percentage of data</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User and POI same direction</td>
</tr>
<tr>
<td>15%</td>
<td>48%</td>
</tr>
<tr>
<td>36%</td>
<td>59%</td>
</tr>
<tr>
<td>53%</td>
<td>76%</td>
</tr>
<tr>
<td>82%</td>
<td>95%</td>
</tr>
<tr>
<td>100%</td>
<td>99%</td>
</tr>
</tbody>
</table>

6.1.4 Time of location tag

We have divided the data sets into two different groups depending on the time the location was tagged. First time frame was from 06:00 am to 06:00 pm and the second time frame was from 06:00 pm to 06:00 am. We can see that the most of the data are collected within the first time frame and a few were collected in the second time frame. We have performed test on both the datasets. The second data set gives less accuracy with the few data where the first data set gives better accuracy.
CHAPTER 6. EXPERIMENTS

Figure 6.4: Accuracy Graph for Time of location tag

Table 6.4: Percentage of points per data set

<table>
<thead>
<tr>
<th>Set</th>
<th>Percentage of data</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set A</td>
<td>15%</td>
<td>30%</td>
</tr>
<tr>
<td>Set B</td>
<td>36%</td>
<td>45%</td>
</tr>
<tr>
<td>Set C</td>
<td>53%</td>
<td>62%</td>
</tr>
<tr>
<td>Set D</td>
<td>82%</td>
<td>84%</td>
</tr>
<tr>
<td>Set E</td>
<td>100%</td>
<td>94.5%</td>
</tr>
</tbody>
</table>

6.1.5 Online Version

For identifying the answer accuracy of POI identification queries for online version, we performed several set of experiments by varying the parameter:

(i) Dataset size

6.1.5.1 Dataset size

We have also analyzed the answers comparing with the number or points collected. It is found that the more the numbers of points are collected from crowdsourced the more the result gets accurate. We have implemented some tests by taking a different percentage of inputs. The percentage of points
per set is given in Table 6.4.

![Graph showing accuracy vs data size](image)

**Figure 6.5: Accuracy Graph**

Figure 6.5 shows that the accuracy of the identified location of POI increases with the increase of number of crowdsourced data. We achieve more than fifty percent of accuracy with only 49% of our collected data. And finally we get 94.5% accuracy with all the spatio-textual data collected by crowdsourcing.

We have also reexamined the results of all four parameters Dataset size, Travel mode and speed range, User and POI direction, and Time of location tag by considering the lower probable regions. And it is seen that we can get an accuracy of 100% if we consider the lower probable regions as an accurate answer for the POI identification queries.

### 6.2 POI Identification Queries

POI identification queries for all the travel modes, we performed several set of experiments by varying the following parameters:

(i) the number of logged information for each POI $n$

(ii) the number of logged information for each travel modes $n_m$

(iii) the number of travel modes $t_m$, and

(iv) the dataset size $d_s$
Table 6.5: Parameter settings for POI identification queries

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of logged information for each POI ((n))</td>
<td>1, 10, 20, 30, 40, 50</td>
<td>20</td>
</tr>
<tr>
<td>Number of logged information for each travel modes ((n_m))</td>
<td>10, 20, 30, 40, 50</td>
<td>20</td>
</tr>
<tr>
<td>Number of travel modes ((t_m))</td>
<td>1, 2, 3, 4</td>
<td>2</td>
</tr>
<tr>
<td>Dataset size ((d_s)) (number of POIs in thousands)</td>
<td>1, 2, 4, 6, 8, 10</td>
<td>-</td>
</tr>
<tr>
<td>Dataset distribution</td>
<td>Uniform</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.5 shows the range and default values used for each parameter. To observe the effect of a parameter in an experiment, the value of the parameter is varied within its range, and other parameters are set to their default values.

### 6.2.1 Euclidean Space

#### 6.2.1.1 Effect of Number of Logged Information for Each POI \((n)\)

We study the impact of number of logged information for each POI on POI identification query by varying the number of logged information for each POI using 1, 10, 20, 30, 40 and 50 and measuring the required processing time and number of I/O access for both UNION and INTERSECTION. For offline version we run the full process described in chapter 4 and save it for future use. For online version for the number of logged information for each POI using 1 we need to run the full process as the offline version as there is no saved information for the POI. For the rest of 10, 20, 30, 40 and 50 number of POIs for online version we need to search the information from the \(R^*\)-tree where the result from the offline version is saved.

Figures 6.6(a) and 6.6(b) show the processing time and for our UNION and INTERSECTION functions. For both functions Figures 6.6(d) and 6.6(e) show the I/O access for our online and offline version.
Figure 6.6: Effect of number of logged information for each POI \((n)\) in Euclidean space
We observe that both processing time and I/O access slightly increase with the increase of the number of logged information for each POI. Our online version requires significantly less processing time and I/O access than the offline approach, which is expected. The offline version computes the approximate location of each POI and runs the complete process, and thus, the identification function is accessed for individual POI. On the other hand, our online version only computes the full identification process for 1 information logged for a POI and it only searches for the saved location of the POI.

In Figures 6.6(c) and 6.6(f), we show a comparative view of the online version for both metrics the processing time and I/O access, respectively. For both metrics INTERSECTION function shows smaller changes with the increase of number of logged information for POIs whereas the UNION function shows large changes in terms of processing time and I/O access. The reason behind this is, for POI identification queries with online version it only requires to retrieve information from the saved $R^*$-tree where the result from the offline version is saved. For offline version we need to calculate the bound of circular and elliptical region changes for each POI which impacts the processing time and I/O access. The number of rectangles increases with the number of logged information for POIs for both functions, thus requiring more time for the offline version.

6.2.1.2 Effect of Number of Logged Information for Each Travel Modes \((n_m)\)

In our experiments, we study the impact of number of logged information for each travel modes on the performance of POI identification query by varying the number of logged information for each travel modes using 10, 20, 30, 40 and 50 and measuring the required processing time and number of I/O access for online and offline version. Figures 6.7(a-b) and 6.7(d-e) show that the processing time and I/O access, respectively, for UNION and INTERSECTION function, increase with the increase of \(n_m\). The results show that our online approach performs much better than the offline approach in terms of both I/O access and processing time. For higher value of logged information for each travel mode the offline version needs to calculate more bounding area of rectangles. Specifically, the improvement for the I/O access is more pronounced than the processing time. We observe in Figures 6.7(d-e) that the processing required by the online approach remains almost constant, and the processing time for the offline version sharply increases with the increase of \(n_m\). The reason is as follows. For the change of \(n_m\) to \(n_m + 1\), the number of circular and/or elliptical regions are also increased, whereas the processing time of the online approach depends on the range search
complexity of the $R^*$-tree. For an additional number of logged POI, the search range may remain constant or slightly increases since the average search complexity is $O(n \log n)$ and for worst case $O(n)$.

For both metrics, the processing time and I/O access, Figures 6.7(c) and 6.7(f) show a comparative view of the online version for both the functions, respectively. We observe that for both metrics, show almost similar changes with the increase of number of logged information for each travel modes.

For having different aggregate functions, in a POI identification query, bound for each POI circular and elliptical search region changes but on average search region remains same. Thus both aggregate functions show similar trends for the processing time and I/O access.

Figure 6.7: Effect of number of logged information for each travel modes ($n_m$) in Euclidean space
6.2.1.3 Effect of Number of Travel Modes ($t_m$)

We vary the number of travel modes by 1, 2, 3 and 4 in our experiments to observe the impact on the performance of POI identification query and measure the required processing time and the number of I/O access for both online and offline version. Figures 6.8(a-b) and 6.8(d-e) show experimental results for different values of the number of travel mode $t_m$ for both \textsc{Union} and \textsc{Intersection} functions. We see that for both approaches, the processing time and I/O access increase with the increase of $t_m$. This is because the POI search region becomes large if the number of POIs in different travel modes are distributed in a large area of the total space. For both metrics, our online approach performs better than the offline approach, which is for the similar reasons mentioned for the experiments of varying $n_m$.

![Diagrams showing the effect of number of travel modes on processing time and I/O overhead for \textsc{Union} and \textsc{Intersection} functions.]

Figures 6.8(c) and 6.8(f) show a comparative chart of the online version for metrics the processing
CHAPTER 6. EXPERIMENTS

71
time and I/O access and both the functions, respectively. Both the function shows similar trends for
the processing time and I/O access for the similar reason that we have described in Section 6.2.1.1
and 6.2.1.2.

6.2.1.4 Effect of Dataset Size \( (d_s) \)

![Figure 6.9: Effect of dataset size \( (d_s) \) in Euclidean space](image)

In this experiment, we varied the size of real dataset from 1k to 10k (1k, 2k, 4k, 6k, 8k, 10k). To
show the effect of dataset size\( (d_s) \), we run experiments using spatio-textual trajectory datasets
collected from crowdsourcing. The corresponding experimental results are shown in Figures 6.9(a-b)
and 6.9(d-e) for both online and offline version for Union and Intersection functions. In this
experiment, we examine the performance difference of the two approaches with respect to data
set size \( (d_s) \). Figures 6.9(a-b) and 6.9(d-e) show that as the size increases, processing time and
I/O access increases for both approaches, which is expected. Like other experiments, the online approach takes much less processing time and I/O access than the offline approach for any dataset size.

Both Union and Intersection function shows similar trends for the processing time and I/O access which we can observe deeply in Figures 6.9(c) and 6.9(f) for the processing time and I/O access, respectively. The reason behind this is similar that we have described in Section 6.2.1.1, 6.2.1.2 and 6.2.1.3.
Chapter 7

Conclusions

In this thesis, we studied the GPS trajectories generated by multiple users, and the identification of POIs from the crowdsourced data. We have used the spatio-temporal data collected from crowdsourcing to identify location of POIs which enables a user to find locations of unfamiliar places. We propose the first solution to evaluate location identification queries in both online and offline models. To identify locations of POIs, we consider different travel modes (walk, rickshaw, car and bus). We have been motivated by the advantage of using crowdsourced data, which is easy to collect in massive scale without any cost to overcome the cost management limitation.

Specifically, we have proposed refinement techniques for the POI search space and a dynamic approach to identify locations of the POIs, which are the key ideas behind the efficiency of our approach. We have exploited geometric properties to refine the POI search space and prune POIs to reduce the area of the search space to accurately identify the locations. To identify the location of the POIs, we have developed an efficient dynamic programming technique that eliminates the extra regions around the POIs that are really far from the actual locations.

We have proposed two different models including online and offline versions by identifying the locations in real-time for the given POI. For offline version we identify the location and save it tagged with the keyword and whenever any user searches for the location of a POI in our system it will recommend the POI saved with the keyword. In particular we are the first to proposed the method to identify not only popular POIs but also others in which people might get interested. Such information can help
us find known and unknown locations and enable travel recommendation as well as mobile tourist guidance.

Since there exists no approach to identify locations of POIs from spatio-textual crowdsourced data in the literature, to validate the efficiency of our proposed approach in experiments, we have used straightforward approach by manually collecting the actual locations of some of the POIs. We have performed experimental evaluation of the proposed techniques and provided an comparative analysis of experimental results using the collected datasets. Experimentation results showed that our approach achieved efficiency and accuracy.
References


[28] Gennady L. Andrienko, Natalia V. Andrienko, Christophe Hurter, Salvatore Rinzivillo, and Stefan Wrobel. From movement tracks through events to places: Extracting and characterizing


