**Design and Development of Software Agents for Location**

**Privacy-risk estimation**

Shashank Sharma

August 2018

**Abstract**

The usage of mobile devices has become ubiquitous in today’s world. The human geographical location is shared with different applications on a mobile device. The sharing of location is active not only when the application is used, but also share the user whereabouts with the third-party applications in the background. This location is used by different applications for advertising, recommendation, finding new friends, suggesting new point of interests based on user trends. This location data can also be used by different third-party applications to predict user’s future locations. This in turn can model the entire user mobility patterns. This is a privacy attack which makes use of user’s past location data.

In this thesis we present an algorithm which predicts user future movements with confidence percentages. This algorithm is first implemented on python using Microsoft Geolife data. The raw trajectories are used from Microsoft Geolife data. Markov chain is formed on this data to simulate a model how user data can be used to predict future locations. This data contains 182 user trajectories data for 5 years. The same algorithm is then implemented on Android device.

**Acknowledgement**

Contents

[**1 Introduction** 5](#_Toc528675076)

[1.1 Motivation 5](#_Toc528675077)

[1.2 Problem Statement 6](#_Toc528675078)

[1.3 Design 6](#_Toc528675079)

[1.4 Outline 7](#_Toc528675080)

[**2** **Related Work** 8](#_Toc528675081)

[**3** **System Model** 12](#_Toc528675082)

[3.1 Components 13](#_Toc528675083)

[3.2 Hypothesis 15](#_Toc528675084)

[**4** **Proposed Concept** 16](#_Toc528675085)

[4.1 Markov Model for Location Prediction 16](#_Toc528675086)

[4.2 Algorithm 18](#_Toc528675087)

[**5** **Implementation** 20](#_Toc528675088)

[5.1 Variables Used 20](#_Toc528675089)

[5.2 Stay-points 20](#_Toc528675090)

[5.2.1 Algorithm 22](#_Toc528675091)

[5.2.2 Implementation Result 25](#_Toc528675092)

[5.3 State Formation 26](#_Toc528675093)

[5.3.1 Algorithm 27](#_Toc528675094)

[5.4 State Weights 28](#_Toc528675095)

[5.4.1 Algorithm 29](#_Toc528675096)

[5.4.2 Implementation Result 30](#_Toc528675097)

[5.5 Forming Markov chain 31](#_Toc528675098)

[5.5.1 Algorithm 32](#_Toc528675099)

[5.5.2 Implementation Result 34](#_Toc528675100)

[5.6 Path Prediction 35](#_Toc528675101)

[5.6.1 Implementation and Result 36](#_Toc528675102)

[5.6.2 Improvements 37](#_Toc528675103)

[5.7 Android implementation 40](#_Toc528675104)

[**6 Evaluation** 43](#_Toc528675105)

[6.1 User data analysis 43](#_Toc528675106)

[6.3 Evaluation and Results 45](#_Toc528675107)

[6.2 Discussion and Summary 45](#_Toc528675108)

[**7 Conclusion and Future Work** 46](#_Toc528675109)

[7.1 Summary 46](#_Toc528675110)

[7.2 Future Work 46](#_Toc528675111)

# **1** **Introduction**

The personal computers and mobile devices have become part of everyday life. It is ubiquitous to interact with computers on an everyday basis. Some of these computers are capable of location awareness, for instance, a mobile device. There are many applications in our mobiles and computers which uses location data for its better recommendations and personalized advertisements. These applications are called location-based services. The applications use user location data while the application is active and inactive. These applications are often using user location to provide a personalized experience. Most applications have in their terms and condition mentioned the usage of the location data which are easily ignored or misunderstood. Consider an example of a weather application forecasting the weather for the next few days on your mobile device. The weather application is using the current location, every time, the user is moving to a new location for fitting weather forecasts. Another example is of a social networking website like Facebook. The user makes check-ins and shares location continuously to find friends and recommendations. It is important to understand that this location data can be used to model human mobility. It should not be forgotten that it enhances quality-of-life but at the same time can reveal mobility patterns based on the data received in the past.

The geographic location can be shared from many sources like Global System for Mobile Communication GSM, Global Positioning System GPS, Wi-Fi network location and so on. The most predominant use of location data is the modern navigation system like Google Maps. The user shares the outdoor movements with the application to receive the path recommendations. This reveals two important facts, the source location and the destination location, and the path taken. The source could be “home”, “hotel”, “shopping mall” or a “restaurant” location and so could be the destination. When this information is collected for several days, it could reveal user “home”, “work” or “favorite restaurant” locations, the time spent at these locations and the transitions from one place to another. This could help, for instance, a restaurant application to suggest a new restaurant, keeping in mind the user’s home location, type of food he/she prefers/like and his/her time preference of visiting a restaurant. On the other hand, this can help to build a model for user mobility which, in turn, can make predictions based on user current location. For instance, if the user is at “home” at 9 am, the mobility pattern can reveal his next locations for the rest of the day, using the model. This is a privacy attack. This information can also be leaked on third-party applications and can have unauthorized access. It is easy to oversee the privacy threats that location-based services can impose. The goal of this thesis to picture the privacy threats to the user by making location-based predictions using user’s location data for a few weeks.

## Motivation

The location-based data can tell a lot about the user. If the location-based data is received for a few weeks, it is easy to answer questions like, where does user live/work? Where is user’s club/gym? What restaurant does he/she likes? Where is he/she on weekends? Which hospital has he/she been visiting? All the answers to these questions can give an insight about user’s private life and whereabouts. After a few weeks of data, the algorithm can precisely define user movement trends, his/her favorite places, his/her lifestyle and so on. With continuous learning, this data can be updated and have the actual user home location and work location updated with time.

This private data is often distributed to third parties. This could be an attack on user’s private life, compromising user privacy and sharing his data with other applications, friends and family. With the advancement of Artificial Intelligence and Machine Learning, it has become ever easier to exploit this data, understand bulk geographic data and infer meanings from several locations.

## Problem Statement

Use of mobile devices is ubiquitous. Users are often using applications on the mobile system which uses their location coordinates. Although the location usage is consensual, the duration of sharing is often not well understood by the users. The applications use the location data in the background when the application is not even in use. According to a Pew survey [1], 91% of Americans agree that they have lost control on their private data and 64% reported that government must regulate advertisers. Another survey reported that only 9% users are confident that social media companies will protect their data.

The next locations can be predicted based on the user’s current location. The prediction can be done for next hours based on this information shared. For instance, if the user who usually leaves for “work” location at 8 am and stays there till 6 pm. If the user shares his location at “work” location at 8 am, it can be easily predicted that he will be at “work” location till 6 pm. The location coordinates at 8 am were enough to predict his/her location till 6 pm. The threat of location prediction and exploitation of user privacy is to be shown to users so that the user can make a wise decision before sharing the location with third-party applications on mobile devices.

The goal of this thesis is to simplify the location prediction algorithm on python and test the prediction accuracy. The same algorithm is implemented on a mobile device. The model of location prediction is built which takes in the location coordinates as input in an online manner and forms a markov chain model. This markov chain model must be then used to make predictions based on current location and hour of the day. The location prediction should continue till the confidence falls below a certain threshold value. The predicted paths are shown to the users to inform about the predictability of his/her whereabouts.

## Design

In this thesis, we present a location prediction model. The location-based data acts as input for the model and forms a prediction model as an output. This model is then used to make future location predictions with confidence percentages, based on a known location as input. The location data used is from Microsoft Geolife data of 182 users for 5 years.

This algorithm is designed to predict human locations in a real-world scenario. The Geolife data is taken as input and then processed using the algorithm. The input GPS coordinates data with the date and time information is fed to the algorithm. The algorithm processes this information to extract significant locations like “home”, “work”, “club”, “restaurant” and builds a model with probabilities of transitioning from one place to another. This model is called markov chain model. We use this location prediction model to predict the future locations with confidence percentages.

## 1.4 Outline

This thesis is organized as follows:

**Chapter 2 (Related Work)** briefs the related work researched in location prediction and presents the basic understanding of the topics.

**Chapter 3 (System Model)** describes the building blocks of the model which includes components used in the algorithm used and the hypothesis in general. The components explain the [intermediary](https://www.google.de/search?q=intermediary&spell=1&sa=X&ved=0ahUKEwjnh8r0xKPeAhUGWSwKHWMECDAQBQgrKAA) steps followed to build location prediction model from raw GPS coordinate points.

**Chapter 4 (Proposed Model)** explains the algorithm build for location prediction. The algorithm explained here is generalized to be applicable to any location data. This chapter explains the aspects for building a model based on the location data received.

**Chapter 5 (Implementation)** contains the implementations along with the individual algorithms and the results. The algorithms describe the step-by-step approach to achieve each building block of the main algorithm. This chapter also include the results of application of these algorithms on the dataset.

**Chapter 6 (Evaluation)** contains the evaluation of the approach and results of the location prediction model. This outcome of the algorithm on the dataset is discussed here.

**Chapter 7 (Conclusion and Future Work)** has the concludes this thesis by summarizing the work done, results obtained, open topics and future work.

# **Related Work**

In this section, we introduce the related work in the field, the references and motivational work and few parts in details.

The author Baumann [2] compared 18 different location prediction algorithms. The focus here is on the accuracy of prediction along with other parameters to compare these algorithms with each other. Based on their analysis and knowledge gained during the algorithm comparison, they also present a new next-place prediction algorithm called MAJOR. The dataset used by the researchers is Nokia Mobile Data Challenge (MDC). It contains 37 user mobile phone data over 1.5 years.

They [2] have considered several spatial and temporal features with different combinations and named each algorithm based on the features used. For instance, the features used are, current location of the user P1, current and previous location of the user P2, time of the day H, Day of the week D, weekday or weekend W. Here P1 and P2 are spatial features and H, D and W are temporal features. Using the combination of these spatial and temporal features, they have formed several algorithms e.g., DP1, WHP2 and so on. On these algorithms, several performance metrics are calculated. Which contains factors like accuracy percentage A1 which is the ratio of correct predictions to the total predictions. Other performance factors which included the true positive, false positive, true negative and false negative with respect to transitions. For example, a true positive transition is the transition which is correctly predicted from one place to another and true positive transition rate TTPR is the ratio of true positive transitions over the total transitions. Similarly, other rates like false positive transition rate, are calculated. Some other interesting performance parameters included transition precision ratio which is calculated as ratio of number of correctly predicted transitions and the total predicted transitions. The researchers also considered the arrival and departure events prediction from a particular place as a performance metric.

Using the combination of different spatial and temporal features, they [2] have compared 18 different prediction algorithms for their predefined performance metrics. The first comparison is highlighted for algorithms considering only spatial features or only temporal features or both together. The most algorithms which can achieve a good prediction accuracy fail to predict a transition and vice versa. This led to the conclusion that there exists a trade-off between prediction accuracy and transition prediction. This is overcome by the novel approach introduced in the paper [2] called MAJOR. This new approach runs all 18 algorithms (spatial and temporal combinations) together and selects the one with the highest votes. This means that, if the highest vote among the 18 algorithms suggests a transition, then the transition is predicted, otherwise not. The same voting approach is applied for the prediction of next place. This gave MAJOR an accuracy of 82% but only 21% detection of true transition. To improve the transition detection ability, they have introduced a voting threshold. The analyses suggested that a median of 8 approaches predict a true transition and a median of 3 approaches predict a transition when no transition occurs. This will help to decide the voting threshold offline. If the minimum number of approaches voting for the transition is greater than of equal to the threshold, then it is considered a transition, otherwise not. The researchers [2] help us to understand the different metric which is important in location prediction algorithms. The paper also suggests that a high accuracy algorithm will have a trade-off for detecting the true transitions.

Another interesting research was done by Noulas [3] in the field of next place location prediction which suggests the next location prediction based on user behavior. The main idea is to use user check-ins on Foursquare to predict user movements. The data of 35 million check-ins from across the globe over the period of 5 years is used. The idea is explained how user check-ins not only allows us to see the locations user visited in the past but also help us understand the mobility patterns of the users. They have used prediction features like user preferences, the popularity of the places and geographic distance between places. On these features, they have applied supervised the learning linear model and M5 model trees.

One of the first tasks addressed [3] is the next check-in prediction. They rank all the possible next location check-in based on current check-in and suggest that 99% of the next check-ins are within 10 kilometers radius from the current check-in and is mostly in urban areas. The ranking is performed based on historical visits by the user to a place, categorical preferences based on what category of places have user checked-in in the past and social filtering based on where user’s friends is checked-in. The next task is a global mobility feature to determine check-in patterns irrespective of user preferences. This uses the popularity of the geographic location, geographic and relative distance of all the other locations from user’s current location, activity transition where few locations are visited after specific locations, for instance going to a hotel after an airport or railway station visit. Next, they assign the temporal feature to each place. Based on the hour category, what type of place has been checked-in in a particular hour of the day or week?

After the assigning these features, the ranks(k), percentile rank (PR) and average percentile rank (APR) are defined for each venue. The analyses from the researchers suggest that APR is scored higher for categorial preferences with 0.84 when compared to historical visits with APR 0.68 and social filtering with APR 0.61. In global mobility section, place popularity has better APR which is 0.86. Activity transition features also achieve only 0.60. The study [3] also suggested that people tended to stick to their set of location check-ins during the day time but visited new locations during evenings. All these features suggest that there could be many factors which can affect user movement patterns. They finally used all these features and combined them into a supervised learning framework. With M5 tree, they have received an APR of 0.94 and linear regression model only resulted with an APR of 0.81 which is lesser than many individual feature APR. The authors [3] explained how the prediction model can have better performances with several features combined.

Gomes [4] also discusses mobile based next location prediction based on current location. They suggest using contextual data along with spatial and temporal data associated with location. The mobile call/SMS logs, accelerometer and Bluetooth can have additional information which has not been investigated before for location predictions. The researchers explain how location prediction is very user specific, the data is evolving with changing city/work location, etc. and it is possible to have missing location data.

The model [4] pre-process the raw data which keeps the short-term data with its contextual information, the model should accept and integrate new location data and an updated check of actual next location vs predicted next location to keep an updated accuracy rate. This model is implemented in an online manner on a mobile device. The data used as an input is from Nokia Mobile Data Challenge (MDC) from 200 participants over one year. First, the raw data is processed to extract temporal features, phone status, phone usage and other features. These features include the hour and time duration of a visit, the ring-tone used, last call/SMS log and others. Then a classification technique is used with a software named WEKA.

The results [4] suggested that the regular users were easy to predict with an accuracy percentage of 80% where as the users with irregular movement patterns are difficult to predict. In feature selection phase, it has been found that keeping all the features gives the best results with accuracy of 92%.

The paper [4] also suggests an alternative advertisement approach. For instance, a user will be more interested in dinner promotion/discount before he/she goes outside on dinner time. The user can share the location-based data with the telecom provider, and the telecom service provider can act as the middle-man between the user and the restaurant advertisement company. The third party like, in this case, the restaurant company, can push the relevant advertainments on the user’s phone based on the predictions result shared by the telecom provider, without disclosing user’s information or location data to the third parties. In this scenario, the user can receive more relevant advertisements and still have not shared his private information with advertisement companies. This can help in preventing personal data to be shared with companies and third parties and hence preserving user privacy.

Baratchi [5] design hierarchical hidden semi-markov-model concerning spatio-temporal of location data to predict human mobility patterns. The hidden semi-markov model does not consider the distance between the two observations, whereas the proposed hierarchical model considers the distance between the observations. In this model, each state denotes either a stay-point or a transition from one place to another. The states which are more visited will be super states consisting of other states, and the states geographically closer or spatio-temporally closer are more likeable to be in one state. Hence, there are super states which contain other states. The next step is to map each location coordinate in a grid with cell id. The algorithm becomes expensive with increasing states. The states can be reduced by using states at a higher level in the hierarchy. This, in turn, reduces the total complexity of the algorithm. The researchers have used real-life location data Geolife dataset and Capricorn dataset. The approach has better results in the presence of noise and missing data.

Kang [6] investigate how to extract significant places for the users from raw coordinates data. The author suggests that users are more interested in “places” rather than location. By “places” they mean where the user work/live/play or so on. The researchers used Place Lab to collect user location data from Wi-Fi enabled devices which works best also for indoors locations. The exiting clustering algorithms are computationally expensive and result in insignificant locations. They also introduce a time-based clustering algorithm to determine user locations. This algorithm waits for the next location to determine the significant places. The cluster within a distance threshold which is stayed for at least a given time threshold is considered as a significant place. Hence this clustering excludes all irrelevant or shortly stayed locations. Their results show that the algorithm could extract significant places. The researchers also suggest that the locations must be labelled to extract semantic meaning behind the location coordinates like work/restaurants and so on.

# **System Model**

The human mobility pattern can be dependent on several features like user’s occupation. For instance, if the user is a, salesperson or has an occupation which requires daily travel, it is very unlikely that the user has a regular “home-work-home” pattern. These users are difficult to be predicted. There could be other users who have very regular movement patterns. These users are easily predicted. The idea is to have a prediction model which can work for everyone.

The user tends to have a pattern where the next location is dependent on their current location. Consider an example where “work” is often visited directly after “home”, but “home” is not very often visited directly after “work.” This is a common trend where the user visits restaurants, gym or some other location after work before he/she comes back to “home.” It is very often that after “supermarket” visit, the user tends to go back to “home.” These trends could be very often predictable but also sometimes not obvious. For instance, a restaurant visit could occur after home or work visit or even after a shopping mall visit. Hence, we can say that a large number of movements is dependent on the current location. Hence, the thesis suggests predicting the next location based on markov chain which are built on states representing user significant places based on his/her visits.

The system model for next location prediction model contains several steps. The process takes GPS trajectory points as input and processes them to create markov chain model on states. The intermediate steps are stay-point detection, state formation, time-slotted states creation and state transitions.

The flow chart depicted in Figure 1 explains how the model works. The input GPS coordinates data with the date and time information is fed to the algorithm. The algorithm keeps collecting the GPS points unit the end of a time-slot which is depicted using the “hour change” decision block. After the end of the time-slot, several steps are performed on these points from the previous hour. The several steps are:

* Detect stay-points (also detect the start or end of the trajectory.)
* Adjust leaving time from one location and arriving time to another location based on speed and distance between them.
* Group the stay-points to form states.
* Create time-slotted data from the states.
* Apply Markov chain for the data available.

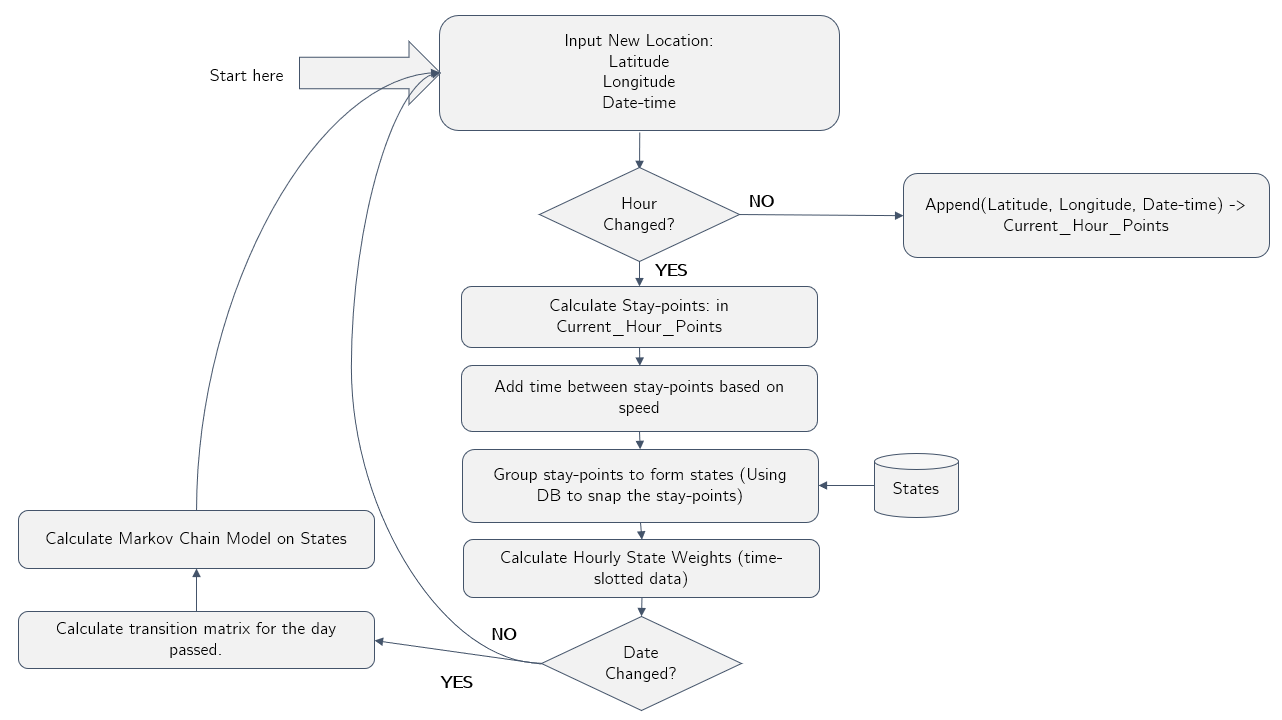


Figure 1 Design Flow-chart

Each of these elements are explained in detail in the further chapters.

## Components

In the model, the location data is input as an online GPS location data on a mobile system. This is to simulate how user shares the location data with other Location Based applications like Google+, Facebook, etc. These locations are sampled and the noise (travelling locations or short stay locations like post-office visit) are removed. This makes sure that the markov model is built on stable and longer stayed locations which are later formed as states. These states can represent many different locations with different semantic meanings like “home”, “work”, “favorite restaurant”, “gym/club”, etc. These states are recorded everyday basis and distributed on time-slots on an hourly basis. This is the assignment of the temporal feature on top of the location data. The hourly weighted or time slotted data is then used to form the markov chain.

These location coordinates are read from Geolife dataset user files in an online manner. This location data has latitude, longitude, date and time information along with other information. This location coordinates are read in an online manner to extract the locations where the user has spent more time. These locations are called stay-points sp. The extraction of stay-points is stored as {sp1, ap2, … spn}. These stay-points are the significant places for this user which has semantic meaning behind the location coordinates specific to the user.

Once the stay-point is extracted, we snap these stay-points to states which forms st = {st1, st2, …stn}. The states st are simply combining the similar stay-points based on their geographical distance from each other. If there is already a state existing, which is close-by to the new stay-point location, it is snapped to the existing state. If a new location stay-point is found, a new state is formed. Once we have the states, we create time-slotted states for each hour as shown in Figure 2.

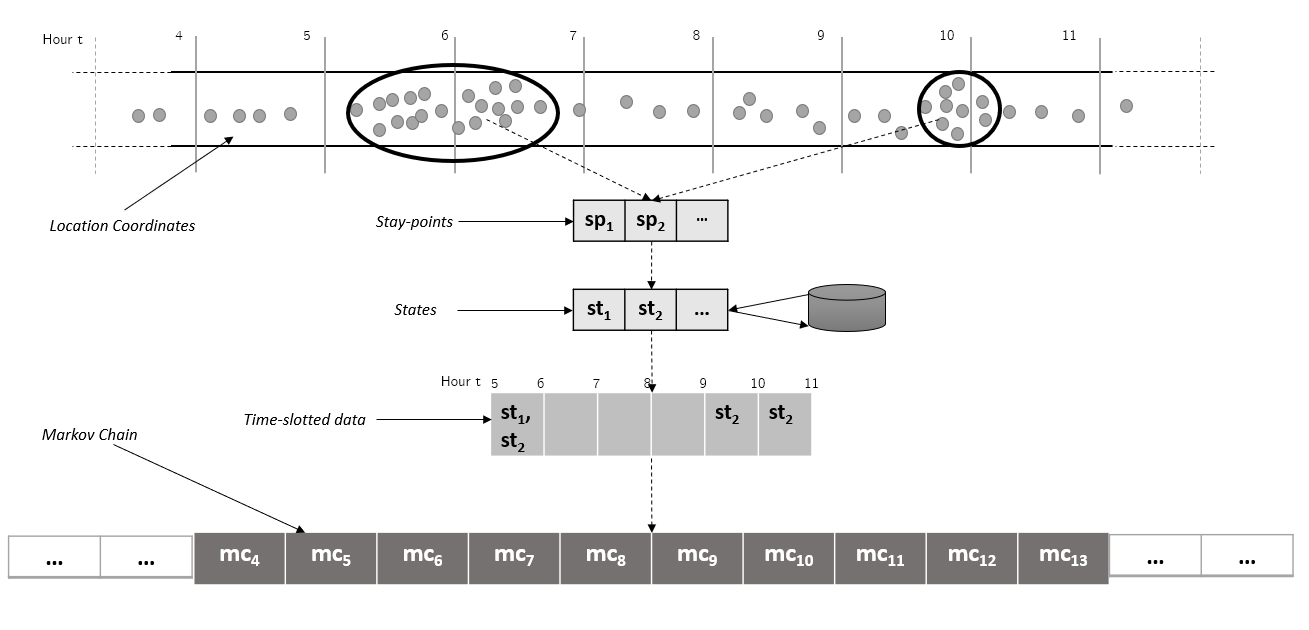


Figure 2 GPS coordinates to Markov Chain Model

Once the states are formed, the time-slotted markov chain on these states are formed. The markov chain holds the probability of transitioning from one state to all the other state (including self) from one time slot to the next. The Figure 3, on the left, depicts two states st1 and st2 markov model. This model involves 4 probabilities *p1* to *p4*. Each of these probabilities represents the transition probabilities from one state to another. For instance, *p1* is the transition probability from st1 and st2. Figure 3, on the right, depicts a third state added to the model st3, which in turn, increase the probability count from 4 to 9. This explains, how the complexity and the computation of the model increase as the number of states increases in the model.

This markov chain model is then used to predict the movements from st1 to all the other states at any given time-slot, based on the probabilities calculated. The similar probabilities are calculated every time a new state is added as depicted in Figure 3 for a new state st3.

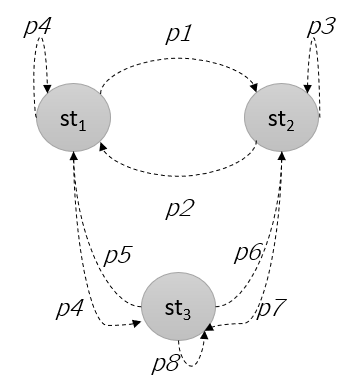
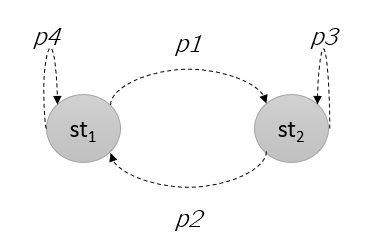


Figure 3 Markov chain on states

## Hypothesis

The location-based services rely on the assumption that the location is shared for longer periods. The model behaves poorly if the location data is shared very rarely by the user. The occurrences of a few popular locations like “home” and “work” for a user will be more, compared to other location. It is very often that the user stays at “home” location during the night hours and spends more time at “work” location during the day. Of course, there could be night-shifts, but then the duration of stay at “work” location, which usually ranges from 8-9 hours, can help to make the right indications. The “home” location is also often the one which has occurrences during weekends or public holidays. These indicators help us in marking the “home” and “work” location while analyzing the data.

However, the location data is sometimes not available to be shared or just turned off. For instance, there is no internet in a skyscraper “work” location or on a cloudy day or the location is completely turned off as soon as the user has entered the “work” location. The next location input, after few hours, is again the “work” location when the user comes in a network coverage area. The first and the last known location help us to fill the missing information during the few hours based on the distance and time difference between the two locations. In this case, we can assume that the user stayed at the “work” location for the missing data.

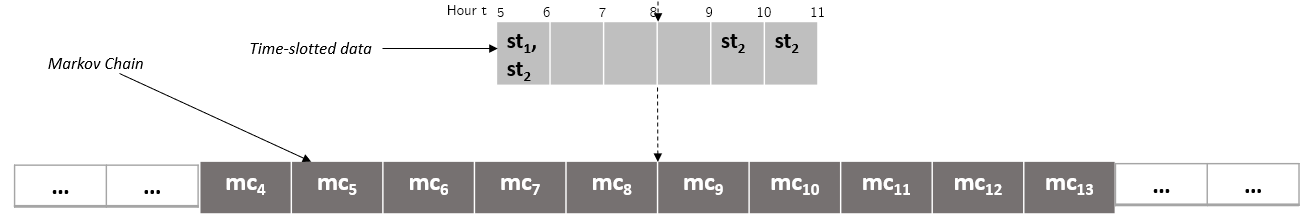
# **Proposed Concept**

In this thesis, a location prediction model is proposed. The idea is to aware the users about the privacy risk, while sharing the location with applications on mobiles and computers. This chapter briefs about the usage of markov chain model for location prediction.

## 4.1 Markov Model for Location Prediction

The markov model for location prediction is formed based on states. The assumption in this chapter is that the states are already created. The state formation from raw location coordinates is explained in further chapters.

The states are the significant locations for a user which are extracted from raw location coordinates. The symbolic meaning behind these states are, for instance, “home”, “work” or “gym”. The markov chain holds the probability of transitioning from one state to another. The transitions are recorded for each hour of the day as shown in the Figure 4. The time-slotted state data is used to derive transitioning probabilities for markov chain model.

Figure 4 Markov chain derived from state

The markov chain model is formed from time-slotted states data. Consider an example as shown below in the Figure 5. There are three states st1, st2 and st3 which exists between hour 4-6, 5-9 and 9-11 respectively.

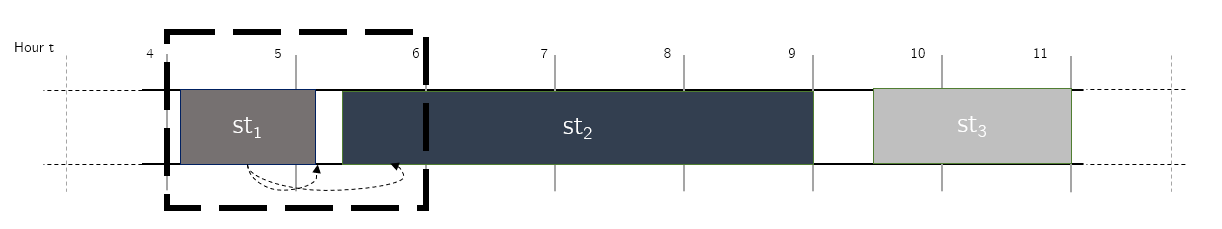


Figure 5 State transitioning

Let us consider the example of transitioning from hour 4 to hour 5 as depicted in Figure 6. In this example, the state transition from hour 4 to hour 5 is from st1 to st1 and st1 to st2. It is important to mention that the hourly weights are normalized before the markov chain is calculated.

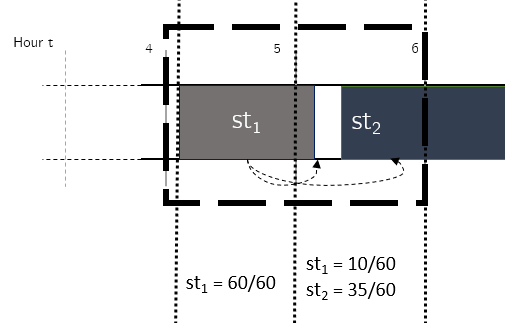


Figure 6 State weights in each time-slot

After normalization the states are shown in the Figure 7. This is done to smoothen the data in each time-slot.

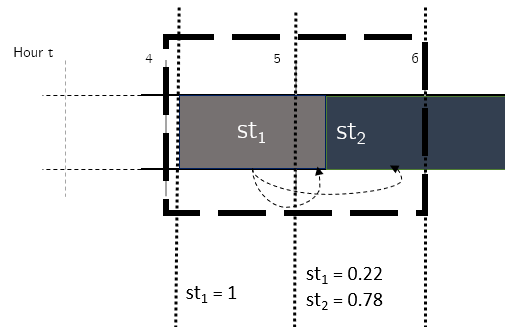


Figure 7 State weights normalized

The probability of transitioning from st1 to st1 from hour 4 to 5 is calculated based on the hourly weight of st1 in hour 4 and hourly weight of st1 in hour 5. Similarly, the probability of transitioning from st1 to st2 from hour 4 to 5 is calculated based on the hourly weight of st1 in hour 4 and hourly weight of st2 in hour 5. The weights vector w = {w1, w2} for hour 4 can be defined as w4 = {1, 0}, where the 1 represents the weight of state st1 and 0 represents the weight of state st2 in time-slot 4. A similar weight vector w for the next time-slot 5 can be represented as w5 = {0.22, 0.78}. The multiplication w4(transpose)\* w5 results into a matrix as represented in the Table 1. The table represents the transitions probabilities among states st1 and st2 from time-slot 4 to time-slot 5. The first row represents the transition probabilities from state st1 all the other states i.e. st1 and st2. Similarly, the second row represents the transition probabilities from state st2.This transition probability matrix represents markov chain model for the given time-slot and the states. The markov chain for transition from hour 4 to hour 5 is called mc5 which is depicted in the Figure 8.

|  |  |  |
| --- | --- | --- |
|  | st1 | st2 |
| st1 | 0.22 | 0.78 |
| st2 | 0 | 0 |

Table 1 State transition matrix

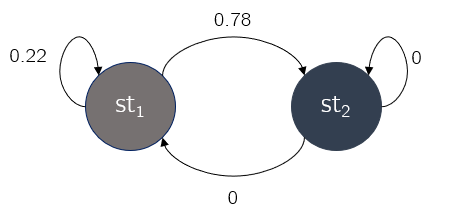


Figure 8 Markov model for two states

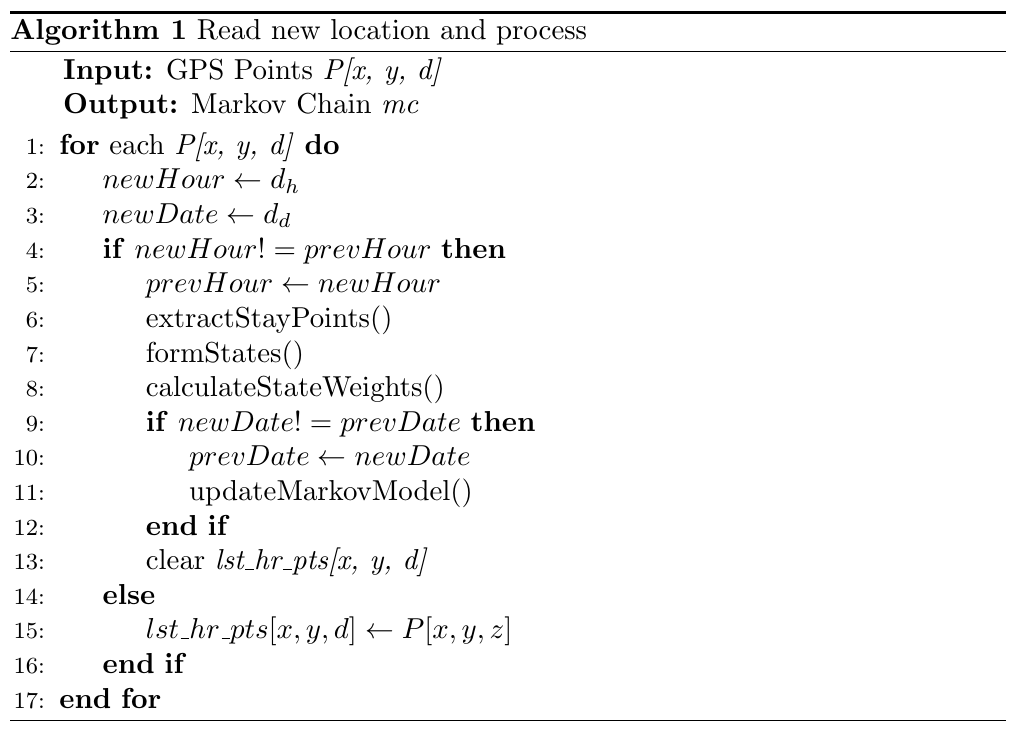
## 4.2 Algorithm

Using the markov chain model, the predictions are done. The predictions are based on user’s current location. Continuing the previous example, if the user is at location st1 at hour 4, there are 78% chances that the user will move to state st2 at hour 5 and 22% chances of staying at state st1 at hour 5. The probability of going to all the other places at from his/her current location, at this time-slot, is present in markov chain model.

The GPS trajectory points are received as input. For each new GPS Point *P[x, y, d]*, which contains *x* as latitude, *y* as longitude and *d* as datetime, the process is run. The transitions are recorded from one time-slot to the next. The first step is to detect the time-slot change. Until the time-slot is changed, the GPS coordinates are collected and kept as *lst\_hr\_pts[x, y, d]*. Once, the time-slot has changed or the next hour is detected, *lst\_hr\_pts[x, y, d]* are processed. There are several steps performed on *lst\_hr\_pts[x, y, d]*  as listed below:

1. Stay-point extraction: For every hour or time-slot, the extraction of stay-points extractStaypoints() is run creating the stay-points sp = {sp1, sp2,… spn}. The stay-points represents the location points which are stayed for longer durations.
2. States: From the stay-points sp, the states st = {st1, st2, … stn} are formed using formStates(). This process combines the geographically close-by stay-points to one state. Similarly, several states are formed from stay-points. These states are used later for markov chain model. The states’ st represent “home”, “work” and other important visited places.
3. State weights: The next step after state formation is to calculate state weights w = {w1, w2, … wk} in each time-slot. A state weight w1 represent the minutes/60 the state st1 has spent in this time-slot. These states are later normalized in each time-slot.

This process (A, B, C) is repeated for the entire time-slot. The same steps are repeated for each time-slot and the data is accumulated. This means, the same locations visited are snapped to the same state id. For example, user has been at location “home” during the early hours of the day. If location “home” is visited again during the day, it will be extracted as a new stay-point in step A. Since “home” location was visited already during the early hours of the day, it will be snapped to an existing state in step B. In algorithm, this will be represented be a numeric id, but, it has a semantic meaning “home”. And, the last step C is used to calculate the weight, or the time spent at “home” at this new time-slot. This process ensures that we keep snapping the known locations with the same ids. Once the day is changed, the markov model *mc* is created using the state weights w. The individual algorithms of stay-point detection, state formation, state weight calculation and markov chain model creation is explained in chapter 5.



# **Implementation**

## 5.1 Variables Used

The Table 2 is to provide an overview of the variables used in the further sub-sections. The variables are used in algorithms and in explanation. The list of variables covers majority of variables used in further sections, but it is not exhaustive. Few new variables are introduced and explained in the further section text for clear understanding of concepts.

|  |  |
| --- | --- |
| Variable | Description |
| P[x, y, d] | point is a tuple:  (Latitude, Longitude, Datetime) |
| dh | Hour from d datetime |
| dd | Date from d datetime |
| th\_tck | Threshold time for tracking GPS location data |
| th\_d | Threshold distance for staypoints |
| th\_t | Threshold time for staypoints |
| spi(x, y, ds, de) | ith Staypoint  (Staypoint Latitude, Stapoint Longitude, Start Datetime, End Datetime) |
| sti(x, y) | ith State  (State Latitude, State Longitude) |
| lst\_hr\_pts(x, y, d) | Last hour GPS points  (Latitude, Longitude, Datetime) |
| w | State hour weights |
| mc | Markov Chain for sti to sti+1 for time slot h to h+1 |

Table 2 Variables used in algorithms

## 5.2 Stay-points

Stay-points are those significant places where user spent significant time. A GPS trajectory is the path taken by the user where the GPS points are continuously received i.e. every 5-10 seconds a coordinate is received. A trajectory can end for several reasons, for example, if the user turns of the phone or location sharing or the user enters a no-network area. Stay-points are any points which are stayed by the user during the user GPS trajectories or it is the start or the end of the GPS trajectory. For example, if the user starts his/her trajectory at home location, the home itself is a stay-point. Now he moves towards work, but he visits a cafe in between for breakfast. The cafe is also, a stay-point and then he finishes his trajectory at work, where work is again a stay-point.

The places like cafe in this case is identified using distance and time-based clustering. Distance and time-based clustering work best in case of location data. This clustering the not so complex and can be run on a mobile device as a background process. The clustering has two thresholds, one for distance (th\_d) and one for time (th\_t). These threshold help determining the stay-points in an online fashion. The location points within the radius of distance threshold (th\_d) and time spent at this location greater than or equal to the time threshold (th\_t), is regarded as a stay-point. For example, a set of points within 200m of radius and total duration of stay greater than 20 minutes, can be regarded as a stay-point. In the example above, the café location will be a stay-point only if the stay is greater than or equal to the time threshold (th\_t). This help to remove noise, like travelling GPS coordinates or short stay locations. Hence, only significant locations from the trajectory are extracted and noise is removed.

Trajectories are continuously received GPS points. The gap of time greater than tracking time threshold (th\_tck) between two GPS points breaks the old trajectory and starts a new one. This means, if the location coordinates are received continuously for few hours and then the location coordinates are stopped, this trajectory has ended. As soon as the new location coordinates are started, a new trajectory has started. Note that the stay-points are found within the trajectory with time and distance clustering algorithm. The second type of stay-points also exists. These stay-points are the locations where user has ended or started his/her trajectory. For instance, the user has entered his work location and now he/she do not share his/her location. If the next shared location is after a threshold tracking time (th\_k), the last trajectory is broken and a new has been started. This means, if there have been no new location coordinates received for a given time, the last point in the previous trajectory is added as a stay-point and so is the next point received consequently in the next trajectory. This ensures that the important locations are not missed even if the location data is not present. Since, the location data is often turned off at stable locations like “home” and “work”, this algorithm makes sure that these locations are not missed for future user movement analysis.

After the collection of stay-points, the stay-points entering and leaving time is recalculated. This is done to estimate the time of leaving a stay-point and the time of entering the next stay-point. Let us understand this using an example as depicted in Figure 9. In this example, we consider the distance threshold th\_d to be 200 meters or 0.2 kms. For instance, user is reported to be at “home” location at 7am and then the next stay-point is found to be “work” location at 8 am. The missing data between 7am and 8am can be for many reasons, for instance, no network coverage or user has purposefully turned off the location sharing. For instance, distance between these two locations is x kms, which is easy to calculate as the “home” location coordinates and “work” location coordinates are known. Let’s consider the distance to be 6 kms between “home” and “work” location. The time difference t between the two points “home” and “work” is also known, which is 1 hour in this example. This information helps us to estimate the actual leaving time from “home” location and actual arriving time at “work” location. The speed of user (spd) can be calculated as (x kms / t minutes) i.e. (6/60) kms/mins or 0.1 kms/mins. Now, the delta time is calculated as minimum (th\_d, distance between)/spd i.e. minimum (0.2, 6)/0.6 or 0.33 minutes. This delta time is added in the departure time and subtracted from the arrival time. So, the estimated departure time at “home” location is 7am + delta time i.e. 7am + 0.33 minutes and the estimated arrival time at “work” locations is 8am – delta time i.e. 8am - 0.33 minutes.

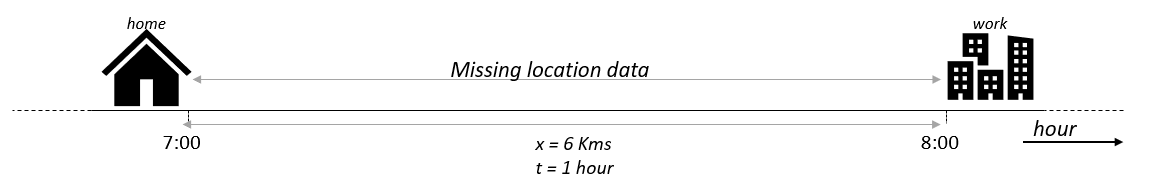


Figure 9 Example 1 of missing data

But there could also be the case where the user is no travelling or moving from one location to another, but rather he/she stays at a location after some missing data. Consider the example shown in Figure 10 where user has shared his/her location at “home” at 18:00 and now the location data is not shared for some reason for the next few hours. The next location shared is again “home” location at 06:00 the next day. The missing location data is most likely the “home” location for the entire time slots between 18:00 on this day till 6:00 on the next day, Since the distance between these two points is 0, the speed *spd* of travel will also results in 0 km/hour. Now, the estimated time of being at “home” location is recalculated. The time difference between the two known points is 12 hours. The time of leaving is calculated as (know leaving time + time difference / 2) and the time if arriving is calculated as (known arriving time – time difference / 2). The time of leaving “home” location is recalculated as (18:00 + 12/2) i.e. at 00:00 on this day and the time of arriving at “home” location for the next day is (06:00 – 12/2) i.e. at 00:00 on the next day.

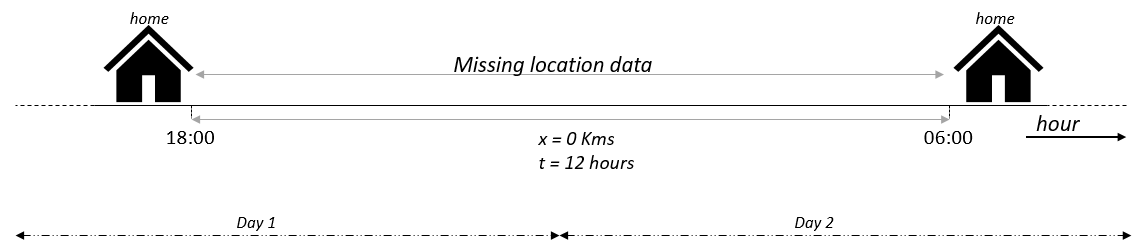


Figure 10 Example 2 of missing data

### 5.2.1 Algorithm

The stay-points are extracted from raw points to remove the noisy points. The noisy points could be travelling with the bus or train or a short stop at the letter box. The stay-point extraction is the process of extracting longer stayed locations from raw GPS trajectories. The Figure 11 shows the transition from “home” to “work”. In this case, both “home” and “work” are extracted as stay-points.

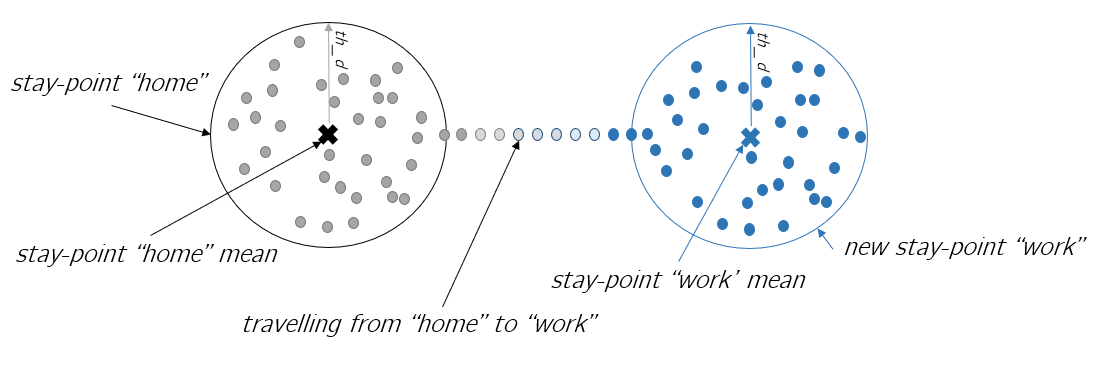
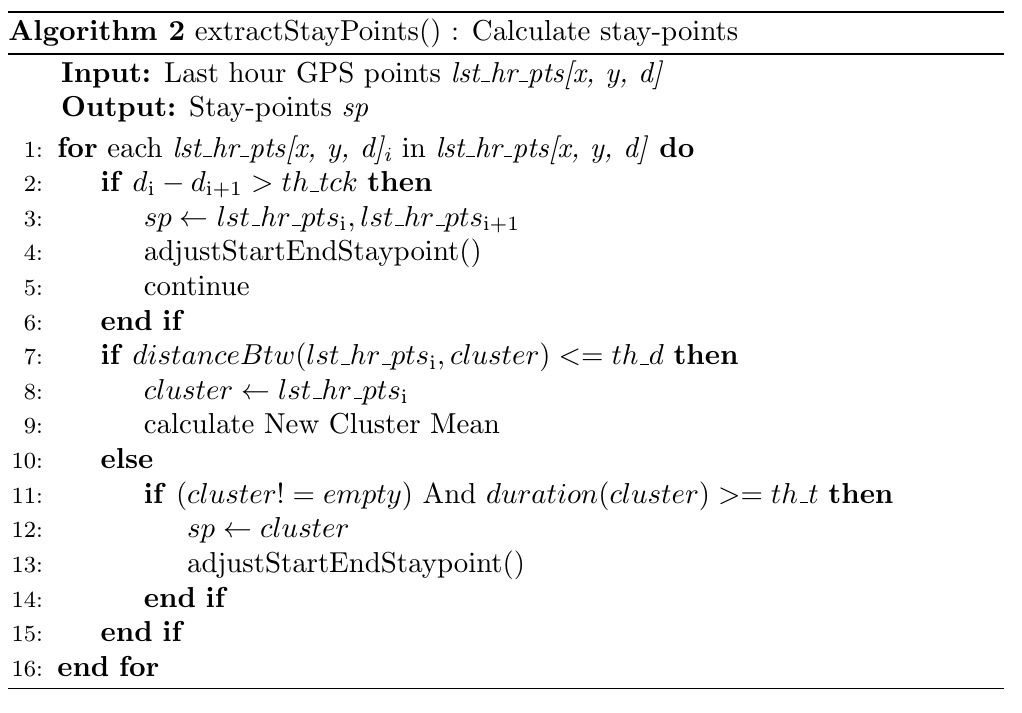


Figure 11 Extracting stay-points from GPS Trajectories

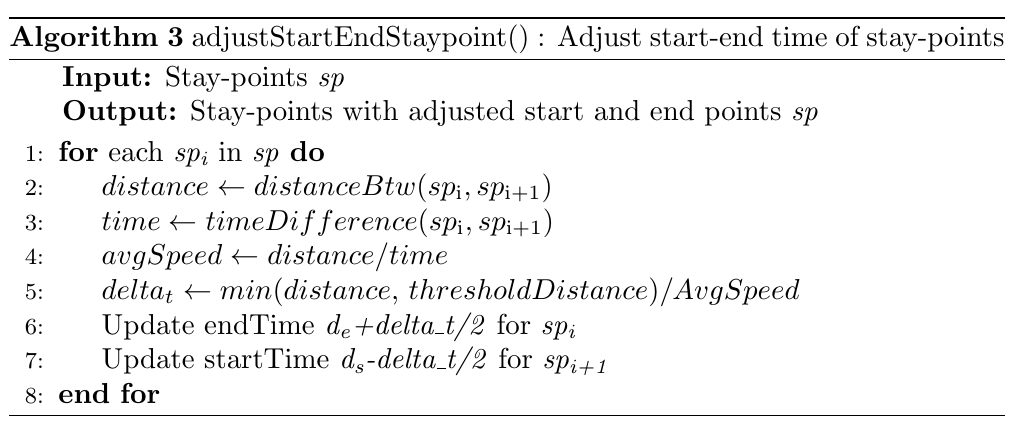
The extraction of stay-points takes lst\_hr\_pts as input and generates sp = {sp1, sp2,… spn} as output. The algorithm cluster the points within the radius of stay-point distance threshold th\_d for a minimum duration of time th\_t. The selection of distance and time threshold is very important. If the distance threshold value is too large, the mean of the stay-point locations will be a confusing location on the map. If the time threshold value is very small, a lot of insignificant locations will be added as stay-points.

A new location from lst\_hr\_pts is added to the cluster if the distance between the new point and the cluster mean is less than or equal to the distance threshold th\_d. Every time a new point is added to the cluster, a new mean of the cluster is calculated and the process repeats. If the new point from lst\_hr\_pts is moving away from the cluster mean, then the point is not added to the same cluster. This means that if the distance between the mean of the cluster and the new point from lst\_hr\_pts is greater than threshold th\_d, then the new point is not added to the cluster. At this point, the cluster duration is checked. The cluster duration is nothing but the largest datetime – smallest datetime from the cluster elements. If the cluster duration is greater than or equal to th\_t, then the cluster is added as the stay-point sp with latitude and longitude as cluster mean, otherwise the cluster is not added as a stay-point. The new point from lst\_hr\_pts is also added as a stay-point if the difference of time between the new point and the previous point is greater than time tracking threshold th\_tck. This is to ensure that if the GPS points are not received for a long time, we add the last and the new point as a stay-point assuming the end of the previous trajectory and the start of the new trajectory.



The stay-points are often not continuously distributed over time. Consider a scenario where a user is at “work” location till 9 am. After entering work, either user decides not to share the location or enters a no-network coverage area. The next stay-point detected is “work” location at 1 pm when user came out for lunch from the building. The time of stay at “work” in this case till 9 am is misleading as user stayed at this location till 1 pm. Hence, once the stay-points are collected, we adjust the starting time and the leaving time of each stay-point.

This is done by comparing each stay-point in sp = {sp1, sp2,… spn} to it’s very next stay-point in sp. Now, the distance and time difference between the two stay-points spi and spi+1 is calculated. Using this the average speed of travel can be easily calculated which is distance/time. The delta time delta\_t is calculated as division of minimum of distance between spi and spi+1 to the average speed. Now we add the delta time delta\_t to spi to change leaving time at the spi location and subtract delta time delta\_t for spi+1 to change the entering time at location spi+1.



### 5.2.2 Implementation Result

The algorithm is applied to Geolife dataset. The user files are read in an online manner to simulate the GPS location points received on a mobile device. The stay-points found for user 1 for November 2008 as shown in the Figure 10. The trajectory is shown with the green line and the red arrows indicate the stay-points. These stay-points represent the locations with semantic meaning behind it like “home”, “work”, “restaurant”. This clearly depicts that a lot of noise in the trajectory data is removed at this step and only the significant stays are extracted.

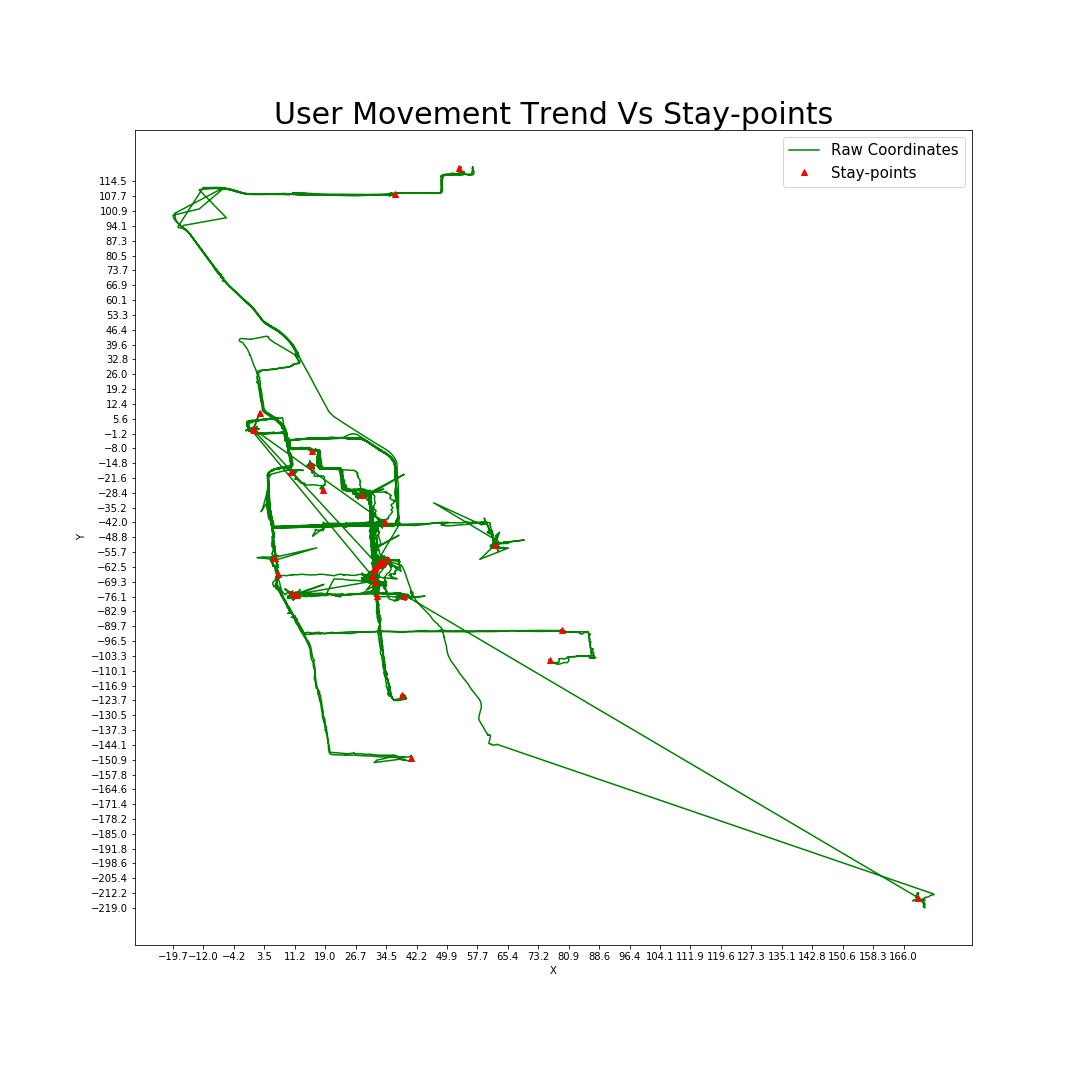


Figure 12 User 1 raw trajectory data vs stay-points extracted

## 5.3 State Formation

A state is formed using a group of stay-points represented as S = {st1, st2, …stn}. Each state has a semantic meaning like “gym”, “restaurant”, “home” or so on. The states are formed to join the stay-points which are geographically close-by as one state. The states are found using a distance threshold for states. All the stay-points within this threshold distance radius is grouped together as a single state. This is called snapping stay-points to the states. This is depicted in the Figure 11. The stay-points are snapped to states to form st1. The first stay-point between hour 5 and 7 is assigned to a state st1. When the second stay-point between hour 9 and 11 is found to be geographically close-by to the state st1, it is also snapped to the same state. The mean of latitude and longitude all stay-points forming the state st1 is stored. This makes sure that if a known location is visited after few days and is a stay-point, then it should get the same id as the known location before. Imagine a case where the user visited a restaurant last Monday and he visits the restaurant again the next Monday. In this case, the restaurant visits during next Monday should get snapped to the already existing state from last Monday. Finally, markov Chain model is applied to the states.

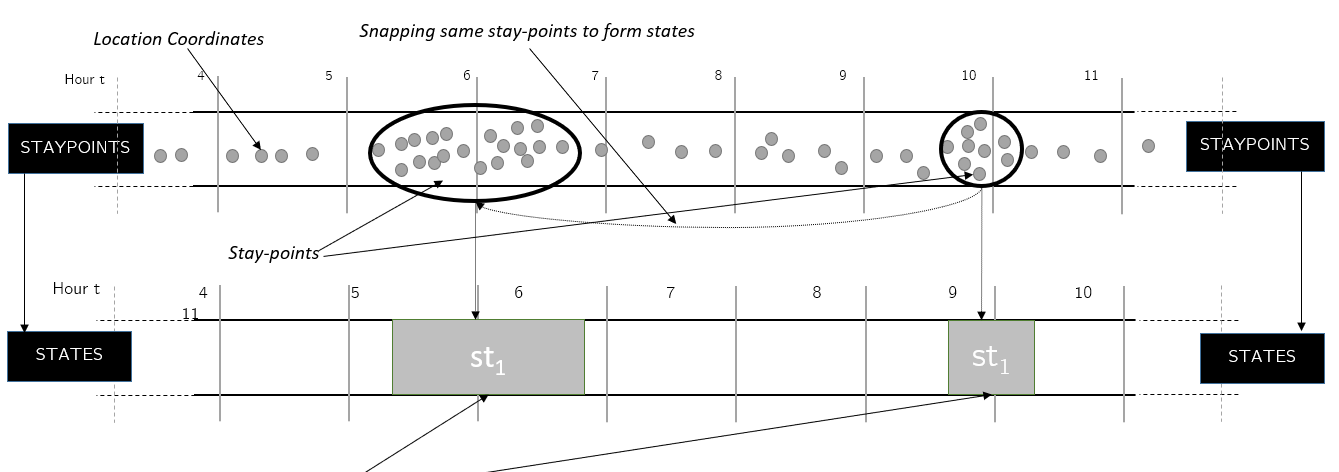


Figure 13 Snapping stay-points to states

### 5.3.1 Algorithm

Once the stay-points sp are extracted for this time slot, the states are formed. Each stay-point in sp = {sp1, sp2,… spn} is compared with every other stay-point in sp. If the distance between the two stay-points is less than the distance threshold th\_d, then the two stay-points are combined to form a state st.

The stay-points are snapped to an exiting state with an exception. The Figure 12 depicts the drifting problem. The current mean of state sti is marked with **×**. The addition of the new stay-point spj will make the mean of the state shifted denoted by **×**. The new mean of the state sti will throw some of the existing points from left out of the state radius. Hence the new stay-point spj is added to a new state in this case. Hence, the idea is, while adding the new stay-point to an existing state, a check is done. If all the existing stay-points stays within the radius of the new state mean, then the stay-point is snapped to the state st, otherwise a new state is formed. In other words, if any of the existing stay-points contributing to the state formation is moving out of the state radius, then the new stay-point is not added to this state and a new state is formed. This is done to avoid the drifting mean problem.

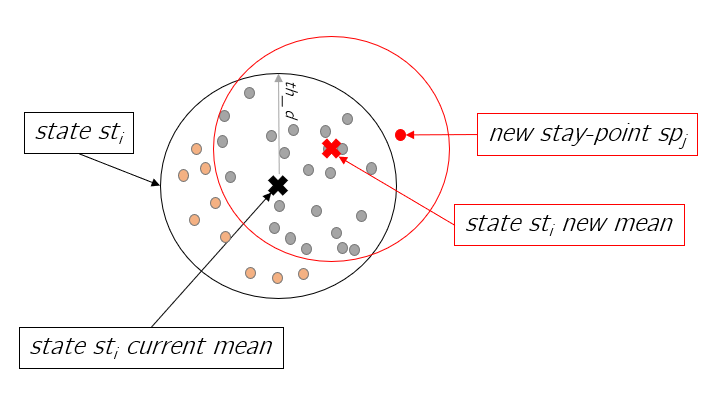
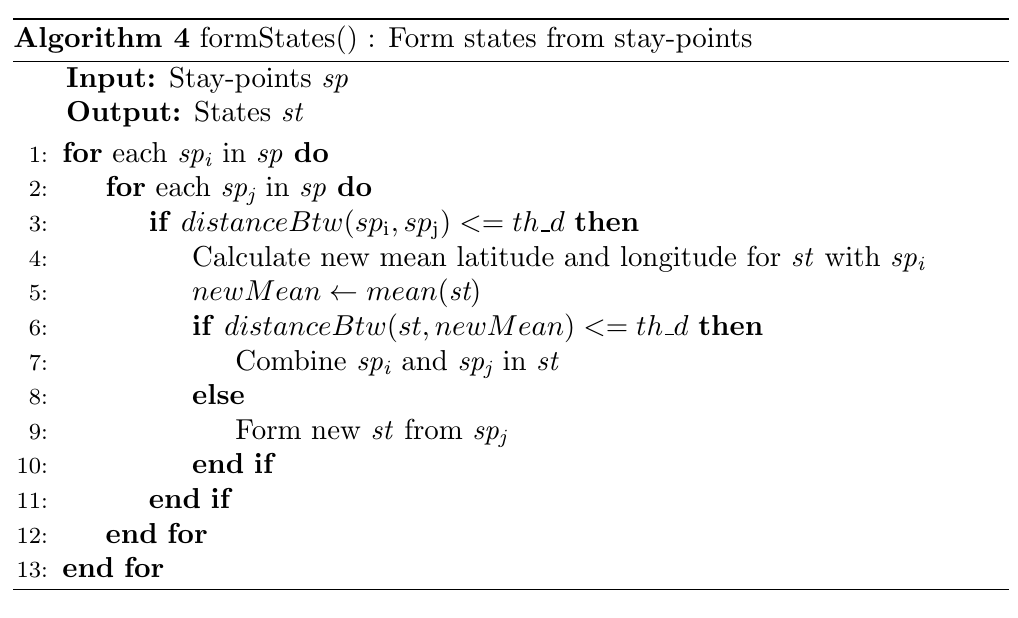


Figure 14 Drifting problem while snapping stay-points to the states



## 5.4 State Weights

The hourly weights form the time-slotted state data. The states can appear in different time-slots. The set of states st = {st1, st2, …stn} are assigned with weights at each time-slot to form w = {w1, w2, …stn} for timeslot t. The weight w1 represent the hourly weight of state st1 in one time-slot and so on. The hourly weights are the ratio of minutes spent at a state to the total minutes in one hour. The weights are then normalized to smoothen the data in each hour slot. As shown in the Figure 13, the st1 hourly weight is calculated for 5-6, 6-7, 8-9 and 9-10. The state st1 location is recorded for 40 minutes between hour 5 and 6. Hence the weight for st1 is calculated as 40/60 = 0.67. Similarly, the weights are calculated for each time slot, for all the states. In this example, we have only state st1. After the hourly weights are calculated for the state, the weights are normalized in each hour before they are used for markov chain calculation. This is also shown in Figure 15.

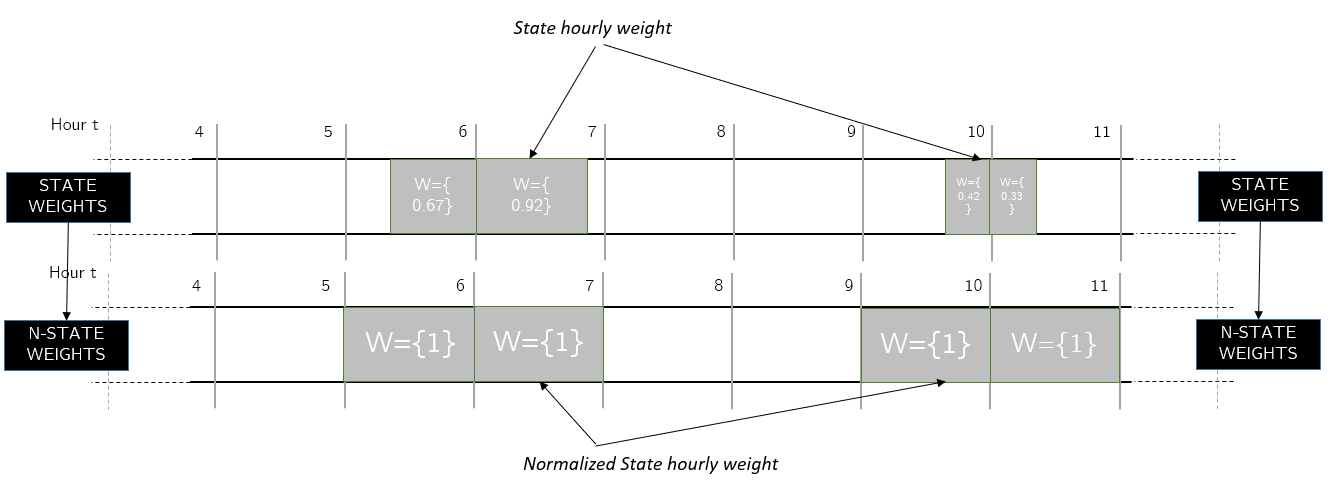
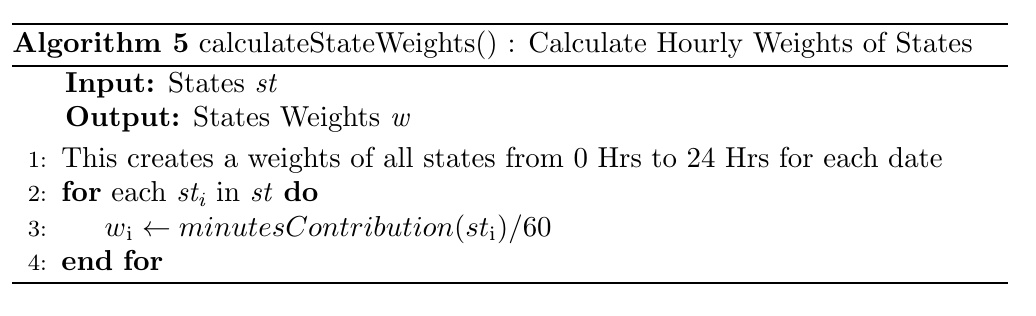


Figure 15 Normalizing state weights

### 5.4.1 Algorithm

Once the states st = {st1, st2, …stn} are formed, the next step is to calculate the weight w = {w1, w2, … wk} of each state in this time-slot. It is important to know that there can be many states in one time slot. The weights w represents the probability of a state in this time-slot. Since states itself carry some semantic meaning like “home”, “work” or others, the weights w represents how long did the user stayed at “home” in this time-slot. Since in our case, the time-slots are divided into each hour, this means time slot t to t+1 has one-hour increment. The algorithm reads each state in st and calculates the weight in this time-slot by simply dividing the minutes contribution of each state with 60 minutes.



After the weights w calculation, the weights are normalized for each hour. This smoothens the data in each time slot. This is done by simply dividing the weights in weight w = {w1, w2, … wk} with the sum of all the weights in one time-slot. The results are shown above in the figure. The state weight w1 is normalized for hours 5, 6, 9 and 10.

### 5.4.2 Implementation Result

The hourly weights calculate the time-slotted data which is used as the base for markov chain model.

The Geolife dataset date and time are represented in GMT, hence to have the correct visualization, the date and time must be adjusted to the local time in the trajectory data. The Figure 14 below depicts the hourly weighted state data for user 1 for November 2008. The x-axis represents each hour of the day from 0 to 24 and the y-axis represent the days. Each rectangle depicts a state where the width of the rectangle represents the weight of the state in the corresponding hour. For example, state 1 is the first state between 9 am and 10 am on day 0. The distribution of the states over the hours and days makes some hints about the semantic meaning behind the locations. For instance, state 3 is most likely user’s home location as on most days’ user is at this location from 9 pm till next day 7 am and state 6 is most likely user’s work location as on most days’ user is at this location from 8 am till 8 or 9 pm. There are many other locations like 13, 22, 2, 16 and so on which could represent the supermarket, shopping mall, fitness club and so on.

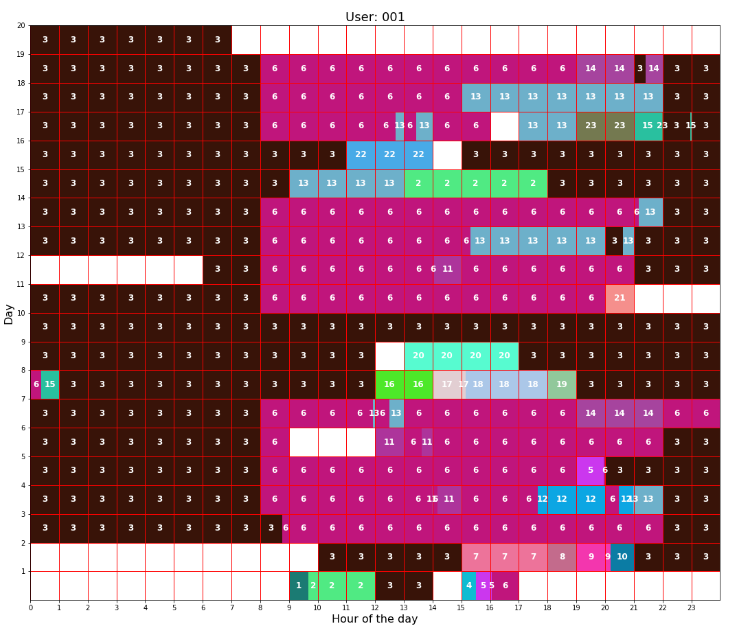


Figure 16 User 1 time-slotted data for the November 2008

## 5.5 Forming Markov chain

Once the time-slotted data is created and normalized, the markov chain is build. The model of markov chain is created to calculate the probability of going from one location to another. The Figure 17 shows on a map how the markov chain looks like. There are four important locations marked on the map. The markov chain model will contain the transition probabilities from one of these important places to all the other places (including self) as depicted in the Figure 17.

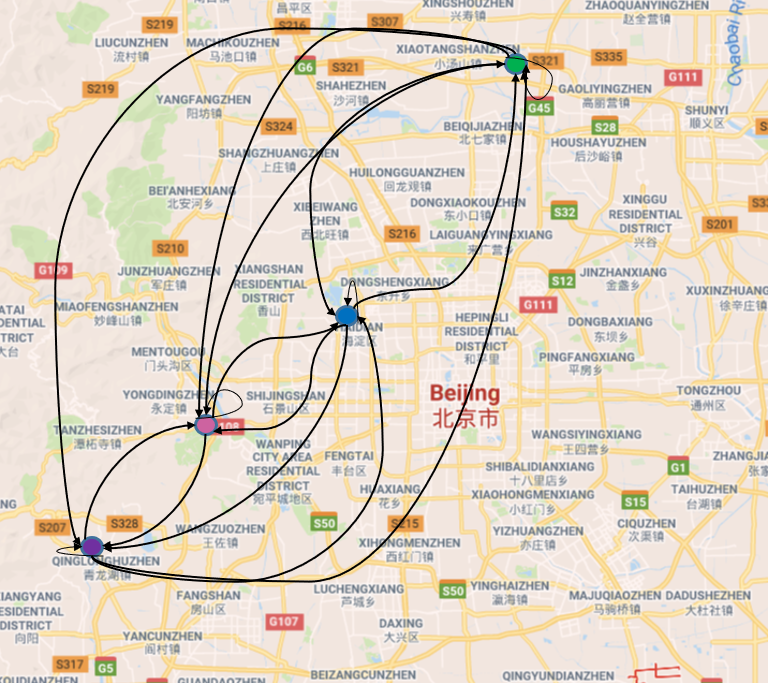


Figure 17 Markov chain example on a map

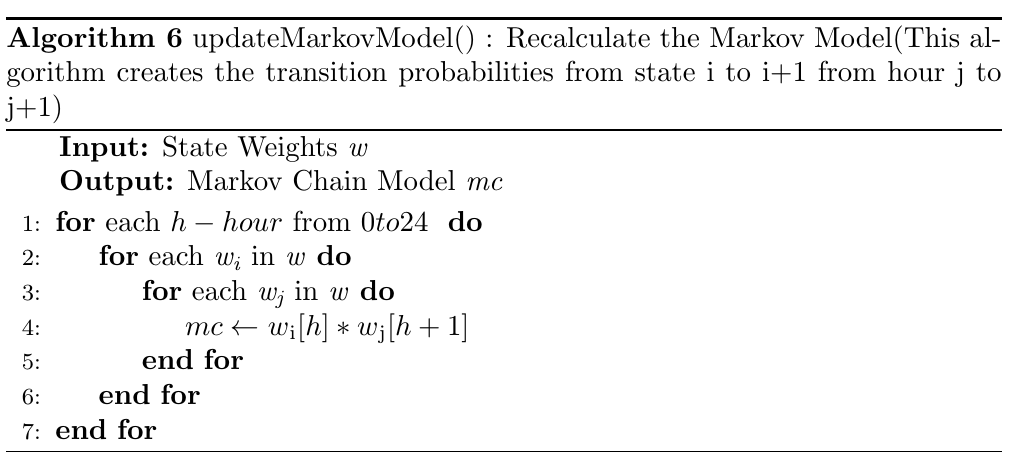
A set of n states’ S = {st1, st2, …stn} where each state represents a significant place or place of interest, with weights Wt = {w1, w2, …stn} at hour t and with weights Wt+1 = {w1, w2, …stn}t+1 at hour t+1 is used. The weight vector Wtand Wt+1 are multiplied to calculate the transition probabilities from each state in set st to every other state in st from time slot t to t+1. This transition probability is a theoretical measure to determine the chances of transiting from one place to another from one time slot to another. A transition can be a self-transition, for instance, st1 to st1 or a transition to another state, for instance, st1 to st2. These transitions represent human mobility from one important place like “work” to another like “gym”. This transitions are recorded for each time slot.

The markov chain built is standard and hence the probability only depends on the current location and not the locations before. This means a transition from “home” to “work” from time slot t to t+1 depends only on time slot t and not on any time slots before t.

### 5.5.1 Algorithm

After state weights w = {w1, w2, … wk} are calculated, the next step is to build the markov chain model. The model is updates after each day to keep a track of all the new locations user has visited in the previous day and update the mobility pattern. The continuous update of the markov chain model helps to track the changes in user behavior. For instance, user may change the work location, working hours, join a new gym or move to a new city.

The markov model mc contains the probability of transitioning from one state sti to another state stj for time slot t to t+1. This information is stored for each state in st = {st1, st2, …stn} transitioning to every state in st for all time slots. The time-slots in this case represents the hour of the day. For each hour of the day, the weights wiis multiplied with the weight wi+1 for hour i to i+1. It is important to understand that a transition can be self-transition i.e. moving from sti to sti to another state stj.



The markov chain model counts the transition from one state to another for each time slot. Consider the figure above for hourly weights for user 1 in November 2008. For instance, the transition from state 3 i.e. “home” to state “6” i.e. “work” can be recorded always from hour 7 to hour 8. These transition between the states are counted for each day, from one time-slot(hour) to next time-slot(hour+1). This is performed by simply multiplying the weights for each state in t time-slot to the weights of each states in t+1 time-slot. Consider the Figure 16 as an example. The weights of state st1 and st2 for hour 4 and 5 are depicted in the figure. There is a transition from hour 4 to 5 from state st1 to st1 and another transition from st1 to st2. The corresponding state weights represents the normalized duration of stay in that time slot. Hence, the Wst1 is 1 as between time-slot 4 and 5, the entire stay is at state st1. Wst1 is 0.33 (20/60) between time-slot 5 and 6 as the stay at state st1 between 5 and 6 hours is 20 minutes. Similarly, the weights of the other states are calculated.

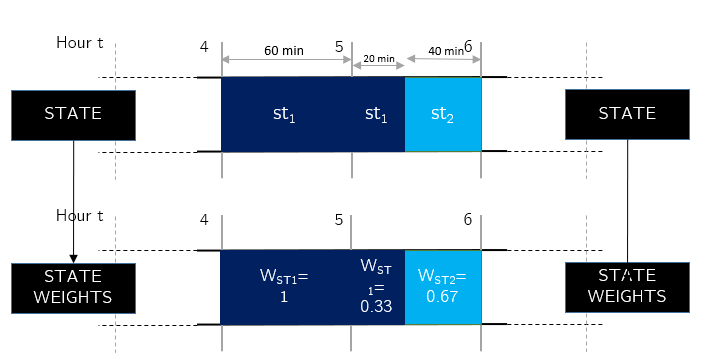


Figure 18 Forming time-slotted data from states

Once the state weights are calculated and normalized, the same weights are used to build the transition matrices for each time-slot. The weights are multiplied from one time slot to another to create the transition matrices. The Figure 17 shows the transition matrices for time-slot t, build for the previous example. The rows and the columns are representing each state and each cell represent the transition from one state to another.

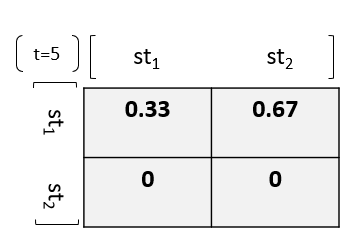


Figure 19 Transition matrix for state 1 and state 2 for time-slot 5

Once the transition matrices for each time-slot are calculated, the markov chain model is build. In markov chain model, transition probabilities from one state to another is calculated, which is done using the transition matrix. Each row in the transition matrix represent the transition from one state to all the other state. In the Figure 19, the first row represents the transition from st1 to all the other states i.e. st1 and st2. To calculate the probability of transitioning from st1 to all the other states can simply be calculated by dividing each cell elements with the sum of the row. If the sub of the row is zero, this indicates that we have no information about this transition. In this case, equal probabilities are assigned to all the states. The Figure 18 shows the probabilities calculated for markov chain for the same example as above.

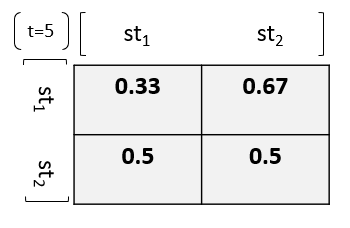


Figure 20 Transition matrix replacing no transitions i.e. 0's with equal probabilities

It is important to understand that all the 0 transitions probabilities are replaced with a small probability. This is done to ensure that the chances of transitioning from known states is never 0. The Figure 19 explains the process of replacing the zero probabilities. In the first step, state weights are multiplied to get the state transition weights. In step 2, 0 is replaced by a 0.00001. In step 3, the cell value is divided by the sum of the row.

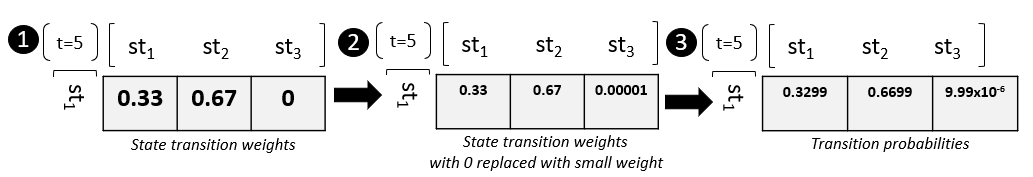


Figure 21 Steps followed to smooth the transition matrix

### 5.5.2 Implementation Result

The markov chain model is implemented on Geolife dataset as explained above. The Figure 20 depicts the markov chain of user 1 for November 2008 for time-slot hour 8. The columns headings and the row headings indicate several state ids from 1 to 23. Each cell represents a probability of transitioning from one state to another. Red indicates the very low probability. For example, the row one indicating the transition from state 1 to all the other 23 states, are equally probable. The only two exceptions are the transition from state 3 to state 3 and transition from state 3 to state 6. The transition from state 3 to state 6 is more probable (0.66) than the transition from state 3 to state 3 (0.34).

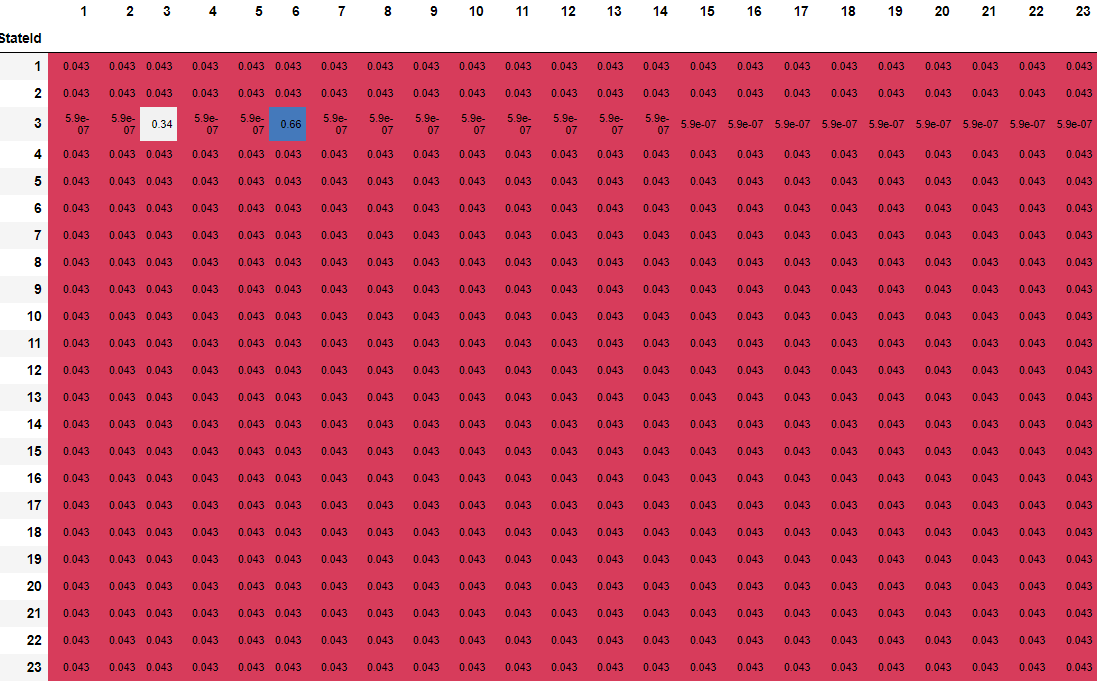


Figure 22 Markov chain model for user 1 for November 2011 time-slot hour 8

Similarly, there exists a matrix for each time-slot. Since we have divided the time-slots in hourly manner, there exists 24 time slots from 0-23 Hours. Hence there exists 24 different matrices, consisting transition probabilities from states-to-states. These Matrices together forms markov chain. This can be used for future prediction. In the Figure 22, the transition probability from state 1 to state 2 from time slot 7 to time slot 8, is 0.043. In other words, if user is at location state 1 at 7 am, there are 4.3% chances that he will move to location state 2 at 8 am. For instance, from state sti in time slot t, the first step predictions can be done in time slot t+1. For this, the matrix for time-slot t+1 and row with state sti is used. The entire row gives the probability to reach all the other states. The highest probable state is the predicted next state. The process continues for time-slot t+2 and onwards.

## 5.6 Path Prediction

The path prediction can be done using markov chain model. The model is memoryless and hence depends only on current location. This means, the prediction is done based on the user’s current location and is independent of this location before. The path prediction is to inform the user about his/her predictability. The idea is to predict the several paths that user may take from a known location at particular time-slot. For example, user’s current location is known to be at “work” at 8 am. Knowing his/her location data for few weeks, the markov model is built. Now this markov model is used to predict all the locations in consecutive time-slots. One of the paths predicted could suggest that he will stay at work till 5pm with very high confidence. Another path could suggest that he will go back to his “home” at 3 pm with medium to low confidence. Each next location is predicted with a confidence percentage. There could be several such paths that could be predicted.

The paths are predicted only based on the markov chain model. If the model is not recent or is built with very limited data, the predicted paths may not make much sense. For example, the markov model is build based on the 2 days data where user transitioned between “home”, “shopping malls” and “restaurants” only. The two days do not include “work” location. It could have been a weekend, public holidays or user has taken vacation days. Now, on day third, if the paths are predicted, only these locations are taken into consideration and the paths could be misleading if user started working from day 3. Hence the data for markov chain should be enough and recent to make improved predictions.

### 5.6.1 Implementation and Result

The algorithm takes the current hour, location and minimum threshold as input and uses the markov model to predict the several paths representing the several locations in consecutive time-slots with their confidence percentages. The Figure 23 shows the paths predicted for user 1 for a known state 3 at hour 7 with minimum confidence 0.1. The x-axis denotes the hour of the day and each path starts in a new line. There exist two paths. The circle with the number denotes the states predicted and the color of each circle is to indicate the confidence of the prediction. The first predicted state at hour 8 in path 1 is 6. Similarly, the first predicted states at hour 8 in path 2 is state 3. The states with darker color indicate a high confidence and the states with lighter color indicates the lower confidence. The confidence reduces as we go further away from the starting hour i.e. 7. The path continues until the confidence drops below the threshold confidence or the day has ended. There is an additional state shown after the drop of confidence below threshold. This additional state is shown to see the drop in the confidence of the next state after the minimum confidence.

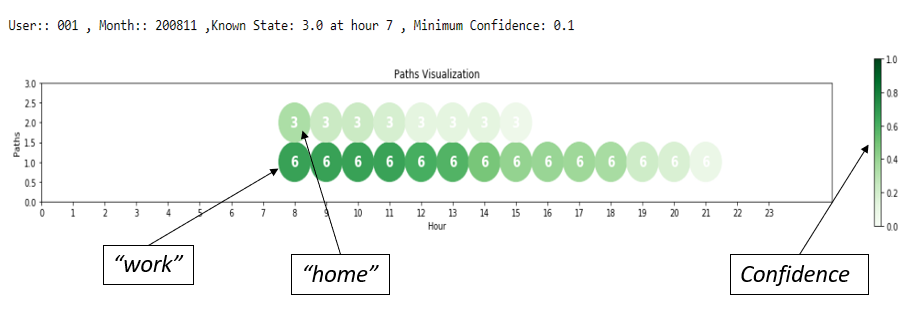


Figure 23 Path prediction for user 1

The path prediction is done to aware the user about the different paths which are predictable based on his/her location data collected. The known location was only state 3 at hour 7, which is then used to predict the paths for several hours. This is a privacy threat for the user.

### 5.6.2 Improvements

The path prediction algorithm is to suggest a user for his/her predictability. The algorithm confidence reduces as we go further away from the starting time slot. This is because of the way markov chain model works. The model keeps a record of each transition. Consider the example of user 1 data for November 2011 for first few days as depicted in Figure 24. The x-axis denotes the hour of the day and the y-axis denotes the days. During the evening the user has been visiting many new places. The transition from hour 18 and 19 as marked by in the figure is interesting to focus. The state 6 seems to be user’s “work” location. The user has visited location 5 and 14 on day 4 and 6 respectively at hour 19, after state 6 “work” location at hour 18. On other days like 2 and 5, he stays at “work” location 6 at hour 19. The markov chain will remember these transitions. This in turn reduces the probability of user to be at state 6 “work” location at hour 19.

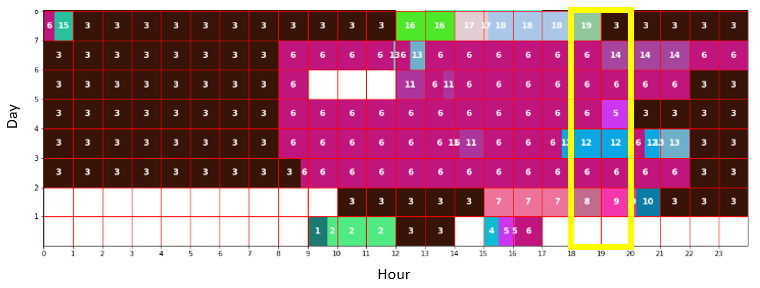


Figure 24 User 1 November 2011 hourly distributed data

As a result of this, the path predicted if the known location is at state 6 “work” at hour 14 is shown in Figure 25. Because of several possibilities at hour 19, the confidence drops instantly.

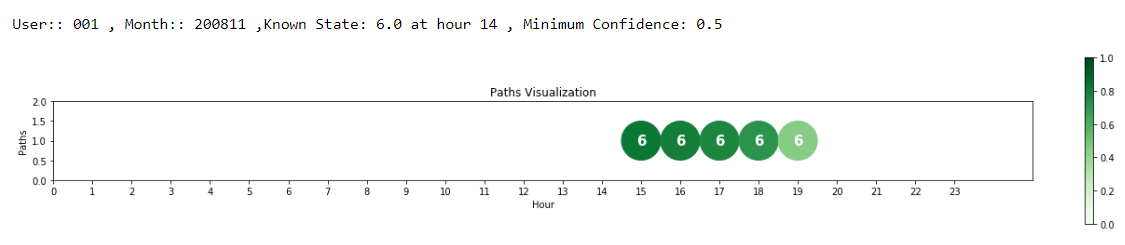


Figure 25 Path prediction user 1 with minimum confidence of 50%

But humans do not consider each detail while thinking about the transitions. It is easy to forget the places we have visited 2 weeks before after work. To improve the path prediction algorithm, it is interesting to forget some minor transitions like humans do. This can be done using the inputs from the user, hence a small survey is conducted.

The survey contained 10 participants with the median age of 26.5 years. The participants are shown the time slotted data as in Figure 16 for 1 minute of time. Before showing the time slotted data, the participants are explained that the x-axis represents the hour of the day and the y-axis represents several days the data is collected for. Each rectangle represents a location with its id inside the rectangle. They are also told to focus on user’s working hours and his movements trends. After this, all participants are asked a same question.

*“How long will the user stay at location “Work” if he was observed at location “Work” at x hour.”*

The question is to understand how the participants forget the minor transitions and expects the user to be at work for longer hours. One of the user data shown to the participants is the Figure 16. The x in the question was 8am. The average answer from participants reported that the user will stay till 8:00pm at state 6 “work” location. The path prediction from our algorithm resulted in Figure 26. The prediction is very strict which says user will stay at state 6 “work” only till 14 hours. The minimum confidence considered here is 80% and the prediction of state 6 at hour 14 is already below the minimum confidence. This clearly the way participants have observed the movements and the way markov chain has calculated the transitions are not same. Most participants ignored the infrequent and minor transitions like from state 6 to state 11 from hour 13 to 14 on day 3, and so on. But markov model recorded these transitions and this in turn reduced the probability of staying at state 6 at hour 14. Hence, markov model also need to forget these minor transitions. This is called as applying memory loss factor to the markov model.

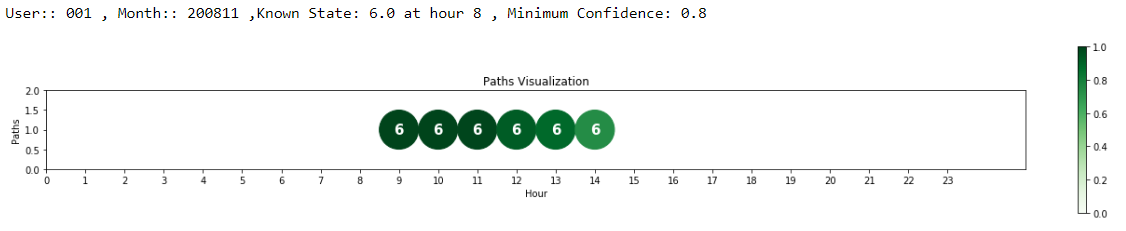


Figure 26 Path prediction result for user 1 without changes in the algorithm

The memory-loss factor is calculated for each participant. The factor is simply calculated by changing the markov model to produce the same output as the participant has suggested. For example, if the participant suggested that the user will stay at state 6 till 9pm, the same output should be produced with our path prediction. This can be implemented if the probabilities are reduced by a factor and then normalized again. This will strengthen the higher probabilities and diminish the lower probabilities completely. The process is explained with the help of a block diagram in Figure 27. It is important to remember that the markov chain model contains probabilities of transitioning from one state to another. In step 1, the relevant probability vector is extracted. In step 2, the memory-loss factor *mlf* is subtracted from each of these probabilities and if the subtraction results into a negative number then it is zeroed. In step 3, the vector is renormalized by dividing each element with the sum of the vector.

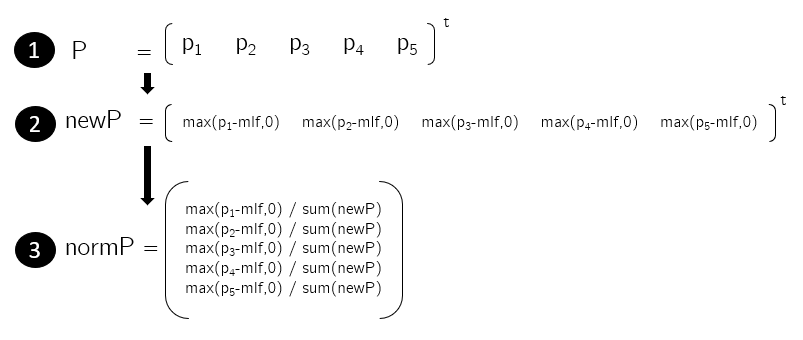


Figure 27 Steps to apply memory-loss factor to the markov chain

This process help build the stronger probabilities and lessen the weaker ones. Figure 28 depicts an example with memory-loss factor *mlf* as 0.05. When this factor is applied to the probability vector P, the stronger probabilities like 0.79 is strengthened to 0.91 and all the other probabilities are reduced. The very small transitioning probabilities are zeroed.

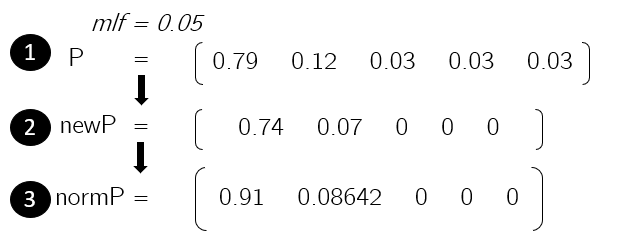


Figure 28 Example of application of memory-loss factor

After application of a memory-loss factor on user 1, we get the following result as shown in the Figure 29. The memory loss factor of 0.22 changes the suggestion to stay at state 6 till 21 hours.

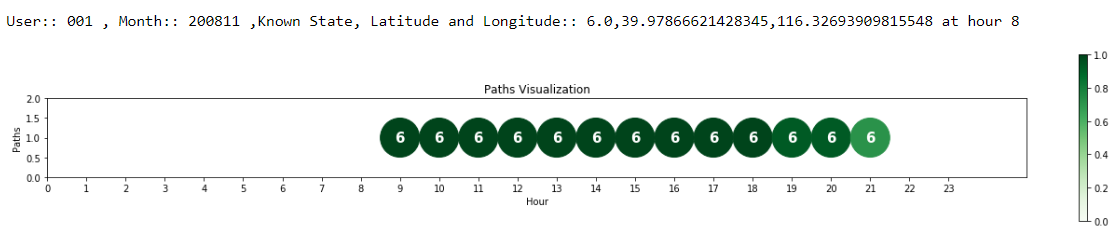


Figure 29 Path prediction result for user 1 with memory-loss factor of 0.22

Similarly, the memory-loss factor is calculated for each participant, based on their answer, to calibrate the algorithm. The average memory loss factor is reported to be 0.17.

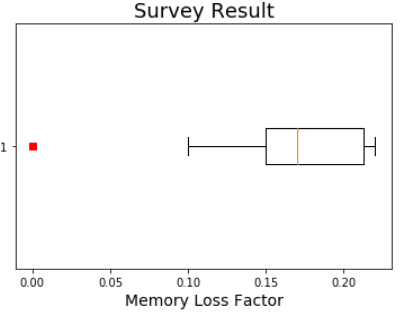


Figure 30 Memory-loss factor result from survey

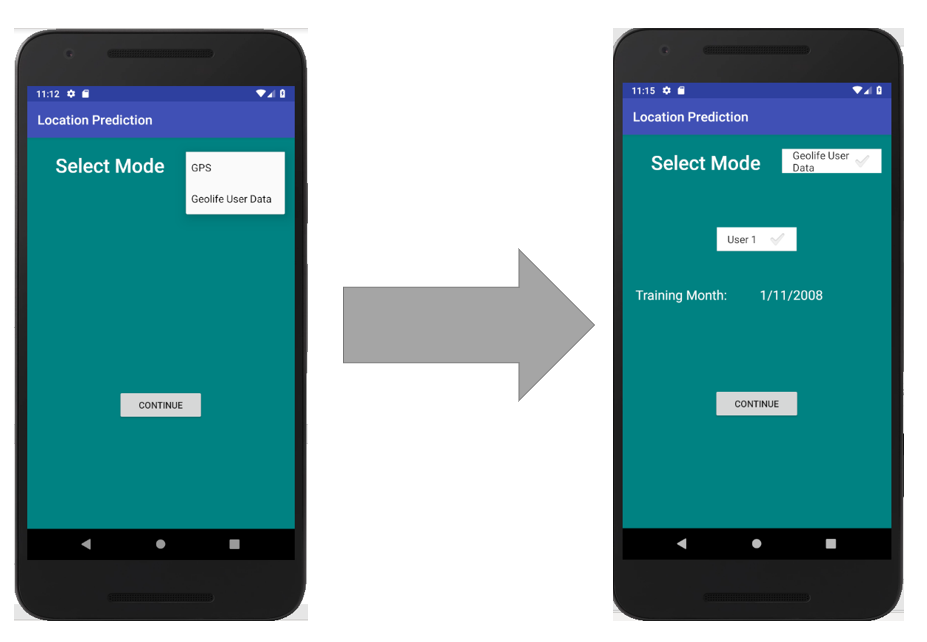
The survey was only conducted with 10 participants. The results can change with more participants and with different age groups. More research has to be done in this direction.

## 5.7 Android implementation

The implementation is done on an android phone as a prototype. This is to showcase how the mobile devices receiving location data can exploit the data and make predictions. The markov chain algorithm is the same as explained earlier. The data is again from Geolife for prototyping purpose. In the actual scenario, this data will be fed from GPS to the application.

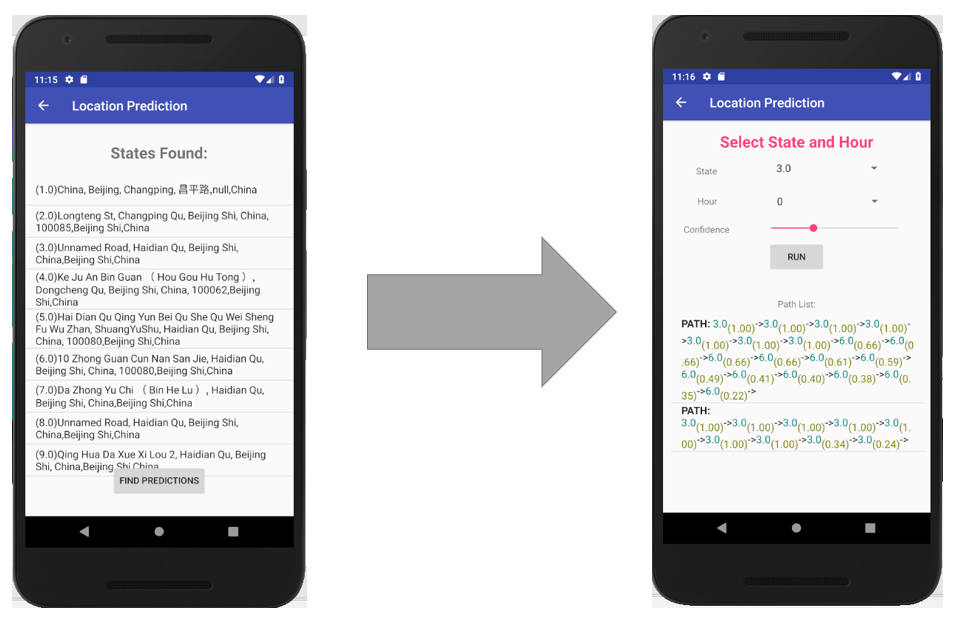
The android implementation has challenges like computational cost. The implementation is similar as the python implementation. The user can see the prediction results of sharing a location at a particular time.

The start screen of the android application has two modes “GPS” or “Geolife User Data”. Once the user has selected “Geolife User Data”, the option of choosing user and month is made visible. The user and month have to be selected from this input screen.



Once the user has made the selection and hit “Continue” button, the markov chain is build in the background from the data file. The states that are extracted which represents the significant places for the user are displayed as shown in the figure. The states may not be the exact positions but will be on the vicinity as they represent the mean of several coordinates within a range. The list is displayed to show the user all the visited placed that has been tracked down as important places.

Once the user proceed by pressing the button “Find Predictions”, user is taken to the final prediction scree. Here he/she can make a selection from the list of states and hour from the screen. The selection is to prototype if the user’s location was found to be at this selected state location at this hour, which paths could be predicted. The confidence level input can be changed using the slider. The paths are displayed below as a list with their corresponding confidence as a subscript.

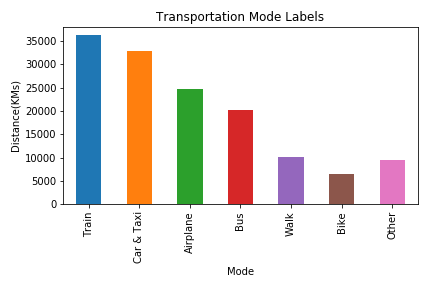


# **6 Evaluation**

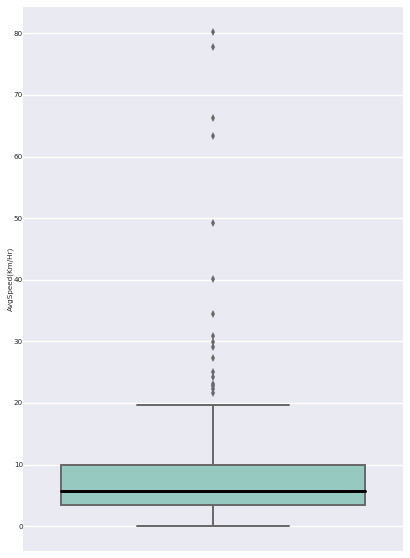
## 6.1 User data analysis

The data used for this master thesis is Geolife dataset from Microsoft. The dataset contains 182 users’ GPS trajectory data for the period of five years. The trajectory data contains the latitude, longitude, date, time, altitude information which is tracked every 1 to 5 seconds. Majority of the dataset was created in China with few exceptions of USA and Europe. This included several different types of users, few with a lot of trajectory data over years and few only for few weeks. For instance, user 17 has 1026179 trajectory points and users 72 has only 81 trajectory.

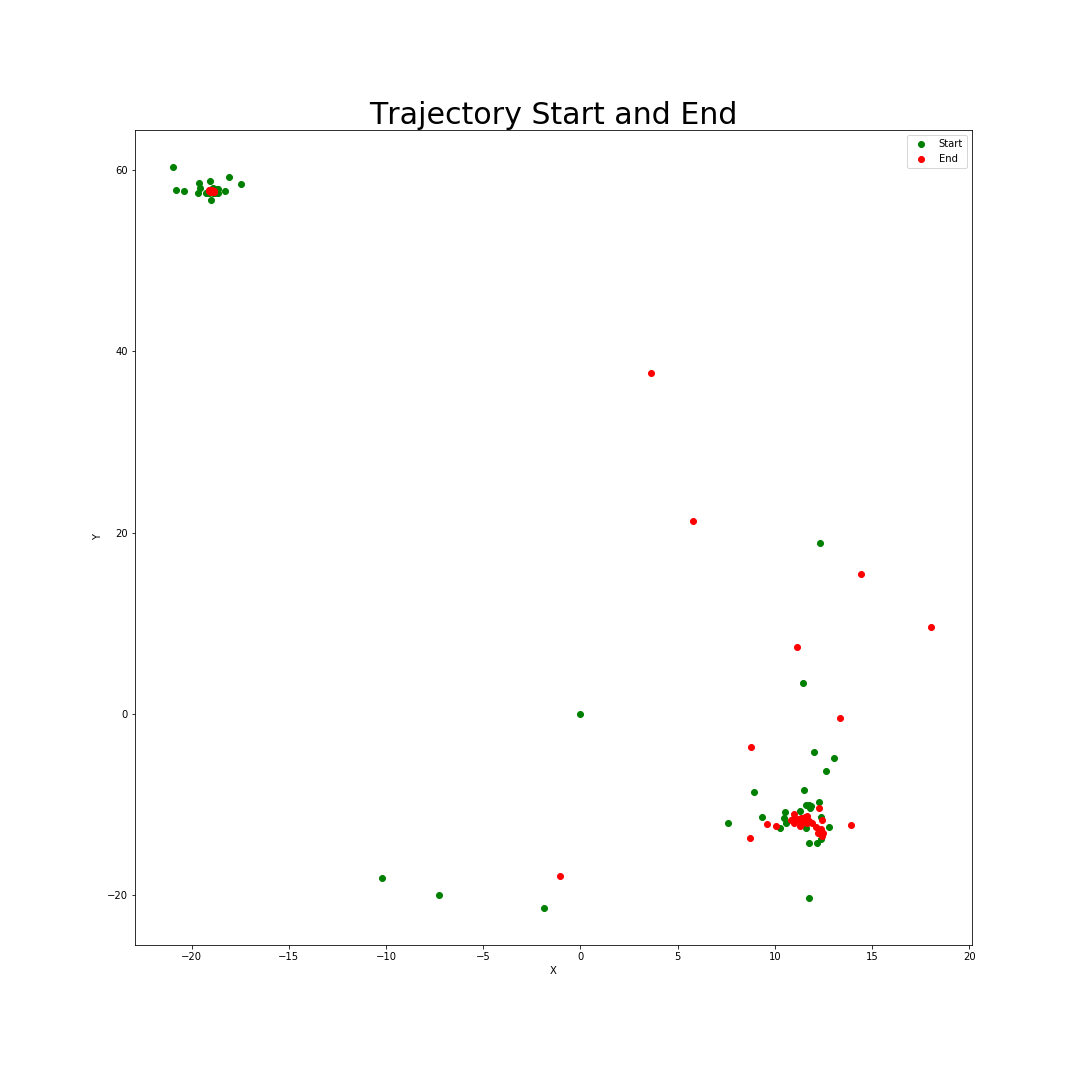
A total of 73 users have also labelled their transportation mode while recording the GPS trajectories. The figure below explains the distance travelled altogether using different transportation modes.



The transportation labels summary helped us to understand that most users were travelling while recording their GPS trajectories. The average speed which is calculated per trajectory file for each user in the dataset depicted the in figure below. The median travel speed was found to be 5.73 km/h. This indicates that most user tracked the GPS trajectories outdoor.



The analysis of each user also shows some important patterns. For instance, the first and last GPS coordinates of each trajectory is depicted in the figure. The green dots indicating the start of each trajectory and the red dots indicting the end of each trajectory. This also shows a pattern indicating that the trajectory data was mostly recorded outdoor. For example, the trajectory was always starting at home and ending at work, and then starting at work and ending at home.



## 6.3 Evaluation and Results

## 6.2 Discussion and Summary

# **7 Conclusion and Future Work**

## 7.1 Summary

In this thesis, we presented an algorithm for location prediction using markov chain model. The markov chain model is built on states. The states represent the significant locations like “home”, “work”, “gym/club”, etc. The thesis also explains each step to reach the states from raw GPS coordinates. The important steps involved are stay-points extraction, state formation, creating time-sotted data and finally markov chain model. The algorithm is built in an online manner to simulate the actual use case scenario.

The dataset used is Microsoft Geolife data which contains 182 users GPS trajectories over 5 years. The prediction model is to simulate the privacy risks for mobile device users. The algorithm is first implemented on python to check the algorithm’s accuracy. It has been found that on an average of cosine similarity \_\_ can be achieved for future location predictions. The same algorithm is also implemented on Android. The result is shown to the user in which each future location is predicted with a confidence percentage.

## 7.2 Future Work

The model is implemented only with one temporal feature which is the hour information for each location data. The prediction model can have better prediction accuracy with more spatio-temporal. For instance, building separate markov models for weekdays and weekends may increase the accuracy for the users who have very distinctive movement patterns on weekdays and weekends. Another aspect which can be tested is, weighting the transition probabilities based on geographical distance between the states. The transition from one state to another state is more likely to happen if the states are geographically close to each other. Hence, weight the transition probabilities more if the are close to each other and weight less if they are very far apart. It will be interesting to know if this increases the overall prediction accuracy.

The algorithm can also be tested in future to accept more data like calendar entries or call/SMS log. As suggested by the paper [3], it can reveal important information about user’s next movement. This information can be used to change the markov model accordingly and then make the predictions.

The idea is prototyped on the android to showcase the basic approach of the application. The location prediction model on android device uses the smoothened data due to the limited duration of time. This can be extended in future to accept the raw GPS points as input on the android device and create and update the markov model regularly. The finished android application can be published on playstore to collect user feedback.

Bibliography

|  |  |
| --- | --- |
| [1] | P. R. Center, "Americans’ complicated feelings about social media in an era of privacy concerns," 2018. |
| [2] | W. K. S. S. Paul Baumann, "The Influence of Temporal and Spatial Features on the Performance of Next-place Prediction Algorithms," *UbiComp’13,* September 8–12 2013. |
| [3] | S. S. N. L. C. M. Anastasios Noulas, "Mining User Mobility Features for Next Place Prediction in Location-based Services," *Computer Laboratory, University of Cambridge,* no. IEEE, 10-13 December 2012. |
| [4] | C. P. S. K. Joao Bartolo Gomes, "Where will you go? Mobile Data Mining for Next Place Prediction," *Institute for Infocomm Research (I2R), A\*STAR, Singapore,* August 2013. |
| [5] | N. M. P. J. H. A. K. S. B. A. T. Mitra Baratchi, "A Hierarchical Hidden Semi-Markov Model for Modeling Mobility Data," *UBICOMP '14,* pp. 13 - 17, SEPTEMBER 2014. |
| [6] | W. W. B. S. G. B. Jong Hee Kang, "Extracting Places from Traces of Locations," *Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots,* October 2004. |