MOBILITY PREDICTION USING PATTERN NETWORK

LIANG CHEN

Ph.D

The Hong Kong Polytechnic University

2016

The Hong Kong Polytechnic University

Department of Computing

Mobility Prediction Using Pattern Network

LIANG Chen

A thesis
submitted in partial fulfilment of the requirements
for the degree of
Doctor of Philosophy

July 2014

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

LIANG Chen (Name of Student)

Abstract

Nowadays, location-based services are ubiquitous in our daily life. Many mobile applications recommend nearby restaurants, transportations or new places based on user's profile. However, considering the privacy protection, customers are less willing to provide detailed information, such as age, status, habits etc.. Only moving trajectories may be accessible by telecom service providers. Mining in trajectories is one of strategies to understand customer's behaviors. Therefore, how to discover user's mobility patterns as well as making accurate mobility prediction become two critical issues for location based services. Moreover, trajectories that are updated frequently from massive people behave as streaming data, which require pattern mining and prediction algorithms to be efficient as well. In this thesis, we introduce our methods to conquer the above challenges. Our first work is to find atomic mobility patterns from trajectories. Since moving trajectory usually consists of many tandem repeats, the proposed pattern mining algorithm is able to perform repeating sub-sequence mining and tandem structure detection concurrently. With a pipeline framework, we can discover various patterns in an online manner. Next, we transform a location sequence to a novel pattern-based network by connecting all discovered patterns. The pattern network models user's historical movements from location level to pattern level, which not only provides a graph presentation for investigating user's mobility, but also serves as a mobility model for better prediction. Our pattern network model is trained by three steps including prediction, verification and weight propagation. Through online tunning the parameters of pattern network, user's next location can be predicted in real time. Finally, we focus on the mobility prediction of unusual behavior. The motivation is that many loca-

tions in our daily life are visited infrequently or only once. Usually, these locations are hard to be predicted successfully by traditional methods. We introduce the concept of mobility change, called Point of Change (POC), to describe people's new and unusual mobility behaviors. Our pattern network model is extended to include spatial-temporal information for learning and predicting possible POCs in a user's trajectory. In general, our experiments show that the pattern network model outperformed other Markov models on location prediction and unusual mobility prediction. Moreover, people's mobility behaviors can be further analyzed according to the structure of pattern network.

Publications

[1] Victor C. Liang, Vincent T. Y. Ng: Modeling of Collective Synchronous Behavior on Social Media. ICDM Ph.D. Forum 2012: 945-952, Dec. 2012, Brussels, Belgium

[2] Victor C. Liang, Vincent T. Y. Ng: A collective synchronous behavior model on social media. DUBMMSM '12 Proceedings of the 2012 workshop of CIKM on Data-driven user behavioral modeling and mining from social media 2012: 3-6, Oct. 2012, Maui, Hawaii

[3] Victor C. Liang, Vincent T. Y. Ng: Incremental Sequential Pattern Mining for Mobility. Submitted to IEEE Transactions on Systems, Man and Cybernetics

[4] Victor C. Liang, Vincent T. Y. Ng: Predicting New and Unusual Mobility Patterns. Submitted to the 14th ACM International Conference on Mobile Systems, Applications, and Services, Jun. 2016, Singapore

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Vincent T.Y. Ng, whose dedicated and devoted work helped and provided me with enough support to write this thesis. I would also like to thank my family that they were behind me through my entire life and without whom I could not have finished my studies.

Table of Contents

List of Figures

List of Tables

Chapter: 1 Introduction

Mobility is ubiquitous in people's daily life. Every day, we move from home to work, from office to restaurant, from living room to garden. In recent years, cost of GPS and cellular network positioning have become lower and lower. Various wearable devices, such as smart phones, wristbands and glasses, incorporate positioning module as their standard component. Additionally, new techniques, such as WLAN and on-boarded accelerometer, supplement GPS and cellular network for the precise of indoor positioning. With such modern techniques, people's moving trajectories can be readily recorded anywhere at anytime. Popularity of positioning techniques opens a floodgate to new researches, and launch an era of smart life and smart city.

Figure 1.1 shows a part of visited locations of a smartphone user at Seoul, Korea, which are highlighted by red dots. During a two-month tracking, thousands of locations were visited by a graduate student [9]. These locations were scattered in different areas of a city, however, most places were distributed near university. These trajectories portrait a life map of a student. We can infer his status of college student easily. Since moving trajectory usually contains spatial and temporal information, mining the mobility behaviors of users can benefit traditional marketing or

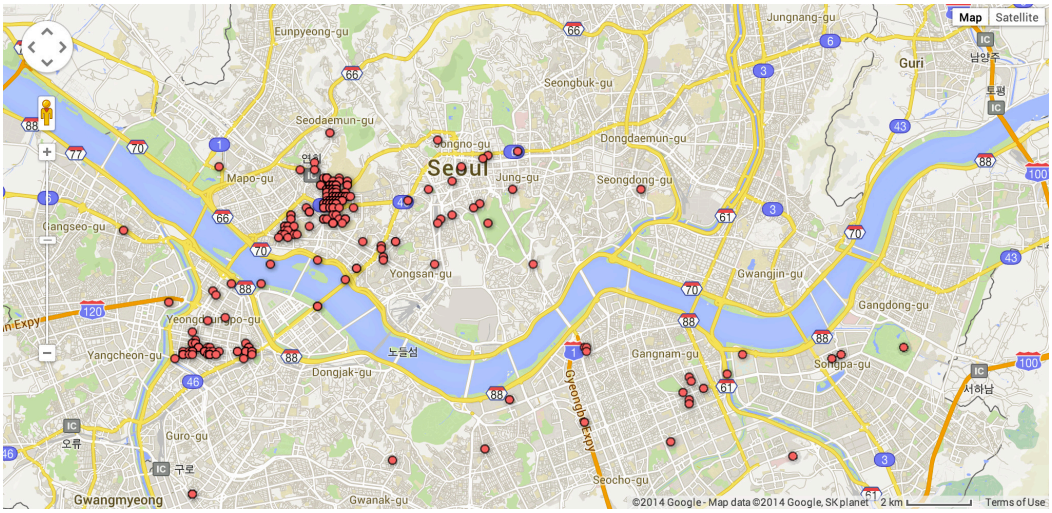


Figure 1.1: Visited Locations of a Smartphone User

urban planning greatly. In marketing, answering the questions like, what time, or where does a customer come from, and where he or she is going to, can characterize a customer from the following aspects:

- transportation mode (e.g. by bus, metro, taxi or walk);
- status (e.g. student, office worker, senior);
- hobbies (e.g. sport fans, book worm, music lover);
- life mode (e.g. walking after dinner, getting up early, traveling at weekend);
- favorite places (e.g. library, park, supermarket).

As we can see, many features of an individual user can be profiled from his or her historical trajectories. Thus, enterprises can deliver proper advertisements to their target customers after knowing customers' profiles. Similarly, in urban planning, trajectories collected from a large amount of citizens can help government to manage a city through understanding:

- traffic status (e.g. traffic jam happens at transportation hub);

- population distribution (e.g. most people live at downtown);
- emerging commercial district (e.g. many stores open at a new metro station);
- region functions (e.g. education, business, residence);
- regular events (e.g. football match every week).

Based on aggregated citizens' trajectories, government can discover problematic areas, traffic bottlenecks, group incidents etc., and make effective administrative measures.

Historical trajectories reveal regular and periodic movements of users in the past. On the other hand, people's future movements attract the attentions of enterprises and governments as well. If a user's next location can be known in advance, recommendations of nearby restaurants, shops, and transportations can be suggested to the user. The chances that a user will buy products at his or her next visiting place will be increased. Also, if government can foresee a large-scale activity that will be taken place at downtown, human resources and public transportation services can be re-allocated. Prediction of people's next movements not only contains great commercial values in business, but also is a priority on urban management, e.g. prediction of traffic flow. Therefore, analyzing, mining and especially predicting people's daily trajectories are of usefulness reaching every aspect of our life.

In this thesis, we will focus on the prediction of mobility, and present our approaches to predict individual's next location as well as unusual mobility change.

1.1 Motivations

(1) Mining Mobility Patterns

Modeling of people's mobility behavior requires to discover patterns from their trajectories. For example, user's morning and evening commutes have strong patterns including residence place, working place, transportations and regular mobility behavior on working days. However, discovering meaningful patterns and modeling mobility behavior are quite challenging. Sufficient prior knowledge and feature engineering work are necessary, which cost huge human efforts and computational resources. Moreover, mobility pattern discovery is usually based on many assumptions. Some assumptions may only fit to a part of people in a particular country. For example, most people use public transportations at Hong Kong every day. Passengers will be brought to different shopping malls along the line of metro, even they do not have the plan for shopping. Oppositely, trajectories of private car users are more straightforward. Every visited location reflects their real intentions. Due to the different culture backgrounds, it is hard to apply a knowledge-based model to understand all people's movements. Therefore, instead of putting great efforts on feature engineering, our motivation is to find out patterns from trajectory as many as possible, and build a generalized model based on these patterns.

(2) Predicting Next Location

One strong motivation that drives the research of mobility prediction is recommendation service. Nowadays, recommendation system plays an important role in many domains and works in our multi-faceted lives. For example, many online malls, online book stores, hotel booking websites, have adopted recommendation systems to infer user's purchasing preference. Based on browsing content, searching keywords or even friends' purchased records, a recommendation system will list relevant products in the welcome page and attract users to click.

With the quick development of mobile computing, recommendation services are not only limited in online marketing, but also extended to our physical world. An observation is that people usually need assistance in unfamiliar places, such as which hotel is the cheapest, where is the transportation hub. Even in a residential community, a user may still have no chance to try all local dinning places. Under such circumstances, location-based recommendation services have a great market on guiding and assisting users in their mobile lives. Binding with a physical location, a recommendation system can suggest nearby restaurants, transportations, activities, friends etc., which cater to user's preference. Therefore, prediction is a way to know which place a user is going to visit next. It is the most important prerequisite for guaranteeing the accuracy of location-based recommendation services.

(3) Predicting Unusual Mobilities

Prediction of next location is based on user's historical trajectories. Finding regular patterns and frequent visited places are main ways to infer user's possible movement. However, in many occasions, people may detour from their normal mobility behaviors, e.g.,traveling to new places or via a new route. In such circumstances, previous mobility patterns are useless, and the well-trained location-based system will be incapable to capture our real intention. Hence, an alternative strategy is to let the recommendation system interact with users and ask their destinations instead of shooting in the dark. Necessary interactions between users and the system instantly not only improve the recommendation, but also fit the real needs of users. Therefore, the ability to predict new and unusual mobility patterns is unique.

In addition to the recommendation scenario, prediction of new and unusual mobility patterns is also useful for the scheduling of mobile phone system. We studied

smartphone's battery consumption and screen activity at the locations of mobility change, and found battery consumption and screen activity increased 20% when people arrived at a new place, or changed from their past mobility patterns. Map searching, photo shooting, finding favorite restaurants nearby, checking-in social networks are the possible actions that are likely to happen. For mobile phone system, pre-loading of relevant apps and data is an effective way to ensure a smooth user experience. With the capability to predict unusual places, our smartphone can

- pre-load possible mobile apps;
- synchronize necessary files from cloud server in case of unstable network at new places;
- report possible mobility change of children to their guardians;
- reschedule regular system work on cell phone.

In view of above motivations, we believe mobility prediction is an important research work in the emergence of mobile computing. Successfully predicting next or new place that a user is going to visit has great benefits to recommendation system and mobile operation system.

In general, the research problems in this thesis are to discuss how to build an efficient mobility model from pattern mining level to model training level. In order to make accurate mobility prediction, it is not easy to build a mobility model without considering user's moving behaviors in real application. Also, high quality training datasets, meaningful patterns are essential for building and training a robust mobility model. Therefore, our motivation starts from mining useful atomic mobility patterns. Then, we can evaluate the quality of datasets by analyzing the number of patterns discovered, and build different mobility models by modeling the ap-

pearance of patterns. From the view of real applications, we need to improve the efficiency of pattern mining and mobility modeling as well. The methods proposed in this thesis consider the distributed environment so that location-based service providers are able to analyze and provide accurate predictions for millions of users in real time. Moreover, various interesting applications can be motivated by proposed pattern network model, which connects all sequential patterns into a graph model, such as next location prediction, unusual mobility behavior prediction or user classification by analyzing their pattern networks.

1.2 Contributions

This section provides an overview of the scientific contributions of this thesis to the mobility prediction.

(1) Contributions to Mobility Pattern Mining

Typically, a trajectory is represented by a discrete sequence of locations. With the growth of moving distance, more and more locations will be appended. Since the frequently updated trajectory behaves as a streaming data, predicting user's movement becomes an online task. The next location should be predicted according to all locations on and before the current time. Mining mobility patterns is necessary for prediction.

(a). In this thesis, we proposed to formulate mobility pattern in terms of repeating sub-sequences, precisely, a near-supermaximal repeat and tandem unit. The observation is that in an individual trajectory, it is unlikely that a path would be visited

twice or more only due to chance. Since a location sequence can be viewed as a string, well-developed notions in string processing can be employed in our work.

(b). We proposed an online pattern mining algorithm to discover various near-supermaximal repeats as atomic patterns. Traditional algorithms are only able to discover tandem repeats and near-supermaximal repeats separately. However, there is no algorithm which can handle the detection of both tandem and near-supermaxial repeat in an incremental manner. Facilitated by Ukkonen's suffix tree construction algorithm, a new sequence can compare with any historical subsequences incrementally. Our contribution is the development of multiple pipelines to maintain any repeats. Each pipeline is corresponding to a branch of suffix tree and caches one of suffixes of a repeat. Once the next element can not be a part of repeating sequence any more, the caching process will be stopped. Among all pipelines, the near-supermaxial repeat can be discovered by finding the longest cached pipeline. Besides, during the caching process, tandem structure of sequence will be detected as well. If a shorter pipeline contains the same content as in a longer pipeline, a tandem structure must exist in this longer pipeline. Thus, caching process will be stopped and the shorter pipeline contains a non-tandem repeat. Utilizing pipeline framework, near-supermaxial and tandem repeats can be discovered concurrently.

The overall time and space complexity of our algorithm is linear to the length of a sequence, and time complexity of each incremental step is linear to the length of the longest cache. With a multi-pipeline computational framework, each pipeline can be computed separately, which allows the parallel executing for extreme big dataset.

(2) Contributions to Next Location Prediction.

Traditionally, Markov model is regarded as the first choice for trajectory modeling. The assumption is that the next location is only dependent on limited previous locations. Usually, 1-order or 2-order Markov chain is a popular configuration for such model. However, due to the complex of people's mobility behaviors, fixed length order may not be sufficient to differentiate various cases.

(a). In this thesis, we proposed a pattern network model to predict possible next location based on discovered mobility patterns. Our intuition is that a location sequence can be composed by many nested mobility patterns along the time. We can infer a user's next location by comparing his or her recent traveling path with all historical patterns. Since the lengths of mobility patterns are different, pattern network can be viewed as a variable-order Markov model. Hence, training based on patterns with various lengths enhances our model to predict in some extreme situations.

(b). Additionally, if a pattern occurred recently, other successive patterns in history may likely to happen afterwards. Therefore, when we construct the relationship between discovered patterns, the pattern network carries all transition information of locations as well as patterns. It is a new presentation of location sequences. Through tunning the weight of patterns on the network, our model is able to learn the features of mobility trace and adapt to the recent trajectory.

(3) Contributions to Unusual Mobility Prediction

Traditional location-based service profiles user's traits by looking for patterns in historical mobility behaviors. Yet, from time to time, people are adventurous and would often like to go to unvisited places, or follow new transition paths. At that time, their next movements will be inconsistent with any previous patterns, making location-based recommendations inaccurate and irrelevant.

(a). In this thesis, we defined the next location as a Point of Change (POC) if it no longer matches the earliest on-going pattern. The POC can be regarded as a mobility change departing from a regular pattern. Furthermore, to investigate what causes a mobility change, we differentiate POC into three types including new point, new follower, and new breaker. New point is a place that has not been visited before. New follower is a visited place but the transition from previous location is new. A new breaker is a visited place arrived via a traveled path. The advantage of classifying three types of POC is that the content of recommendation can be refined in terms of respective situations.

(b). To predict POCs, we further extended the pattern network to the spatial-temporal mobility model, called ST-Pattern Network. With the consideration of both spatial and temporal aspects, the likelihood of POCs can be learnt according to the regularity of patterns. Our assumption is that under a strong regularity pattern, the chance of POC occurrence will be small, and vice versa. Therefore, we utilized different mobility patterns as many as possible from historical location sequence. By computing the variations of patterns, our model can predict future POCs as well as adapt to new trajectory via a pattern network.

(c). Furthermore, we investigated the relationship between unusual mobility behaviors and smartphone usage behaviors. We found the average battery consumption

of smartphone increases 21% on POC locations. Similarly, the times of screen activated and duration increase 19% and 20% respectively. These results proved that people utilize their smartphones more intensively under unusual situations. Hence, the ability to predict POC is useful for the optimization of smartphone utilization.

1.3 Outline of Thesis

In this thesis, a systematical study of various techniques on mobility prediction will be presented. In Chapter 2, we first introduce the background of trajectory analysis, mobility pattern mining, location prediction. Furthermore, we review some recent literatures on location-based recommendation systems. In Chapter 3, we demonstrate our novel online approach for discovering non-tandem near-supermaximal repeats. A framework of pipelines for maintaining and discovering patterns will be studied in details. In Chapter 4, discovered mobility patterns will be utilized for predicting next location. Construction of pattern network and the training of pattern network model will be demonstrated also. In Chapter 5, we include spatial and temporal information of mobility patterns to enhance the pattern network model for unusual mobility prediction. Definitions of Point of Change (POC) will be described in detail, and algorithms of POC learning will be presented afterwards. Lastly, we conclude our work and further work in Chapter 6.

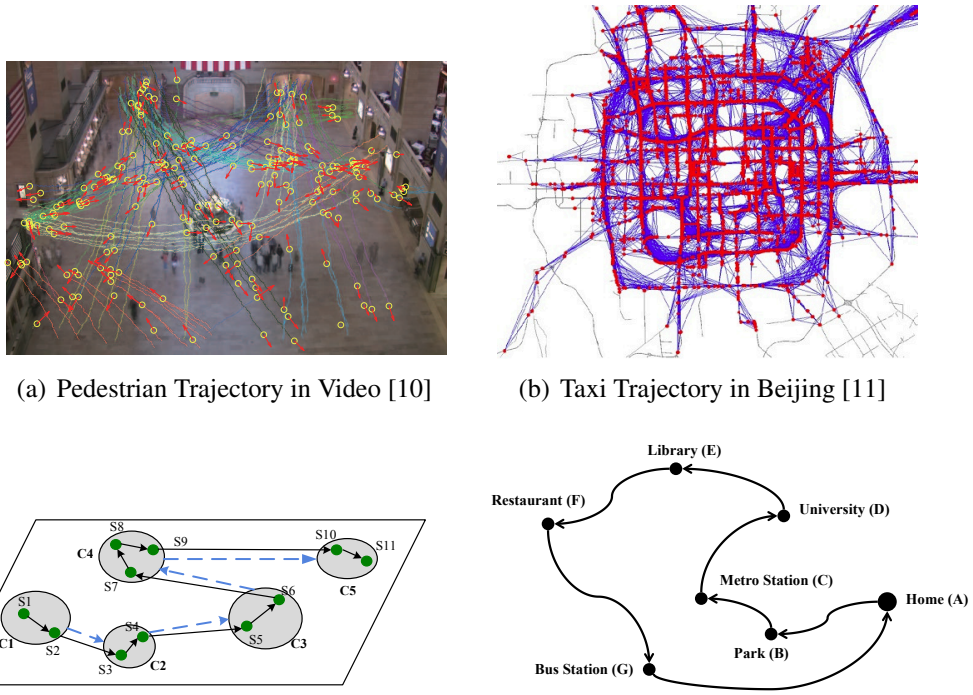
Chapter: 2 Background

In this chapter, we will give an overview of the related works on trajectory analysis, location prediction, unusual mobility prediction and location-based recommendation system. Recent advances in data mining enhanced the capability of pattern recognition and knowledge discovery from huge amounts of trajectories. These knowledges enable us to have a deeper understanding of people's preference and behaviors from the view of mobility.

2.1 Trajectory Presentation

What is a moving trajectory? In different research areas and applications, the definition and data presentation of trajectory could be different. According to their underlying formats, we categorize a moving trajectory into four types, *pixel-based sequence, real-world coordinate-based sequence, location-based sequence, and semantic-based sequence*. Figure 2.1 shows four examples of moving trajectories in different domains.

(1) Pixel-based Sequence



(a) Pedestrian Trajectory in Video [10]

(b) Taxi Trajectory in Beijing [11]

(c) Location Sequence [4]

(d) Semantic Sequence

Figure 2.1: Categories of Trajectory

Figure 2.1(a) draws a bunch of lines to represent pedestrians' motions at central station [10]. The trajectory of a pedestrian's motion in a video sequence is formalized by a set of 2 dimensional pixels (p_x, p_y). By extracting these pixel-based sequences, researchers aim to model motions, interactions or abnormal behaviors of moving objects automatically for surveillance purpose [12], [13], [10]. The main property of pixel-based sequences is that trajectory only covers a specific scene, such as street, station or shopping mall, during a short time period.

(2) Real-world Coordinate-based Sequence

Figure 2.1(b) demonstrates movements of thousands of taxis in Beijing [11]. A taxi's position at each time point is recorded by a real-world coordinate pair via GPS, i.e. $(longitude, latitude)$. In terms of sampling frequency, coordinates can be

acquired secondly, minutely or hourly. The sequence of coordinate pairs is regarded as a trajectory. The coordinate-based sequences can be generated when tracking people, animals or vehicles. By partitioning or clustering these trajectories, many researches have mined people's transportation modes or periods of animal migration successfully.

(3) Location-based Sequence

Figure 2.1(c) plots a series of connected discrete points [4]. Each green point S is a coordinate and each shaded circle C is a location. Different from pixel-based and coordinate-based sequence, location sequence is a collection of abstracted stay points sorted in time order, which is usually represented by

$$S_{1:n} =< \eta_1, t_1 >, \ldots, < \eta_n, t_n >, \tag{2.1}$$

where η_n represents the name or ID of a physical location, and t_n is the arrival timestamp of location η_n. The stay point can be extracted by various methods, such as clustering nearby coordinates, filtering through stay duration or velocity etc. [4], [9]. Location sequence is a high level transformation of pixel-based or coordinate-based sequence, which is one of the most common presentations. In many applications, rather than dealing with exact coordinates, location-based sequence can be processed by many string processing algorithms for pattern mining and prediction.

(4) Semantic-based Sequence

Figure 2.1(d) plots a user's visited locations in day time, and each location is annotated by a semantic label. Instead of using location ID, a trajectory can also be

presented by a sequence of semantic labels sorted in time order. For example, a transition between two locations "*A*" and "*B*" in Figure 2.1(d) can be transformed as a move from "*Home*" to "*Park*" by incorporating geographic knowledge. The semantic sequence is easy to be understood and interpreted. Especially, we can compare two person's trajectories according to their semantic labels, although they are not in the same city.

According to the definitions above, we can find real-world coordinate-based sequence, location-based sequence and semantic-based sequence can be converted each other. By grouping nearby coordinates, we can obtain a location sequence. And, a location sequence can be further transformed to a semantic-based sequence by labeling the meaning of each place. In our work, we focus on location sequence, because our target applications are location-based services. The recommendation is based on a place a user visited. The location sequence is very similar to a character stream, because each element in the discrete sequence is a location ID. Therefore, it is very convenient for us to process location sequence by existing string processing methods, such as suffix tree and suffix array.

In this thesis, our datasets are collected from two sources, *location-based social networks and smartphone records*. Location sequence is the major data format we need to handle for pattern discovery and location prediction.

2.2 Sequential Pattern Mining

As discussed above, both location-based and semantic-based sequences are popular data presentations in mobile applications. One advantage of using location sequence is that many traditional sequential pattern mining algorithms can be ap-

plied to discover interesting patterns directly. Kumar et al. compared and classified lots of well-known approaches in their work [14]. Here, we list some of them as follows:

- Apriori Based Method (e.g. GSP: Generalized Sequential Patterns);
- Vertical Format Based Method (e.g. SPADE: Sequential PAttern Discovery using Equivalent Class);
- Pattern Growth Based Method (e.g. FreeSpan, PrefixSpan);
- Constraint Based Method (e.g. SPIRIT: Sequential pattern mining with regular expression constraints);
- Closed Sequential Pattern Mining (e.g. CloSpan);
- Sequential Pattern Mining in Data Streams (e.g. SS-BE, SS-MB);
- Mining Incremental Patterns (e.g. IncSpan: Incremental Mining of Sequential Patterns);
- Multidimensional Sequential Pattern Mining (e.g. UNISEQ);
- Mining Closed Repetitive Gapped sub-sequences.

The above methods consider a general problem to find the complete set of frequent items (sub-sequences) when given a set of sequences. Typical applications of sequential pattern mining can be found in the analysis of supermarket transactions, web click stream, DNA sequence etc.. Next, we briefly introduce two representative algorithms, *GSP:Generalized Sequential Patterns and PrefixSpan*.

(1) GSP: Generalized Sequential Patterns

GSP (Generalized Sequential Patterns) is an Apriori-based method proposed by Proposed by R. Srikant and R. Agrawal in their early works [15]. In a sequence

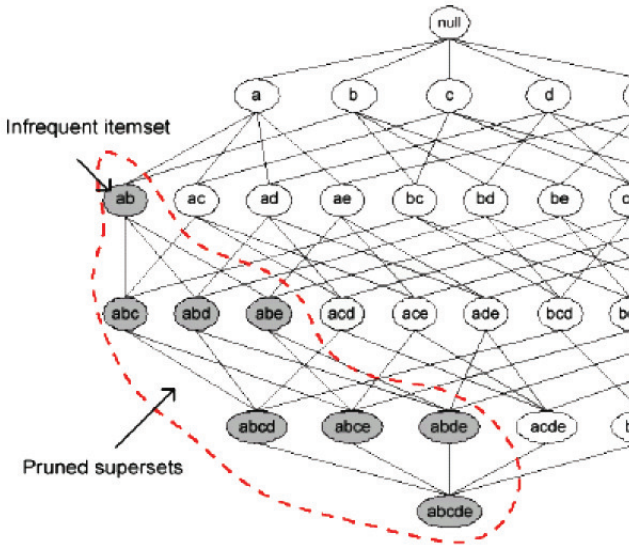


Figure 2.2: Candidate Generation using GSP

database, GSP mines frequent patterns in a level-wise fashion. The principle is that if a sequence S is not frequent, none of the super-sequences of S can be frequent. A super-sequence is the one that contains S. And, the frequency is quantified by a support count, which is the number of times S appears in different records. For example, if a sequence $< a, b >$ does not appear more than once, all its super-sequence like $< a, b, c >$, $< a, b, d >$ is not possible to occur twice.

After specifying a minimum support threshold, GSP starts to find all length-1 sequences in the database. In the next level, the algorithm uses length-1 candidates as seeds to search longer candidate sequences with corresponding support counts. This process is repeated until no frequent sequence can be found. Figure 2.2 shows

a diagram of candidate generation in different levels. Candidates in dark circles are infrequent sequences. Using Apriori, these candidates are pruned since they cannot pass support threshold, which save the cost for database search.

However, the drawbacks are (1) a large set of candidates will be generated at each level, and (2) database need to be scanned many times. Therefore, Apriori-based algorithm is inefficient for mining long sequences.

(2) PrefixSpan

To avoid huge number of generated candidates in Apriori-based algorithm, a pattern growth based method, PrefixSpan [1], is designed for efficient search. Instead of considering a super-sequence, pattern growth method utilizes the fact that any frequent sub-sequence can always be found by growing a frequent prefix. Therefore, PrefixSpan works by examining only the frequent prefixes of sub-sequences and projecting their corresponding postfixes into projected datasets. A sequence database is projected into a set of smaller databases recursively, and patterns can be mined from each projected database.

Figure 2.3 shows an example of PrefixSpan algorithm. At the beginning, it finds all length-1 sequences like $< a >, < b >, \ldots, < f >$ as prefixes, and groups the corresponding postfixes into 6 subsets. To further narrow down the search space, for example, all sub-sequences in the $< a >$-projected database will be partitioned into the following 6 subsets having prefixes $< aa >, < ab >, \ldots, < af >$. The partition will continue recursively until no more sub-sequence satisfying the minimum support threshold. The advantages of PrefixSpan are (1) no candidate needs to be

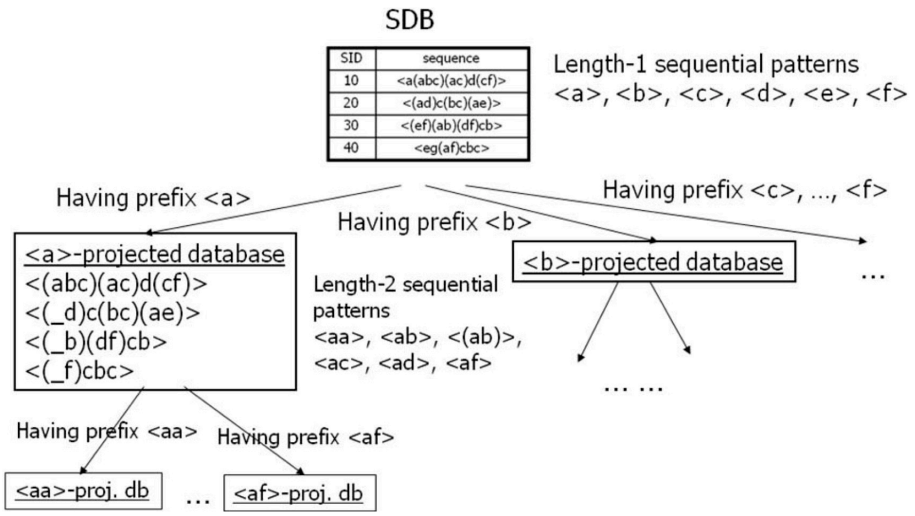


Figure 2.3: Example of PrefixSpan [1]

generated, and (2) the projected databases to be searched keep shrinking. The PrefixSpan is much suitable to mine large sequence database.

The data handled by the algorithms above is a list of transactions ordered in time. Each transaction contains a set of items. Unfortunately, a trajectory is usually a long location sequence. Arbitrary sampling of a sequence in terms of hour, day or week may lead to information loss. Therefore, string processing algorithms are better to discover interesting patterns from a location sequence directly, for example, finding the maximal repeats. Here, we review two main data structures that support various string processing algorithms, *suffix tree and suffix array*.

(3) Suffix Tree

Suffix tree is a mature data structure that allows many string matching problems to be solved quickly, e.g. finding the longest common substring and maximal repeat [16]. For a sequence of length n, it can be constructed in $O(nlogn)$ time and requires

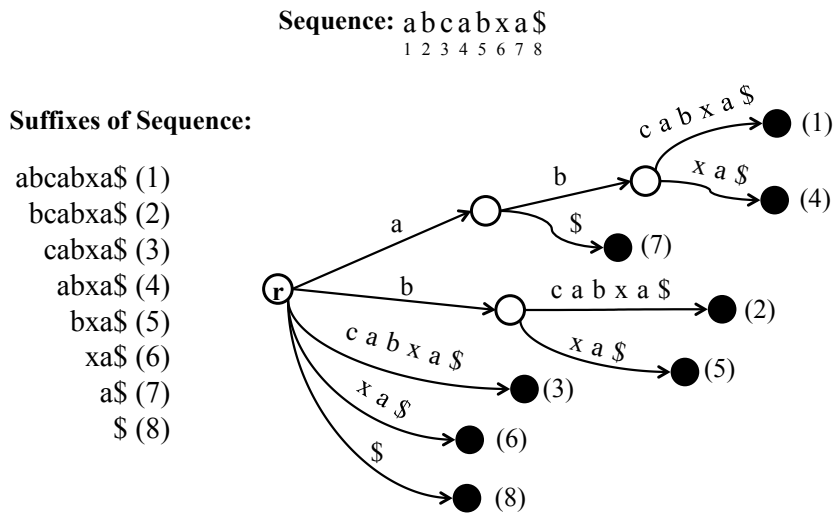


Figure 2.4: Structure of Suffix Tree

only $O(n)$ space by Ukkonen's online algorithm [17]. With a compact tree structure, all suffixes of a sequence can be stored completely.

Figure 2.4 demonstrates a suffix tree that stores a string $S =$ "*abcabxa*\$", where \$ is a terminal symbol. All 8 suffixes of sequence S are stored along the paths starting from the root to a leaf node. The start position of every suffix in a sequence is labeled at each corresponding leaf node. Due to the compact structure, an edge in-between two internal nodes (or root and internal node) may share a common prefix of multiple suffixes. For the suffix tree of a n-length sequence, there are total n leaf nodes. All internal nodes, except the root, have at least two children, and the suffix tree can have at most n internal nodes. With the tree structure, string matching can be performed by comparing characters from the root to a leaf.

To find out all maximal repeats in a sequence, D. Gusfield [18] proposed an algorithm to search maximal repeats on the nodes of suffix tree. He defined the *left character* which is the character left to the character at position i in a string S. Also,

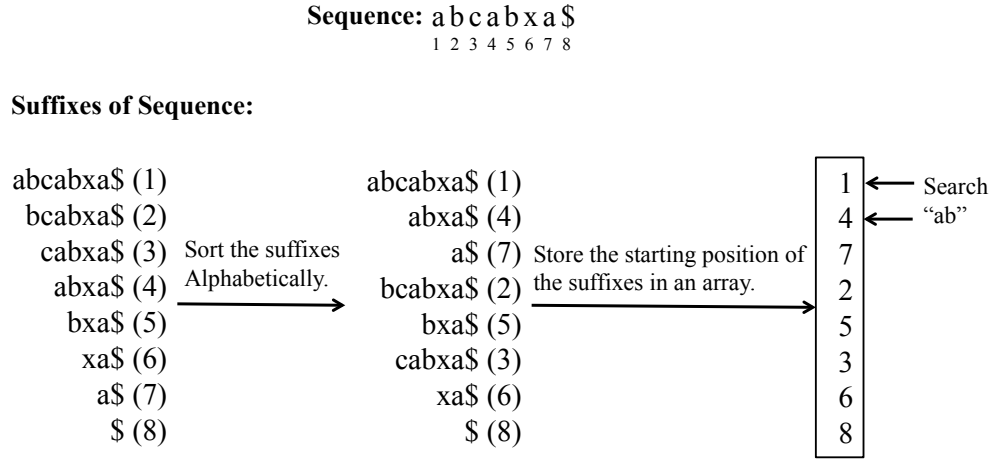


Figure 2.5: An Example of Suffix Array

a node v of suffix tree is called *left diverse* if at least two leaves in $v's$ subtree have different left characters. The algorithm goes through every node v from the bottom to the top, and marks all nodes with left diverse by distinguishing different left characters of node v. D. Gusfield proved that finding the left diverse nodes is sufficient to find all maximal repeats.

(4) Suffix Array

In addition to the tree structure, storage of suffixes can also be implemented in an array. Suffix array [19] is a lexicographically sorted array. Each suffix in this array is represented by its starting position in the sequence S.

Figure 2.5 shows the same string example of Figure 2.4, but in the form of suffix array. Firstly, all suffixes are rearranged according to their alphabetic characters from left to right. Secondly, the numbers of suffixes' start positions are filled into the array with the same order. For a substring search, if the substring occurs in S, then all its occurrences are consecutive in the suffix array. A binary search can

be used to locate the substring in the array quickly, and string counting can be completed by counting the consecutive entries.

Comparing with Figure 2.4, the ordering of start positions in suffix array is the same as the ordering of leaf nodes in suffix tree (leaf nodes from top to bottom). This observation implies suffix array can be deduced from suffix trees. Any of the linear time suffix tree construction algorithms can not only be used for suffix arrays, but achieve the economy of space by only storing an array. Therefore, many string processing problems can be solved on suffix array as well [20].

In this thesis, we mainly use suffix tree to perform pattern discovery from location sequence.

2.3 Pattern Discovery in Trajectory

What is the pattern of people's mobility? How can we mine such patterns from a moving trajectory? Apart from sequential patterns, many researches have defined different types of spatial-temporal patterns to investigate regularities in trajectory. Here, we give some typical approaches.

(1) Periodic Pattern

A periodic pattern can be considered as the repeating activities at certain locations with regular time intervals. It is a non-trivial movement characteristics that can provide an insight to understand people's historical movement. Li et al. answered two questions in their early works [2], *how to detect the periods in complex movement, and how to mine periodic movement behaviors*. First, they identified that multiple

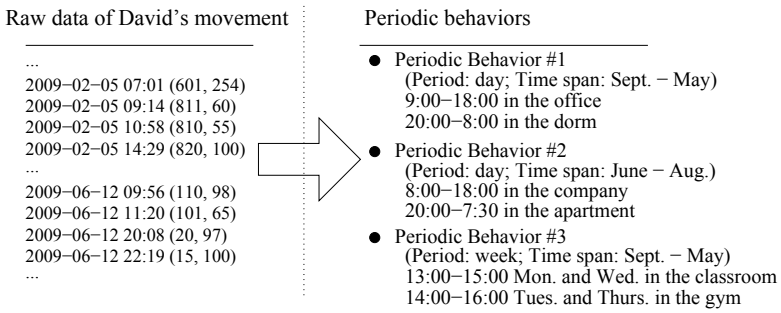


Figure 2.6: An Example of Periodic Behaviors [2]

periods like week or day usually co-exist, and it is hard to set a unified threshold to determine hourly, daily, and yearly period. Many traditional techniques, such as Fourier transform and autocorrelation, are failed to catch the periods in spatial-temporal data. Due to the noisy and variation of people's movements, an activity may not happen exactly at the same location and at the same time point. Secondly, they showed that people can have different periodic behaviors even the time period is the same. An example shown in Figure 2.6 was used to justify their claim about this possible case. In the figure, it shows that David's daily behaviors were different from September to May and from June to August.

To solve the above problems, they assumed that multiple interleaved periodic behaviors usually occur when associated with certain reference locations. The reference location is defined as a dense area that is frequently visited in the movement, called a *reference spot*. A two-stage algorithm, Periodica, was proposed to mine these periodic movements. At the first stage, all possible periods were detected directly from the raw data. They partitioned a space into different grids and used a kernel method to estimate density regions as reference spots. Then, for each reference spot, they transformed a spatial sequence to a binary sequence. A binary value of 1 will be assigned to i-th bit if an object visited this reference spot at time i.

Finally, they combined Fourier transform and autocorrelation to obtain all periods from each binary sequence.

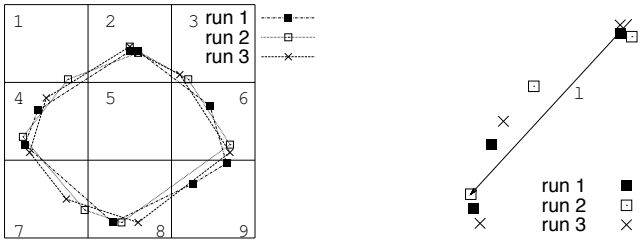
Since within the same period, different behaviors may exist, at the second stage, they needed to find out all periodic behaviors for all periods. A hierarchical clustering method was used to cluster different behaviors in the same period, and a generative model was applied to measure the distance between two periodic behaviors. After clustering, each cluster represents a type of periodic behavior. For example, in Figure 2.6, "School days" is one behavior and "Summer days" is another one.

In their later works [21], they extended the problem of mining event periodicity from incomplete observations by a probabilistic measurement.

(2) Spatial-Temporal Patterns

Besides looking for the periodicity, patterns in trajectory can also be frequent in spatial and temporal aspects. Cao et al. [3] observed that locations are not always repeated exactly in every instance of a movement pattern. A better way is to enlarge the spatial area of comparison, and transform a location sequence to a series of large regions. Figure 2.7 demonstrates a circular movement pattern, but is composed by different locations and regions.

However, the definition of region could be arbitrary and loss information of underlying movements. To conquer such problem, they used a line segmentation technique to transform a spatial-temporal sequence into a list of directed line segments. Similar lines were merged to a mean line segment by minimizing their errors. After that, the region could be formed by finding two vertical and two horizontal mean lines



(a) (b)

Figure 2.7: An Example of Object Movement [3]

around it as shown in Figure 2.7(b). Finally, through a substring tree in a heuristic way, they mined the frequent spatial-temporal patterns as a series of frequent regions that satisfy a minimum support threshold.

Their work solved the spatial trajectory mining problem in a coordinate level, which compensated some limitations of sequential pattern mining. However, their approach did not consider temporal information or more geo-knowledges.

(3) Life Patterns

On the other hand, Ye et al. [4] considered the trajectory mining problem in a higher knowledge level. They defined and summarized a life pattern from different properties:

- Temporal Granularity and Condition (e.g. an hour, a day, a week, on Monday, on work days);
- Significant Places (e.g. School, Company, Hospital);
- Sequentiality (e.g. Transitions: *"Home"* $\rightarrow$ *"Bus Station"* $\rightarrow$ *"University"*);
- Timestamp-Annotation and Timespan-Annotation (e.g. arrival, departure time, stay duration);

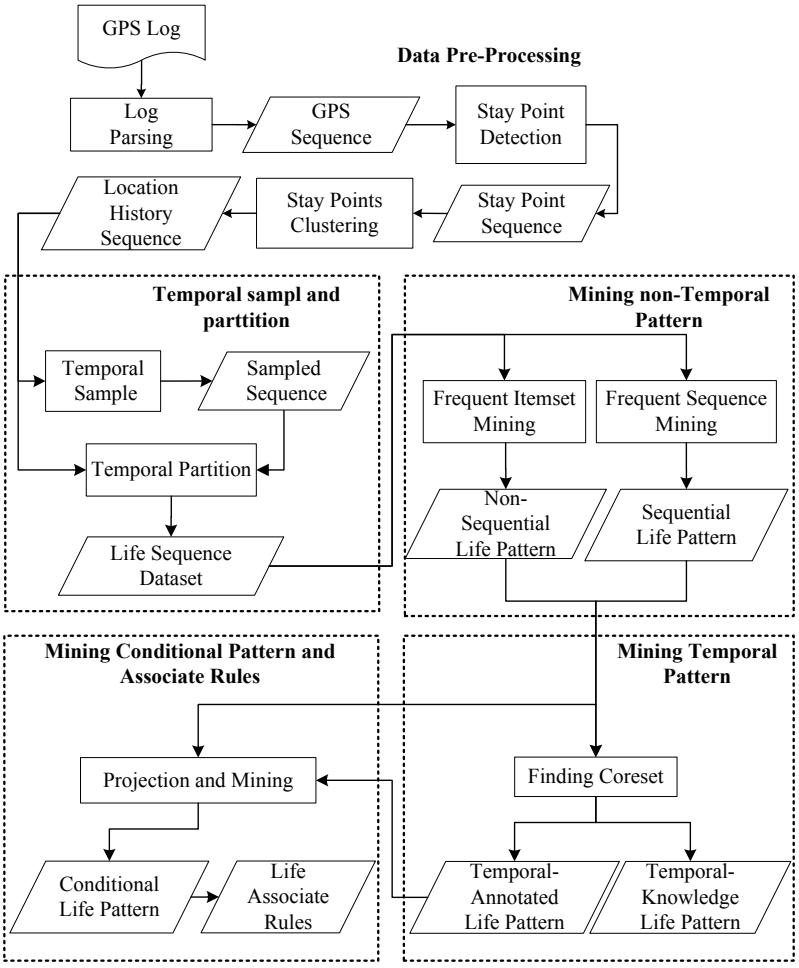


Figure 2.8: Architechture of LP-Mine [4]

- Conditional Life Pattern (e.g. "If Tom goes to school, 80% times he arrives at school before 9AM.");
- Life Associate Rule (e.g. "When Tom goes to school, 60% time he will visit convenient store.").

In their definitions, a significant place is a place that can reflect his or her major activities, and can be regarded as an atomic pattern. A normal form of *life pattern* (LP-normal form) was defined by combining different atomic patterns to complicated ones using above properties.

Figure 2.8 illustrates an architecture of life pattern mining framework called LP-Mine. The framework contains two stages, data pre-processing and life pattern mining. The pre-processing stage mainly serves for the transformation of a raw GPS log to a historical location sequence. To mine life patterns in location sequences, four modules are necessary to ensure finding out above six properties:

(A) Temporal sampling and partitioning: This module divides a location sequence into many sub-sequences according to the temporal granularity like “day”, “week”, etc. And, these sub-sequences are stored into a life sequence dataset.

(B) Mining non-temporal life patterns: This module focuses on spatial information only. On one hand, the module treats each sub-sequence as a set of independent significant places. Transition between two places were ignored. A closet+ algorithm is applied to retrieve frequent closed set of places across all sub-sequences. On the other hand, considering the ordering property of a sub-sequence, the CloSpan algorithm is used to mine sequential life patterns in the sequence database.

(C) Mining temporal-life patterns: This module further retrieves the corresponding arrival, departure timestamps and stay durations of each non-temporal life pattern. Then, temporal-annotated and temporal-knowledge life patterns can be mined using a geometric approximation through corsets.

(D) Mining conditional life patterns and life associate rules: This module retrieves all sub-sequences that contains a particular non-conditional life pattern initially. Then, it uses a projection-and-mining step to project conditional life patterns by deleting elements corresponding to the non-conditional life pattern from the retrieved sub-sequences.

Life pattern involves in multi-faceted patterns in our daily life, which is a domain specific definition. By discovering various life patterns, recommendation systems can suggest contents not only according to single spatial-temporal pattern, but also from pattern combinations.

2.4 Location Prediction

Location prediction is an important task in many applications including cellular networks [22], ad hoc wireless networks [23] [24], transportation recommender systems [25], smartphone energy optimization [26], etc. With the emergence of mobile computing, extensive researches have been made on human mobility prediction [5] [27].

In general, location predictors can be classified into two categories, *domain-independent predictor* and *domain-dependent predictor*. For a domain-independent predictor, prediction of next location is only based on spatial and temporal information. Usually, these predictors can be performed in an online manner, which summarize historical trajectory, extract current context, and predict next location incrementally. On the other hand, domain-dependent predictors may consider more information beyond spatial-temporal trajectories. For example, check-in on location-based social network, or friend's visiting records of social network can be integrated into a location predictor as well.

There are two popular domain-independent predictors, *order-k Markov scheme* and *Lempel-Ziv scheme*. Song, et al. [5] evaluated the performance of Markov model family and LZ family in their work respectively. Here, we briefly introduce two approaches.

(1) Order-k Markov

Markov model is a well-known predictor that has been applied in many applications [28][29]. In a location sequence $S_{1:N} = \eta_1, \ldots, \eta_N$, the order-k (or "O(k)") Markov predictor predicts the next location η_{N+1} based on the information from k most recent contexts of locations $\eta_{N-k+1}, \ldots, \eta_{N-1}, \eta_N$ in history. It models each context η (e.g. location ID, name) as a state, and a location as a random variable X. Let $\mathbb{A}$ be the set of all historical location contexts, $\eta_{N+1} \in \mathbb{A}$, and $n = 1, \ldots, N$. The order-k Markov model can be presented as follows,

$$\begin{aligned} & P(X_{N+1} = \eta_{N+1} | X_N = \eta_N, X_{N-1} = \eta_{N-1}, \ldots, X_1 = \eta_1) \\ & = P(X_{N+1} = \eta_{N+1} | X_N = \eta_N, X_{N-1} = \eta_{N-1}, \ldots, X_{N-k+1} = \eta_{N-k+1}) \\ & = P(X_{n+1} = \eta_{n+1} | X_n = \eta_n, X_{n-1} = \eta_{n-1}, \ldots, X_{n-k+1} = \eta_{n-k+1}). \end{aligned} \tag{2.2}$$

In Eq. 2.2, for the location in time n, if the contexts of k recent locations are the same, the probability of next location will be the same. That shows the stationary distribution of probabilities from history to the current. In a Markov model, the probabilities of location transitions can be represented by a transition probability matrix for $O(1)$ Markov model, or a transition cube for a $O(2)$ Markov model. For example, in the transition matrix of an $O(1)$ Markov model, each row or column is a location context belonging to $\mathbb{A}$. The probability from location η_n to η_{n+1} can be represented by $P(X_{n+1} = \eta_{n+1} | X_n = \eta_n)$.

To build the transition matrix, we need to estimate each probability of location transition from historical location sequence $S_{1:N}$. Let $s_{n-k+1:n} = \eta_{n-k+1}, \eta_{n-1}, \ldots, \eta_n$ be a k-length sub-sequence. The probability estimation of an $O(k)$ Markov model

can be calculated as follows:

$$\begin{aligned}&\hat{P}(X_{n+1} = \eta_{n+1} | X_n = \eta_n, X_{n-1} = \eta_{n-1}, \ldots, X_{n-k+1} = \eta_{n-k+1}) \\ &= \frac{N(s_{n-k+1:n}, \eta_{n+1} | S_{1:N})}{N(s_{n-k+1:n} | S_{1:N})},\end{aligned} \tag{2.3}$$

where $N(s_{n-k+1:n}, \eta_{n+1} | S_{1:N})$ denotes the number of times the sub-sequence $\eta_{n-k+1}, \eta_{n-1}, \ldots, \eta_{n+1}$ occurred in the location sequence $S_{1:N}$.

Given a location sub-sequence ending at time N, $s'_{N-k+1:N} = \eta_{N-k+1}, \eta_{N-1}, \ldots, \eta_N$, we can predict the context of next location η_{N+1} by finding the maximum probability of location transition from the following equation:

$$\eta_{N+1} = \arg\max(\ \hat{P}(X_{N+1} = \eta | X_{N-k+1:N} = s'_{N-k+1:N}\ ),\ \ \eta \in \mathbb{A}', \tag{2.4}$$

where $\mathbb{A}'$ is a small set of locations that followed the contexts of sequence $s'_{N-k+1:N}$ before, and $X_{N-k+1:N}$ is a series of random variables from time $N-k+1$ to N. If $\mathbb{A}'$ is empty, the Markov model makes no prediction.

According to the properties mentioned above, a Markov model has the following advantages:

- Implementation of model is easy. Only a transition matrix is needed to be maintained;
- The model is straightforward and easy to be understood;
- Only one transition probability is required to be updated after moving to the next location;
- An online prediction is supported.

However, the weakness of $O(k)$ Markov model is that the order-k has to be determined before training. Usually, the best k is hard to be selected in different applications.

(2) Lempel-Ziv Predictor

Lempel-Ziv predictor is derived from an incremental parsing algorithm, which is originally used for text compression. Let γ be an empty sequence. For a long sequence S, the LZ algorithm partitions the sequence into multiple distinct sub-sequences $s_0, s_1, \ldots, s_m$, and makes $\gamma = s_0$. In the sequence S, suppose there are two sub-sequences s_i and s_j, where $0 \leq i < j$. The prefix of s_j without its last character is equal to s_i. According to this property, partitioning of a sequence is done sequentially from beginning to end. Later sub-sequence s_j is extended from earlier sub-sequence s_i by one character each time. For example, a string $gbdcbgcefbdbde$ can be incrementally parsed as multiple sub-sequence $\gamma, g, b, d, c, bg, ce, f, bd, bde$.

As discussed above, every later sub-sequence is necessary to be compared with all earlier sub-sequences to determine its distinctness. To facilitate the string comparison, LZ algorithm constructs a LZ tree dynamically associating with the incremental parsing process. Figure 2.9 shows the structure of LZ tree. A sub-sequence s_i is inserted into the tree from the root node γ to a leaf node. Each node of the tree is labeled with a character of a particular sub-sequence. If two sub-sequences have the same prefix, common nodes will be shared from the root. To indicate the number of occurrences of each sub-sequence, a counter value is stored in each node as well. Initially, a pointer refers to the root γ of the LZ tree. For a new character, insert a new node under the root and set its counter as 1. Exam a sequence from left to right. Suppose an examining character matches with one node of LZ tree. If its next

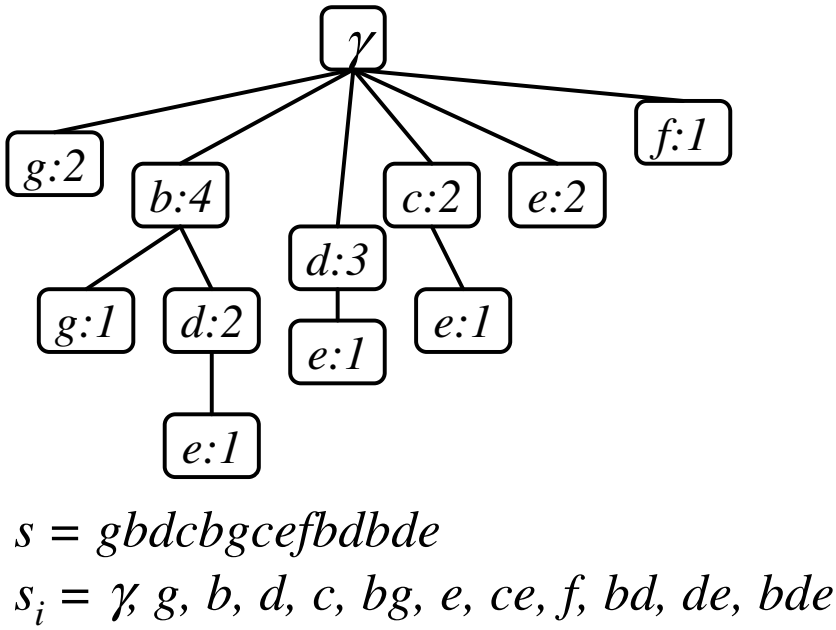


Figure 2.9: LZ Tree [5]

character also matches a child node of the current node, move down to that child node and increase its counter. If not, add a new child under the current node and reset the pointer to the root. Through parsing a sequence, a complete LZ tree can be constructed with statistics of each sub-sequence.

Similar to the $O(k)$ Markov model, the probability that a location context η_{n+1} will happen next can also be estimated by a conditional probability as follow:

$$\hat{P}(X_{n+1} = \eta_{n+1} | \ddot{X} = \ddot{s}) = \frac{N(\ddot{s}, \eta_{n+1} | s)}{N(\ddot{s} | s)}, \tag{2.5}$$

where $\ddot{s}$ is a prefix of a sub-sequence s parsed by LZ algorithm, $\ddot{X}$ is a series random variables for $\ddot{s}$. To predict the next location, we select the location context that is

transited from prefix $\ddot{s}$ with the maximum probability as follow:

$$\eta_{N+1} = \arg\max(\ \hat{P}(X_{N+1} = \eta | \ddot{X} = \ddot{s}\ ),\ \ \eta \in \mathbb{A}', \tag{2.6}$$

where $\mathbb{A}'$ is a finite set of location contexts that followed a prefix $\ddot{s}$ of a sub-sequence*s* before. In a real implement, it matches a path from the root γ to a current node. The next possible location can be determined by selecting one of child nodes that has the maximum counter. If the current node is a leaf, the LZ predictor makes no prediction. LZ predictor requires to maintain a LZ tree and can be executed in an online manner as well.

LZ predictor relaxes the condition of fixed order in Markov model by utilizing finite-length sub-sequences. However, it also has some limitations. (1). As sub-sequences are adjacent each other; any possible pattern that is across two parsed sub-sequences is lost. (2). Also, a possible pattern that is contained within sub-sequence is lost as well. Therefore, LZ predictor improves the fixed order limitation of Markov model, but still has information loss over the entire sequence.

(3) Domain-dependent Predictors

Besides spatial information, is it possible to predict next location from other domain knowledge? Nowadays, location-based social network (LBSN) integrates location tags into the contents of social networks, and brings more variates to new business model. The location tags in LBSN can be provided by user's check-in action or recognized from mobile phone's locations. Cho et al. [30] proposed a human mobility model based on friendships in social network. This interesting work found that short-ranged travel is periodic both spatially and temporally and not effected

by the social network structure, while long-distance travel is more influenced by social network ties. They collected users' friendship information and check-in sequences from location-based social networks. By applying their Periodic Mobility Model (PMM) model, they showed a reliable prediction of locations and dynamics of future human movement.

Meanwhile, Ye et al. [31] solved the same prediction problem by a two-stage method. As the check-in data from social network contains millions of distinct locations, it is difficult to predict the next location directly. They proposed a mixed HMM to predict a type of user activity first, and then estimate the possible exact location belonging to this activity. Their method reduced the prediction space and reached better prediction accuracy.

In addition to successful prediction, semantic interpretation of place is especially important, because it can help user to understand the properties of location. Similarly, social network, such as Foursquare, Facebook or Twitter, are also useful channels that offers semantic information. By extracting semantic meaning from tags, text description, comments or even Twitter messages, we are able to annotate the related places. Usually, topic modeling method [32] and classification method [33] are two popular ways in semantic annotation. Topic modeling method considers a location as a document, semantic meanings as topics, and properties of location as key words. Through LDA algorithms, different semantic labels will obtain different probabilities that indicate the correlation to the place. Classification method treats all characteristics of location as a feature vector. By learning a large amount of feature vectors with labeled places, the classifier has the capability to identify

unlabeled places. Furthermore, [34] used crowdsourcing to label the places, which could get more accuracy than machine learning methods.

In this thesis, we will adopt several location predictor mentioned above as our benchmarks for performance evaluations.

2.5 Unusual Mobility Prediction

Most of times people follow their multi-faceted routines every day. However, due to emergency situations or vacations, people may change their mobility patterns temporarily. In this section, we will review some works about detection and prediction of mobility change.

(1) Detection of Mobility Change

Detection of mobility change or outliers has been researched in many works [35] [36] [37]. These works considered to mine abnormal trajectory from a data level. A trajectory can be partitioned into many line segments. It is relatively easy to compare trajectories based on different line segments using distance-based and density-based approaches. Thus, outliers can be detected from comparisons or clustering.

Also, many spatial-temporal features can be extracted from a long trajectory. These features is formalized as a feature vector, inputed into traditional classifiers. By learning the features of trajectories and their labeled categories, future suspicious movements can be detected quickly via classification.

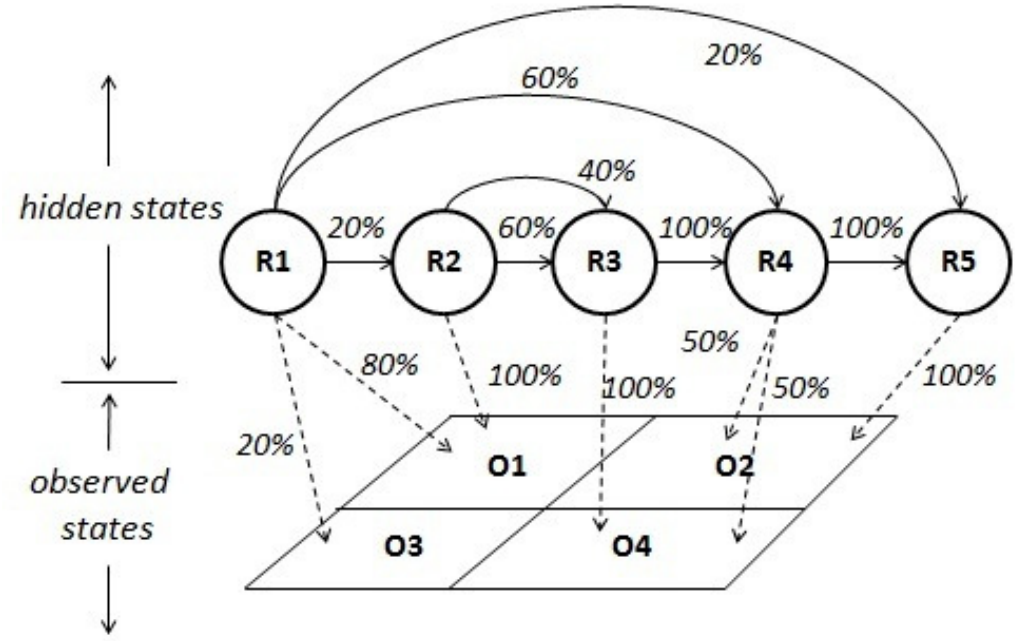


Figure 2.10: Modeling Trajectory by Hidden Markov Model [6]

In [6], the author attempted to detect trajectory changes from a model level. Figure 2.10 shows an illustration of trajectory modeling by hidden Markov model (HMM). He first clustered a user's trajectory coordinates into many density regions, notated as R. These regions may represent a home, a working place or an university in which a user visited frequently. Secondly, he partitioned a 2-D space into many grids O using grid-based discretization. Thus, a trajectory can be mapped into a discrete grid-based sequence. Then, a hidden Markov model was established by assuming each density region R as a hidden state, and each grid as an observation O. Using Baumwelch algorithm, parameters including state transition probabilities, emission probabilities can be learnt from training trajectory. Finally, the mobility change can be found by finding significant changes of these probability parameters via several detection algorithms [38] [39].

However, due to a model based method, parameters of HMM reflect a user's long-term mobility in statistics. Real-time mobility change may not be able to detect immediately, which is not suitable for some online location-based recommendation systems.

(2) Prediction of Mobility Change

Detection of mobility change is straightforward, but is it possible to predict such changes? In practice, many applications require the prediction of mobility change so that recommendations and resource allocation can be prepared in advance. So far, most popular mobility researches utilize regularity of movement and behavior pattern to predict known location. However, few literatures describe the irregularity of mobility trace and the prediction of mobility change. To our best knowledge, only one work [7] attempted to predict the temporary departures from routine.

In their solutions, they first proposed an instantaneous entropy as metric to measure the changeability of an individual mobility. They observed that Shannon entropy can only be used to measure the variety of locations within a periodic time slot. For example, measure the entropy of locations, such as home, school and library, which appear during 13:00-18:00 on Sunday with a probability 60%, 30%, 10% respectively. However, Shannon entropy can not be used to indicate the variety of next location based on all historical locations across different time slots. An alternative way is to formalize the location sequence as a series of random variables $\mathbf{X} = X_0, X_1, \ldots, X_N$, and compute an entropy rate over all variables. The entropy rate

can be expressed as follow:

$$H(\mathbf{X}) = \lim_{N\to+\infty} H(X_N|X_{N-1},\ldots,X_2,X_1) = -\sum_{x_1,\ldots,x_N} p(x_1,\ldots,x_N)\log_2 \frac{p(x_1,\ldots,x_N)}{p(x_1,\ldots,x_{N-1})}, \tag{2.7}$$

where $x_1,\ldots,x_N$ are the observed value of random variable $X_1,\ldots,X_N$. To calculate this conditional entropy, N should be sufficiently large, and the probability $p(x_1,\ldots,x_N)$ exists. In practice, location sequence may not be long enough during several months sampling. Therefore, the entropy rate can only be estimated from a Lempel-Ziv estimator. The Lempel-Ziv estimator is defined as:

$$\hat{H}_N = (\frac{1}{N-1}\sum_{i=2}^{N}\frac{\Lambda_i}{\log_2(i)})^{-1}, \tag{2.8}$$

where Λ_i is the length of the shortest sub-sequence *starting* at position i that is not in the sequence $x_1,\ldots,x_{i-1}$. The advantage of Lempel-Ziv estimator is that it can converge to the true entropy rate rapidly.

The entropy rate is a single value that reflects the change of overall sequence. But it can not answer the question like whether an individual is behaving unpredictably at a position i. In order to do so, they modified the Lempel-Ziv estimator to a real-time entropy estimator. An instantaneous entropy can be calculated to indicate the entropy rate of a per time slot, which is defined as:

$$\tilde{H}_i = \frac{\log_2(i)}{\Gamma_i}, \tag{2.9}$$

where Γ_i is the length of the shortest sub-sequence *ending* at position i, which never occurred from sequence $x_1,\ldots,x_{i-\Gamma_i}$. It is worth to note that Γ is a sub-sequence on

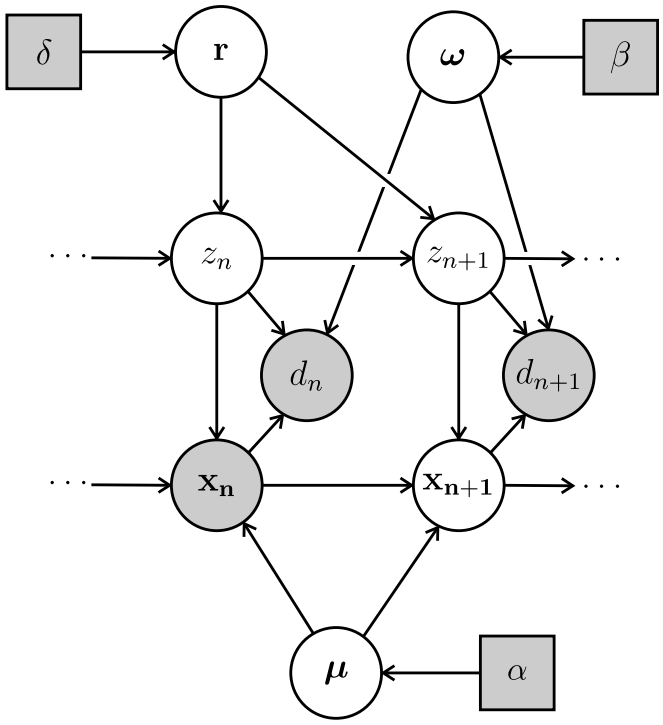


Figure 2.11: Predicting the Break of Habit Using Hidden Markov Model [7]

and before position i, and Λ is a sub-sequence starting at position i. The reason is that the entropy rate can be obtained at position i immediately after calculating the Γ, rather than waiting for the shortest sub-sequence appearing in the future.

In their second work, they used a Bayesian framework to model and predict possible departures in the future mobility. Figure 2.11 shows the Bayesian model. In this model, the departure from routine is regarded as a binary latent state, which is denoted as z. While, x_n is an observation representing the location a user visited at time n. As a spatial-temporal model, context at time d is also considered, which indicates a specific time or calendar.

On spatial aspect, the model is implemented in a fashion of order-1 Markov model, where the probability distribution of the next location is only dependent to the cur-

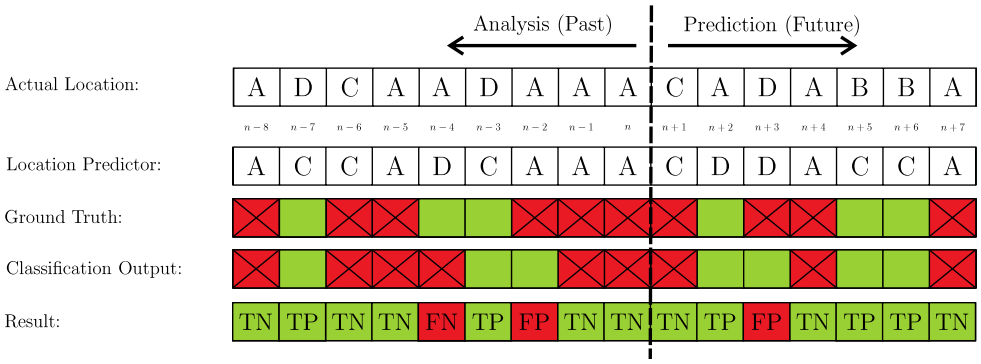


Figure 2.12: Experiments for Predicting the Break of Habit [7]

rent location, $p(\mathbf{x_n}|\mathbf{x_{n-1}},\mathbf{x_{n-2}},\ldots,\mathbf{x_1}) = p(\mathbf{x_n}|\mathbf{x_{n-1}})$. In the transition matrix, the probabilities $p(\mathbf{x_n}|\mathbf{x_{n-1}})$ from location $\mathbf{x_{n-1}}$ to $\mathbf{x_n}$ are governed by a multinomial distribution with Dirichlet priors μ, and α is a hyper-parameter of the Dirichlet distribution. Additionally, the latent state z follows the Markov property as well. $\mathbf{r}$ is a 2*2 transition matrix and δ is a hyper-parameter.

On temporal aspect, they assumed the temporal information d_n only depends on one location x_n at at time n. The dependency probabilities $p(\mathbf{d_n}|\mathbf{x_n})$ are also controlled by another Dirichlet distribution, where ω is the Dirichlet prior and β is the hyper-parameter of the Dirichlet distribution.

Finally, they assumed the independence of spatial and temporal variables. Therefore, conditional probability of the Bayesian can be calculated by

$$p(\mathbf{x_n},\mathbf{d_n}|\mathbf{x_{n-1}},\mu,\omega,\alpha,\beta) = p(\mathbf{x_n}|\mathbf{x_{n-1}},\mu,\alpha)p(\mathbf{d_n}|\mathbf{x_n},\omega,\beta). \tag{2.10}$$

To infer all parameters of Bayesian model, they adopted an Expectation-Maximisation (EM) method, specifically the forward-backward algorithm, to

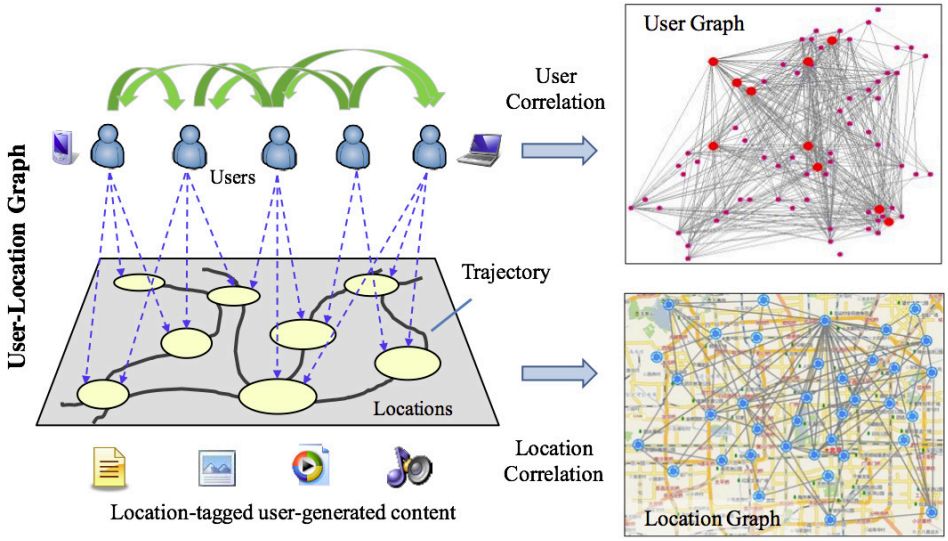


Figure 2.13: Concept of Location-Based Social Network [8]

train the model from historical observation data $\mathbf{X}$. Figure 2.12 shows their experiment settings of training and predicting process. By iteratively updating the posterior distribution of μ, ω and $\mathbf{r}$, the model can be trained for future prediction of habit breaking.

However, in their works, they did not give a formal definition on what is departure from routine. They treated a location which received wrong predictions from different location predictors as a departure place. Using this method, they labeled the ground truth of their experiments, which made their model hard to be interpreted as well as increase the difficulty for result evaluation.

2.6 Location-based Recommendation System

Location-based recommendation is one of important applications for mobility prediction. Bao et al. [8] compared and categorized various recommendations from

three aspects including objectives, methodologies and data sources. In this section, we briefly list these categories. Figure 2.13 illustrates a structure of a location-based social network. Users can be connected each other according to their friendships, or linked in terms of a common location visited before.

(1) Recommendation Objectives

According to the recommendation objectives, there are four types of information that can be recommended:

- Locations or Sequential locations (e.g. POIs, best travel paths);
- Users (e.g. experts, friends, communities);
- Activities (e.g. sport match, concert, promotions);
- Social media (e.g. photo of attractions, travel guide video).

Integrating mobility prediction into a recommendation system, activities, social media can be recommended to smartphone users by knowing their next place in advance. Especially, friends can also be suggested once their predicted locations are close, which may increase the opportunities for the meeting of two old friends. Predicting unusual mobility behavior can also be used for location recommendation. For example, knowing a user is planing to travel in the coming weekend, information of new travel destinations can be delivered to users for references.

(2) Recommendation Methodologies

To realize various recommendations, methods can be classified into the following three groups:

- Content based (e.g. user's age, gender, preference);
- Link analysis based (e.g. use hypertext induced topic search (HITS), PageRank to analyze links among users and locations as shown in Figure 2.13);
- Collaborative filtering (CF) (e.g. inference based on similar users with similar preference).

(3) Data Sources

Recommendation system can utilize various heterogeneous data to infer user's preference, which are listed as follows:

- User profiles (e.g. user's age, gender, interests, preference)
- User geo-located content (e.g. ratings, geo-tags, check-ins on social network);
- User trajectories (CF) (e.g. location sequence obtained from GPS on smartphone).

In this thesis, our experiment datasets for mobility prediction are mainly collected from the check-ins on location-based social networks, and the location sequences recorded by smartphones.

Chapter: 3 Incremental Sequential Pattern Mining for Mobility

Building an accurate mobility prediction model relies on mining useful patterns from historical visiting locations. The frequent questions are: what constitutes a pattern for mobility, and how can we discover patterns efficiently? We can use sequential patterns to represent a user's spatial transitions in history. These patterns can be regarded as building bricks for a mobility model.

Many definitions and mining algorithms of sequential patterns have been proposed in the past decades [18][14]. However, some limitations make traditional works not ideal for applying in mobility scenario. Types of patterns and efficiency of mining algorithms are two major concerns when designing mobility model. The following are some of the examples:

(1) In gene and protein sequence, motifs are the basic components of a DNA sequence [40]. But, motifs are usually found according to the known length and structure. Algorithms for recognizing fixed motif are not suitable for mining mobility patterns with variable lengths.

(2) As discussed in Chapter 2.4, the Lempel-Ziv algorithm has the property to discover non-overlapping sub-sequences [41] [42]. However, these sub-sequences being parsed do not contain any semantic meaning, which makes the interpretation of mobility patterns difficult. Furthermore, any possible patterns across two parsed sub-sequences are ignored.

(3) Finding maximal repeats from a location sequence may be of benefit to explaining user's frequent routines. However, existing algorithms in string processing can not cope with the discovery of the family of maximal repeats in an incremental way.

At the same time, the efficiency problem of sequential pattern mining should be considered in mobility scenario. One of the biggest location-based social networks, Foursquare, has more than 33 million registered users and 3.5 billion check-ins till 2013 reported by the company [43]. About 3 million check-ins on average are generated per day. The workload of servers will be very heavy if a location sequence needs to be re-scanned every time for a new check-in reported in order to find a new mobility pattern. Therefore, deploying an incremental algorithm to discover and maintain patterns becomes a must for mobility service providers.

In this chapter, we will first introduce a set of sequential patterns that aim at handling the special characteristics of location sequences. Secondly, we will present our sequential pattern mining algorithm via pipeline framework, called PipeMining, to discover the corresponding patterns efficiently. Finally, experiments will be demonstrated to evaluate the performance of our algorithm.

3.1 Sequential Pattern for Mobility

Different from text documents such as new articles and books, mobile location sequences have three prominent characteristics.

- (A). Many repeating sub-sequences are contained in a long historical sequence, which may be caused by the visiting of frequent routines; An incremental method is necessary to match and maintain all discovered patterns rather than detecting repeats simply.
- (B). Same location or short sub-sequence may appear consecutively, which make a maximal repeat contain many trivial sub-repeats. Due to the instability of phone signal, miscounting or noise records are inevitable to cause such tandem structure. For example, a user could be recognized as re-entering into the same place multiple times, even he does not move. The following shows a typical example of tandem structure from the real dataset in [9]. Each number in the angle brackets represents a location ID; Location "1845" and short sub-sequence"1845,1847" are tandem together. Mobility data contains more noisy information that need to be recognized and removed.

$$s =< 1845 >< 1845 >< 1847 >< 1845 >< 1847 >< 1926 >< 1845 >.$$

- (C). Location sequence can be viewed as a streaming data, which needs to be updated frequently. A distributed method requires to be considered for a big data scenario.

Hence, many new issues arisen from location sequence are different with traditional text processing. We need to consider pattern matching, noisy data and high volume of updating. Mining maximal repeats can be an appropriate strategy to interpret a user's mobility behaviors, because a repeating sub-sequence should be non-trivial and longest possible. Furthermore, tandem sub-sequences should be detected concurrently in order to classify different types of patterns. Considering the efficiency of mining algorithm, maximal repeat and tandem sub-sequence need to be discovered incrementally.

3.1.1 Family of Maximal Repeats

In this section we will discuss the family of maximal repeats, and formalize the definition of mobility pattern for our mobility scenario. An observation is that a path may be visited once by chance, but it is less likely to be visited more than once. We thus propose to formulate a mobility pattern in terms of repeating sub-sequences (i.e. sub-sequence occurs more than once). Since a location sequence can be viewed as a string, and well-developed notions in string processing can then be employed for our work [18]. Here, we give three definitions in the family of maximal repeat and discuss their properties separately,

DEFINITION 1 (Maximal Repeat) *In a sequence S with L elements, maximal repeat occurs in a pair of identical sub-sequences β_1 and β_2 such that the element to the immediate right (left) of β_1 is different from the element to the immediate right (left) of β_2.*

DEFINITION 2 (Supermaximal Repeat) *A supermaximal repeat is a maximal repeat that never occurs as a sub-sequence of any other maximal repeat.*

Sequence: S_1 = "x a b c a b d y a b c z a b d p a b c a b d q"

- *Maximal Repeats:*
(1). "ab"
(2). "abc"
(3). "abd"
(4). "abcabd"

- *Supermaximal Repeats:*
(1). "abcabd"

- *Near-supermaximal Repeats*
(1). "abc"
(2). "abd"
(3). "abcabd"

Sequence: S_2 = "x a b c a b c y z a b c a b c m"

- *Tandem Repeats:*
(1). "abc"

- *Full-tandem Near-supermaximal Repeats:*
(1). "abcabc"

Sequence: S_3 = "x a b c a b c a y z a b c a b c a m x a b n"

- *Tandem Repeats:*
(1). "abc"

- *Semi-Tandem Near-supermaximal Repeats:*
(1). "abcabca"

- *Non-Tandem Near-supermaximal Repeats:*
(1). "xab"

Figure 3.1: Patterns Discovered in a Sequence

DEFINITION 3 (Near-supermaximal Repeat) *A near-supermaximal repeat is a maximal repeat that occurs at least once in a sequence where it is not contained in another maximal repeat.*

(1) Maximal Repeat

Figure 3.1 shows various patterns discovered in terms of three definitions. First, let us look at the example of maximal repeat. In string S_1, we can find a pair of sub-sequences which have the structures like "$x\beta_1 c$" and "$c\beta_2 d$", where $\beta_1 = \beta_2 =$"ab". If we extend both β_1 and β_2 to the left and right directions, the new pair "$xabc$" and "$cabd$" will be not identical. Thus, the maximal condition is fulfilled and sub-sequence "ab" is a maximal repeat. Similarly, sub-sequences "abc", "abd" and

"*abcabd*" are maximal repeats as well. Note that one mobility pattern may have multiple instances distributed in an individual trajectory at different times. In S_1, pattern "*ab*" has 6 instances. Both pattern "*abc*" and "*abd*" appear 3 times, and pattern "*abcabd*" occurs 2 times respectively.

(2) Supermaximal Repeat

Supermaximal repeat is one of the variants of maximal repeat, which emphasizes a maximal repeat of interest should not be contained in other maximal repeats. That is, if a maximal repeat appears k times, all k instances need to be independent. According to this independent condition, in Figure 3.1, "*ab*" is not a supermaximal repeat, because it is contained in both "*abc*" and "*abd*" 3 times. While, "*abc*" and "*abd*" do not fulfill the independent condition as well, since they belong to "*abcabd*" once. Only "*abcabd*" is a supermaximal repeat. Therefore, the inclusion impose a strong restriction on the maximal repeat. Many potential patterns could be unqualified.

(3) Near-supermaximal Repeat

If a maximal repeat is always embedded in others, like "*ab*", it appears to be insignificant and redundant. Near-supermaximal repeat is another variant that relaxes the constraint of independent condition. Near-supermaximal condition allows a pattern to be contained, but at least 1 instance should be independent. Thus, more interesting patterns have chances to be discovered. One advantage is that sequential information in-between two patterns can be preserved. For the example of S_1, pattern "*abcabd*" covers both "*abc*" and "*abd*". In the meantime, pattern"*abc*" and

"*abd*" are near-supermaximal repeats because sub-sequences "*yabcz*" and "*zabdp*" are not repeats.

In general, *near-supermaximal repeat* is more suitable to be used to define an atomic pattern. Compared with the sub-subsequences parsed by LZ algorithm arbitrarily, the near-supermaximal repeat makes our understanding of a mobility pattern more straightforward and meaningful. Its properties can be summarized as follows:

- The maximal condition ensures a sub-sequence is non-trivial;
- Near-supermaximal definition allows the inclusion of patterns. Each pattern can be an sub-pattern of others;
- Any consecutive patterns that occur more than once is a pattern as well, which minimizes the information loss of trajectory.

3.1.2 Tandem Repeats

Near-supermaximal repeat satisfies our expectation for finding atomic patterns. Nevertheless, there is one exceptional case which can lead a near-supermaximal repeat being non-atomic. That is, tandem repeats are contained in a near-supermaximal repeat. The definition of tandem repeat is given as below:

DEFINITION 4 (k-TANDEM REPEAT) *A tandem repeat is a sequence that is composed of k consecutive copies of identical sub-sequences* β. *It can be represented in the form of* $\beta_1, \ldots, \beta_k$, *where* $\beta_1 = \beta_2 = \ldots = \beta_k$.

According to the number of tandem repeats, we can further classify a near-supermaximal repeat into three combinations: *full-tandem near-supermaxial*,

semi-tandem near-supermaxial and *non-tandem near-supermaxial*, which are defined next:

DEFINITION 5 (Full-tandem Near-supermaximal Repeat) *A full-tandem near-supermaximal repeat is a near-supermaximal repeat, which is completely composed of k consecutive copies of identical sub-sequences.*

DEFINITION 6 (Semi-tandem Near-supermaximal Repeat) *A semi-tandem near-supermaximal repeat is a near-supermaximal repeat, which is partially composed of k consecutive copies of identical sub-sequences.*

DEFINITION 7 (Non-tandem Near-supermaximal Repeat) *A non-tandem near-supermaximal repeat is a near-supermaximal repeat, in which there is no consecutive copy of identical sub-sequences.*

Let us take the following two sequences to illustrate above cases.

S_2 = *"xabcabcyzabcabcm"*;

S_3 = *"xabcabcayzabcabcamxabn"*.

In string S_2, the repeat "*abcabc*" is a near-supermaxial repeat. Note that it can be further partitioned into two consecutive identical repeats "*abc*". Thus, "*abcabc*" becomes a *full-tandem near-supermaximal repeat*, and "*abc*" is called a *tandem unit*. For the purpose of keeping pattern irredundant, "*abc*" should be accepted as an atomic pattern, because the longer sequence "*abcabc*" can be regarded as a combination of duplicate sub-sequences.

In string S_3, repeat "*abcabca*" contains an extra element "*a*" at the end, which makes the near-supermaximal repeat not completely tandem. We call this pattern as a *semi-tandem near-supermaximal repeat*. Semi-tandem near-supermaximal repeat can also be significant if the proportion of tandem part is much smaller than the rest. Among all patterns, only "*xab*" is a *non-tandem near-supermaximal repeat*.

Detecting tandem structure is as important as mining near-supermaximal repeat, because we can further refine a near-supermaximal repeat into an atomic pattern. Also, removing tandem sub-sequence can reduce the noise to the mobility model. For mobility modeling, if we want to keep pattern atomic completely, we can only use the *non-tandem near-supermaximal repeat* as pattern to avoid some exceptional cases. If we allow some tolerances for the precision model, *semi-tandem near-supermaximal repeat* can be a better choice, because more patterns have chances to be discovered. Moreover, for any mis-recognized location, it is hard to be included in high frequent patterns since it will not appear frequently. By selecting appropriate mobility patterns, we can control the influence of noisy data to the precision of mobility model.

In the rest of this chapter, we need to solve the following problems. Given a location sequence $S_{1:N} = \eta_1, \ldots, \eta_N$,

- how to discover all *near-supermaximal repeats* from η_1 to η_N incrementally;
- how to discover all *tandem repeats* in the near-supermaximal repeats concurrently.

Table 3.1 lists the key notations used in this chapter.

Table 3.1. Notations Used in Chapter 3

Notation	Description
$S_{1:N}$	Location sequence from time t_1 to time t_N
η_N	N-th location in $S_{1:N}$
β	Mobility pattern
γ	Repeating sequence
$\boldsymbol{P} = \{p_1, \ldots, p_j, \ldots, p_L\}$	Pipeline framework
v	Cache position on suffix tree

3.2 Data Structure for Repeat Discovery

Repeat discovery requires many comparisons of sequences. In order to speed up searching, we adopt suffix tree as an underlying data structure. In this section, we will briefly introduce the online construction of suffix tree.

A suffix tree can be constructed in many ways. One intuitive method is to extract N suffixes of a N-length sequence, and insert them into a trie tree respectively. This method requires $O(N^2)$ time, and can not be executed incrementally.

As discussed above, online pattern discovery is a necessity for our mobility scenario. Fortunately, Ukkonen [17] provided a linear time algorithm for the online construction. He transformed a suffix tree to a more compact structure called *implicit suffix tree* denoted by $I(S)$. Compared with suffix tree, the implicit suffix tree can be transformed by three operations,

- removing all terminal symbols "$";
- removing all edges without label;
- removing all nodes that do not have at least two children.

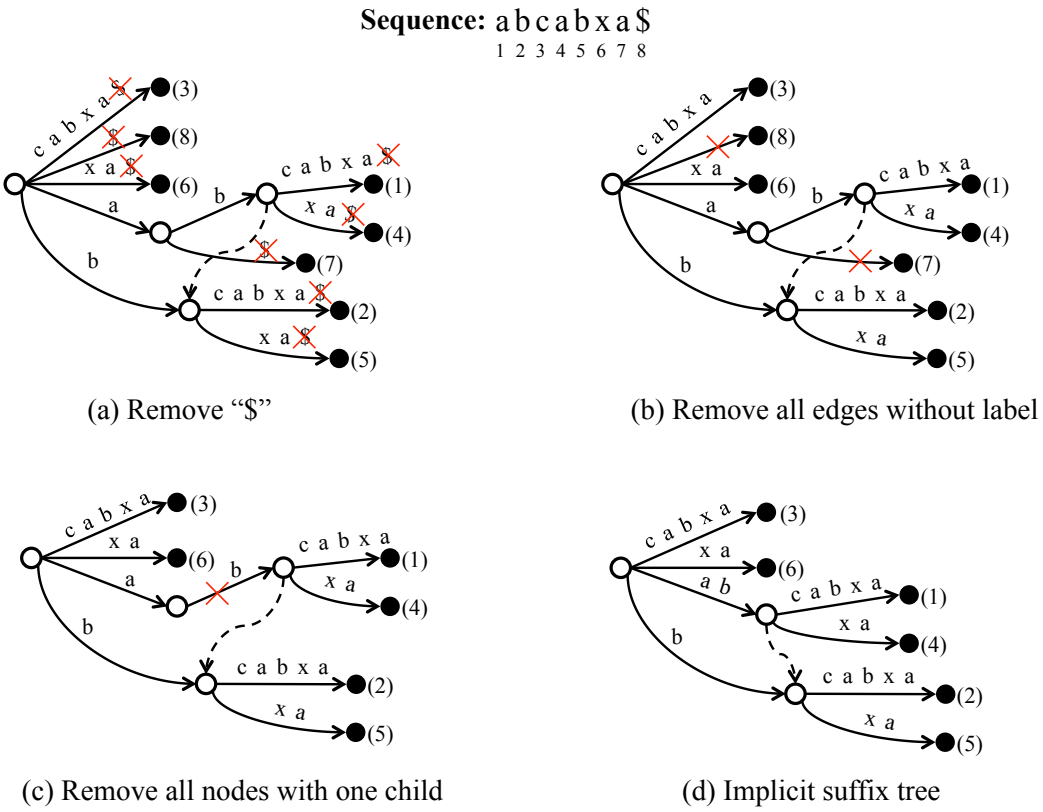


Figure 3.2: Converting Suffix Tree to Implicit Suffix Tree

An example of this transformation is shown in Figure 3.2. The implicit structure is reflected on the compact edges. In a suffix tree, each suffix has a unique path and is corresponding to one leaf node. However, in the implicit suffix tree, several suffixes may share a common edge and have fewer leaf nodes. An edge does not need to be split only if one of edge's characters has different following characters. It can be proved that the implicit suffix tree will equal to a suffix tree once a unique terminal symbol "$" is inserted at the end of sequence.

The implicit suffix tree displays an intermediate state of a suffix tree. It can be built by scanning a sequence from η_1 to η_N. Figure 3.3 demonstrates the evolution of implicit suffix tree. Given a string S = *"abcabxa"*, the Ukkonens algorithm grows

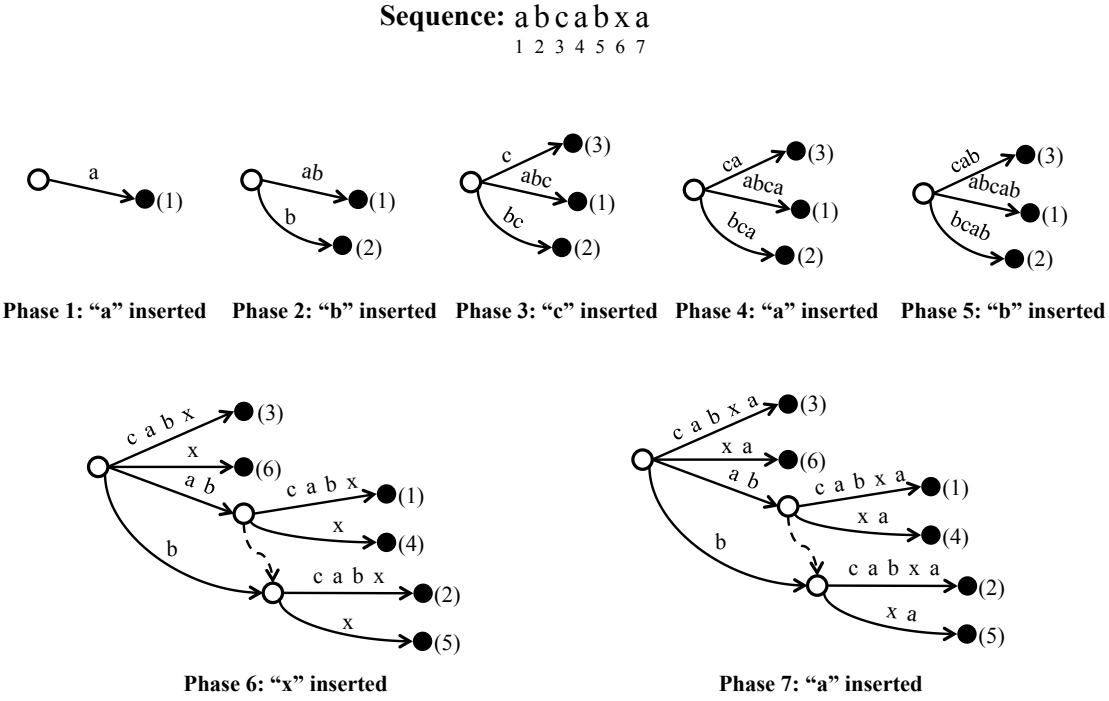


Figure 3.3: Implicit Suffix Tree after Insertion

an implicit suffix tree from $I(S_{1:n})$ to $I(S_{1:n+1})$ by inserting an element η_{n+1}. Each insertion phase can be imagined as appending an element η_{n+1} to all suffixes of $S_{1:n}$. Since the same prefix of suffixes is stored in a common edge, if any sub-sequence before the element η_{n+1} matches with a particular prefix, η_{n+1} only needs to be inserted after this prefix. Meanwhile, other edges of leaf nodes need to be grown by η_{n+1} as well.

In order to indicate the position that an element needs to be inserted after the matched prefix, triple variables, $v(\textit{active node } O, \textit{active edge } E, \textit{active index } j)$, are introduced. Each variable represents a specified node, edge and index. For example, variable $v(\textit{"root"}, \textit{"a"}, 1)$ specifies that the insertion position is the edge starting with *"a"* under the root node, and the index of element in edge is 1. If the

insertion position already has an element $E(j)$, we need to compare the element $E(j)$ with inserting element η_{n+1}, where $1 \leq j \leq |E|$ and $|E|$ is length of edge.

During the process of insertion, three cases are considered.

- Case 1: If η_{n+1} is a new element that never appears in $S_{1:n}$, a new edge with initial element η_{n+1} will be created under the root. Meanwhile, other edges of leaf nodes will be extended by adding η_{n+1}. In Figure 3.3, phase 1-3 show the case of simple extension;
- Case 2: If η_{n+1} exists and is the same as $E(j)$, the edge creation is skipped. We only append η_{n+1} into all edges of leaf nodes, and move the insertion position to the next element $E(j+1)$ of this edge. Meanwhile, we use a counter to memorize the number of total skips. Phase 4 shows the insertion of *"a"* at the position $(\text{"root"}, \text{"a"}, 1)$. Because the first element *"a"* on the edge *"abc"* is matched, edge creation is not necessary. The counter is increased to 1. Phase 5 repeats the same situation;
- Case 3: If η_{n+1} exists and is different with $E(j)$, an edge creation is necessary. The current edge is split from the insertion position, and a new edge with initial element η_{n+1} is linked to an internal node. Meanwhile, append η_{n+1} into other edges of leaf nodes. Phase 6 shows the split of edge *"abcab"* at position 3. The reason is that *"ab"* will be followed by two distinct charters, *"c"* and *"x"* after inserting *"x"*. Moreover, the split of edge *"abcab"* will cause a cascade of splits of other edges, e.g. edge *"bcab"*. The value of counter in Case 2 specifies the number of splits required in this phase. After all splits, the counter is deducted to zero and the insertion position is reset to the root.

So far, the incremental insertion of implicit suffix tree is not linear to the length of sequence. Because all edges of leaf node need to be extended at each phase and edge split is a cascade process. It will be a huge cost if traversing each leaf node from the root every time. Ukkonenn [17] solved this problem by introducing suffix links in-between internal nodes, which is illustrated as a dash line in Figure 3.3. Thus, all leaf nodes can be visited via suffix links directly as well as the cascade of splits can follow the suffix links to find other edges. Moreover, for the storage of suffix tree, we use a pair of pointers (start, end) to represent the position of edge in *S* instead of storing an actual sub-sequence content on the disk. By applying these tricks, Ukkonen's algorithm can be realized in linear time and space.

3.3 Mining Near-supermaximal Patterns

Mobility patterns can be discovered by looking for near-supermaximal repeats and detecting tandem structure. Although similar algorithms [44] [45] have been developed, no algorithm can simultaneously discover near-supermaximal repeat and tandem structure in an incremental way. In this section, we introduce our method for mining near-supermaximal repeats via a pipeline framework. We name it as PipeMining.

3.3.1 Pipeline Framework

As discussed in Sub-section 3.2, the construction of implicit suffix tree requires the comparison of inserting element with the element on a corresponding edge. If they are matched, no edge split is required, and the insertion position will be moved to the next. A repeating sequence can be revealed during the continuous comparisons.

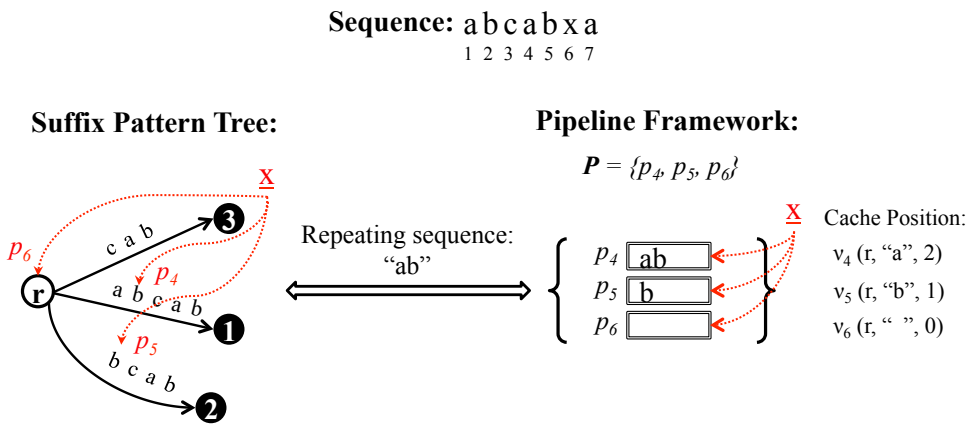


Figure 3.4: Pipeline Framework

In order to discover a maximal repeat, our intuitive idea is to use a pipeline p to cache the repeating sequence γ, which is denoted by $\gamma \in p$.

However, repeats may be embedded in each other. One pipeline is not sufficient to contain all repeats. Hence, we introduce a collection of pipelines to cache all suffixes of a repeating sequence. For a L-length repeating sequence γ, each pipeline p_j stores one of its suffixes $\gamma_{j:L}$, where $1 \leq j \leq L$. The cached suffix $\gamma_{j:L}$ is a sequence that can be exactly found from the root to the insertion position. We name the insertion position as *cache position*, and store it in each pipeline. The pipeline framework can be represented by $\boldsymbol{P} = \{p_1, \ldots, p_j, \ldots, p_L\}$, and sorted from the longest pipeline to the shortest. Figure 3.4 shows the status of three pipelines before inserting *"x"*. The pipeline caches a repeating sequence *"ab"* as well as its suffix *"b"*.

In general, when importing a new element η_{n+1}, the pipeline framework will be performed as follows:

- Create a new pipeline and append it to pipeline framework;
- Pick the longest pipeline and read the memorized cache position;
- Compare the element η_{n+1} with the element $E(j)$ on cache position;
- If η_{n+1} is the same as $E(j)$, append η_{n+1} to pipeline;
- If η_{n+1} is different with $E(j)$, destroy the corresponding pipeline;
- Pick the next pipeline.

In the next section, we will explain why these steps are sufficient to verify the major conditions of near-supermaximal repeats.

3.3.2 Online Detection of Near-supermaximal Repeats

Now, we take the longest pipeline as an example to describe how to detect the near-supermaximal repeat. The definition of near-supermaximal repeat in Section 3.1.1 states two conditions, (1) *maximal condition*, (2) *independent condition*.

(1) Maximal Condition

The maximal condition states that at least there is a pair of identical sub-sequences β, for which the immediate right (left) element to one β should be different from the elements of another. However, before reaching the end of sequence, it is impossible to have all pairs in hands in an online environment. Therefore, we have to check the maximal condition each time when reading a repeating sub-sequence. Fortunately, the maximal repeat can be determined immediately when the right element η_{n+1} following β is different with the previous one at the first time. The reason is that β is completely recorded on the suffix tree after its first appearance. The repeating β will be cached into pipelines as well as compared on the edge of suffix tree. If

the new right element η_{n+1} of β does not continue to be a repeating element, the caching processing is stopped, and the cached longest sub-sequence is the maximal repeat.

So far, the difference of right elements can be detected during the comparison on suffix tree. However, the suffix tree does not record any prefix of a sub-sequence. To guarantee the left elements are different as well, we have the following lemma.

LEMMA 1 *For the longest repeating sequence that is cached in a pipeline, the current immediate left element of the repeating sequence must be different from its previous one.*

PROOF. Suppose a pipeline caches the longest repeating sequence $\gamma \in p$, and γ has two left elements η_l and η_l' in previous sequence. If η_l is the same as η_l', the pipeline p will cache a sub-sequence $\{\eta_l', \gamma\} \in p$ instead of $\gamma \in p$. Because $\{\eta_l', \gamma\}$ itself is a repeating sub-sequence, it will correspond to another branch starting from the root of suffix tree. ■

Lemma 1 allows us to skip the comparison of left element. Therefore, we only need to check the right element η_{n+1} when a repeating sub-sequence is appending to the pipeline framework. If the right element is found to be different with the corresponding one on the edge of suffix tree, the left element must be different as well. Thus, the maximal condition can be fulfilled, and the longest repeating sub-sequence cached in pipeline is a maximal repeat. Figure 3.5 shows the pipeline status after inserting "*x*". As "*ab*" is detected as a maximal repeat, pipelines containing "*ab*" and its suffix "*b*" are required to be destroyed. A new pipeline will be created at phase 7.

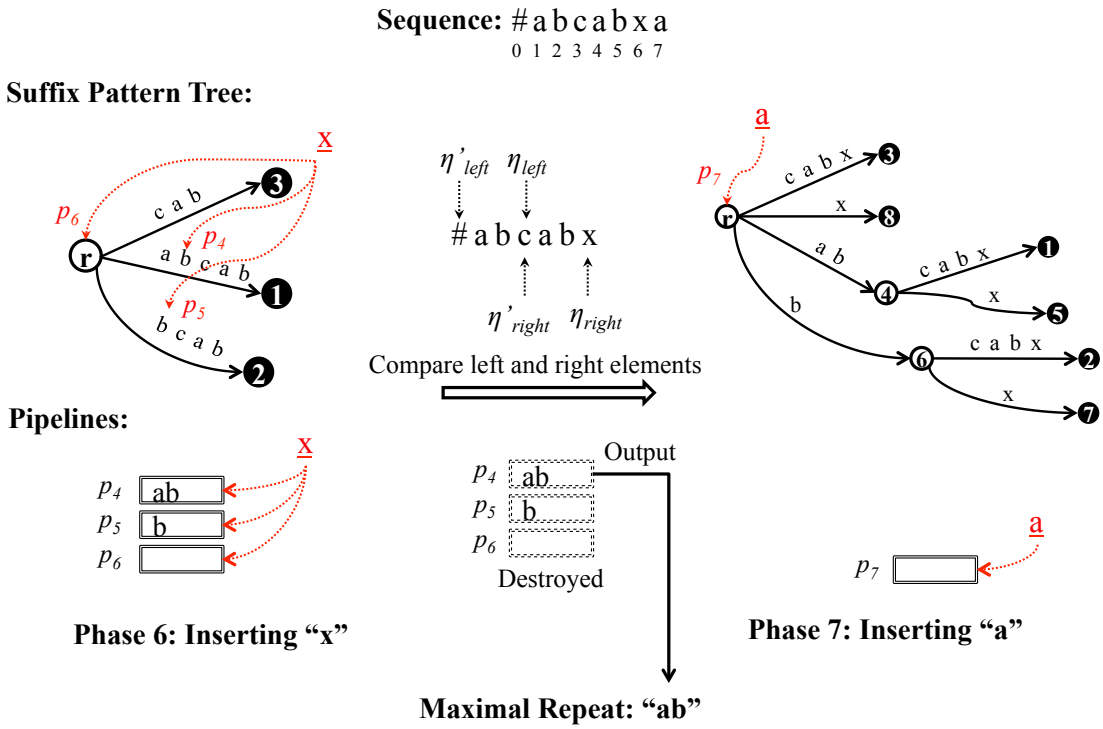


Figure 3.5: Pipeline Destroyed after Maximal Repeat Detected

(2) Independent Condition

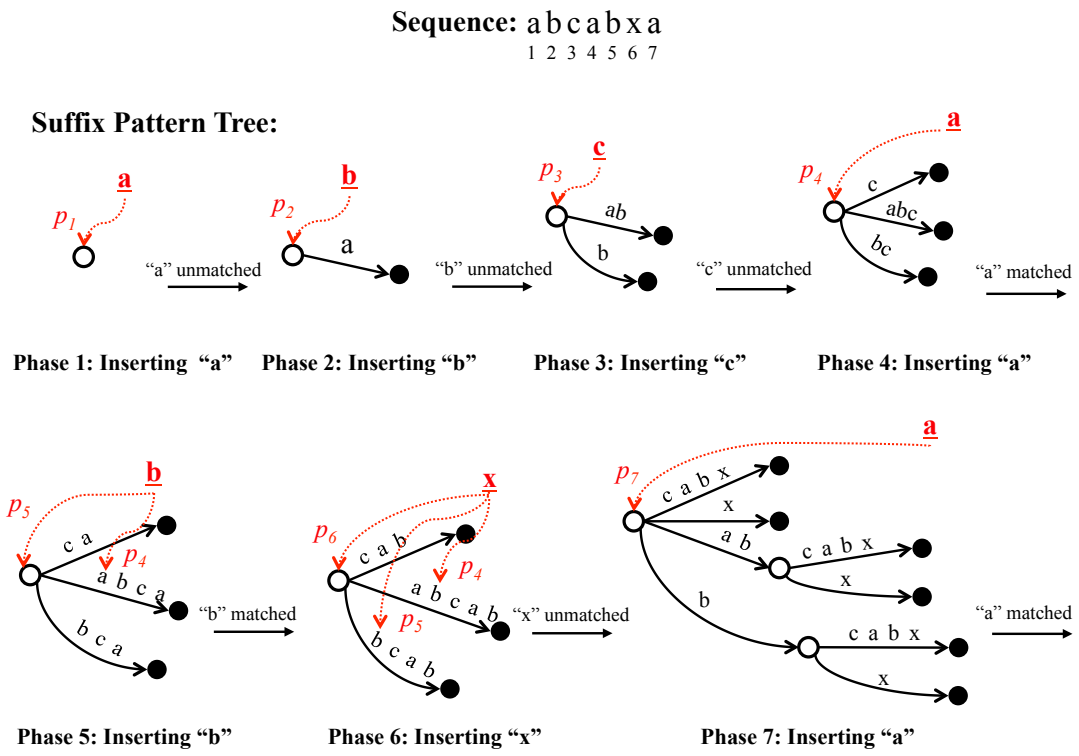
For the near-supermaximal repeat, an additional condition is applied, which requires at least one instance should not be included in others. Similarly, checking the independent condition will cost many position comparisons for all repeats, which is not practical for online processing. Our solution is to count the occurrences of immediate left (right) elements to a maximal repeat β. If there is no combination of left (right) elements (η_l, η_r) that makes sub-sequence "$\eta_l \beta \eta_r$" unique, we can conclude that β is always included in other maximal repeats. To verify this solution, the following lemma need to be proved:

LEMMA 2 *If there is a combination of left (right) elements (η_l, η_r) that makes sub-sequence "$\eta_l \beta \eta_r$" appears only once, the maximal repeat β is a near-supermaximal repeat.*

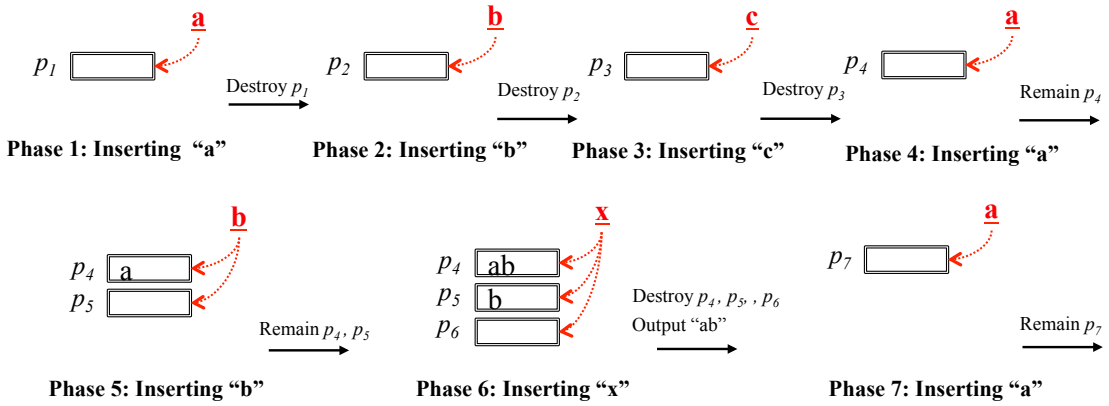
PROOF. In a sequence, any sub-sequence that occurs more than once must be a maximal repeat or a part of maximal repeat. If the sub-sequence "$\eta_l \beta \eta_r$" occurs only once in the entire sequence, there is no chance that any longer maximal repeat will contain "$\eta_l \beta \eta_r$" completely. Hence, the β in-between the elements (η_l, η_r) is an independence instance. ■

Therefore, for *m* combinations of left (right) elements of β, we keep *m* counters. When reading the sequence, we count the appearances of all combinations. Once all values of *m* counters are greater than 1, the repeat β must be contained in other repeats thoroughly, and can not be a near-supermaximal repeat any more. Lemma 1 and Lemma 2 guarantee that the near-supermaximal repeats can be discovered incrementally using pipelines and suffix tree.

Figure 3.6 illustrates the entire flow of inserting string "*abcabxa*" into the suffix tree and the pipeline framework at each phase. From phase 1 to 3, since element "*a*", "*b*" and "*c*" are not repeats, pipelines p_1, p_2 and p_3 are destroyed. At phase 4 and 5, pipelines p_4 and p_5 are remained because a previous sub-sequence "*ab*" is matched consecutively. However, the caching process is stopped when element "*x*" makes "*b*" have two different followers "*x*" and "*c*". Pipelines containing all suffixes of "*ab*" are destroyed and "*ab*" is recognized as a maximal repeat. At phase 7, the cache position is reset to the root, and growth of pipeline framework continues.



(a) Insertion on Implicit Suffix Tree



(b) Status of Pipeline Framework

Figure 3.6: Flow of Detecting Near-supermaximal Repeats

3.4 Mining Patterns with Tandem Structure

For a location sequence, it has a great chance that tandem structure appears in a near-supermaximal repeat, where shorter identical sub-sequences are linked consecutively. To further refine a sequential pattern, in this section, we will describe how to detect the tandem structure concurrently via our pipeline framework.

3.4.1 Online Detection of Tandem Structure

According to the Definition 4, a k-tandem repeat behaves as the connection of k tandem units like $\beta_1, \ldots, \beta_k$, where all βs represent tandem units with the same context. In addition to differentiating the type of near-supermaximal repeat based on the number of tandem units, we also want to utilize the tandem unit as a single sequential pattern, because it is atomic. Therefore, the tandem unit is expected to be discovered incrementally as well. As discussed in Section 3.3.1, each pipeline maintains a cache position. Our strategy is to match these cache positions on the suffix tree in order to find out tandem structure incrementally. There are two situations we need to deal with during the growth of suffix tree, *comparison case* and *estimation case*.

(1) Comparison Case

The comparison case will be illustrated via the following string:

$$S = \text{“}abcababa\text{”}.$$

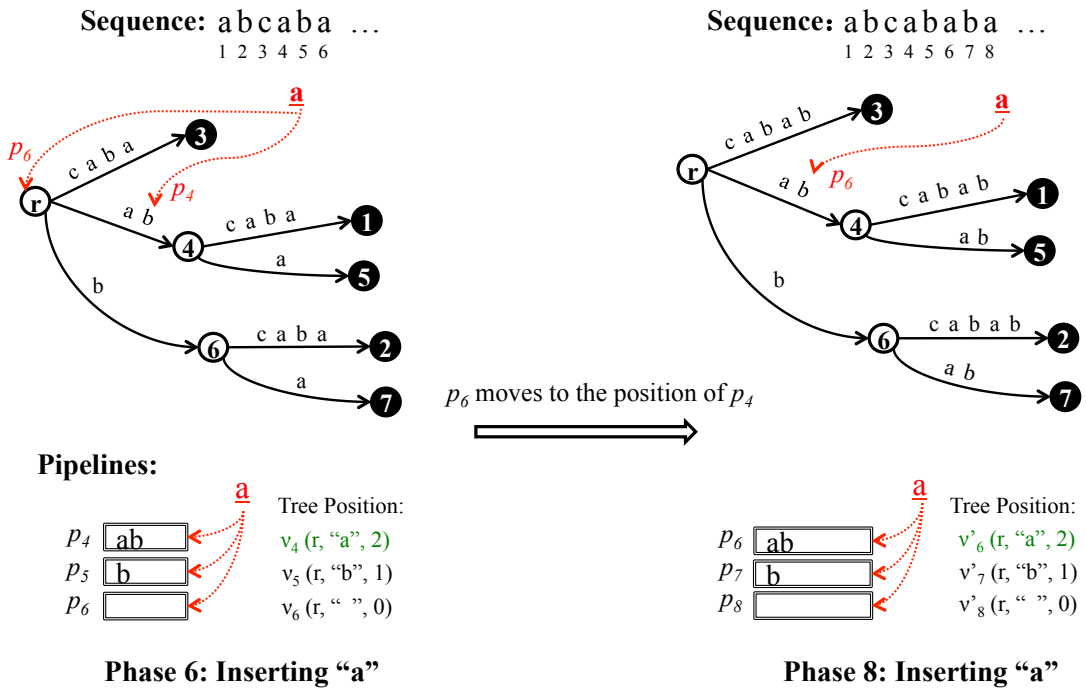


Figure 3.7: Tandem Detection in Comparison Case

Comparison case occurs when a tandem structure can be detected by only comparing the cache positions of relevant pipelines. Figure 3.7 demonstrates an example of comparison case. In each pipeline, a cache position v indicates the position on suffix tree where an element is supposed to be inserted. According to the properties of pipeline and suffix tree, we can have the following observations:

- The longer distance from the root to a cache position is, the more characters are cached in a pipeline;
- If repeating sequences cached in pipelines have the same initial element, their cache positions will start from the same branch under the root.
- If the next element is a part of repeat as well, the cache position of this pipeline will move to the next.

Based on the above observations, we know the cache positions $v_1, \ldots, v_k$ of $\beta_1, \ldots, \beta_k$ will follow the same branch under the root, because all tandem unit βs have the same initial element. Our solution to detect tandem structure is to check whether a later cache position will move to the same position that an earlier cache position has been visited before. If so, the later pipeline must contain a tandem unit β that the earlier pipeline has already. During the growth of the later pipeline, β is also appended to the earlier pipeline. Therefore, the earlier pipeline will have two successive βs. To realize this idea, the following operations need to be performed when a new pipeline containing the initial element η_{n+1} is created:

- Find all existing pipelines $p_1, \ldots, p_L$ that has the same initial element with η_{n+1} as reference pipelines;
- Store the cache positions of all reference pipelines in the new created pipeline p_{L+1} as reference positions;
- When reading the later characters $\eta_{n+2}, \eta_{n+3}, \ldots$, compare the current cache position of p_{L+1} with all reference positions;
- If one of reference positions is the same as the current cache position of p_{L+1}, that reference pipeline contains a tandem structure.

In Figure 3.7, a pipeline p_4 is found as the only reference to the new pipeline p_6 at phase 6. The reference position $v_4(r, \text{“}a\text{”}, 2)$ is marked in p_6. After growing to phase 8, the cache position of pipeline p_6 moves the reference position $v_4(e, \text{“}a\text{”}, 2)$. Thus, a tandem sequence “*abab*” can be determined in pipeline p_4, and sub-sequence “*ab*” will be outputted as a tandem unit.

(2) Estimation Case

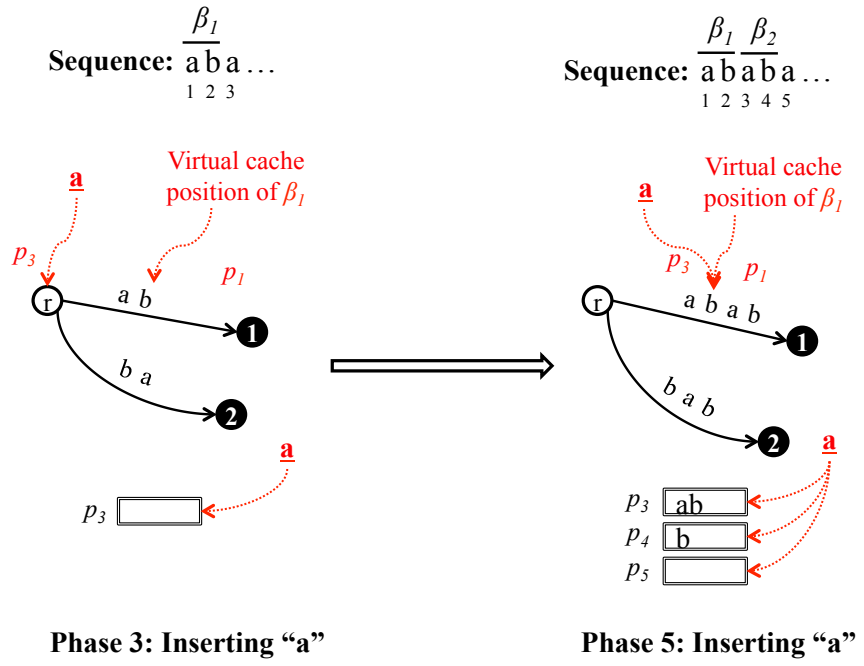
In the comparison case, finding the *reference pipeline* is necessary. For a tandem structure $\beta_1, \ldots, \beta_k$, however, the reference pipeline does not exist when the first tandem unit β_1 is not a repeating sub-sequence. It is possible that the entire β_1 is new or the last element of β_1 is new. For example, in string "*abcxyzxyzx*", "*xyz*" is not a repeat. Or in string "*abcabdabeabea*", "*e*" is not a repeating element for "*abe*". In such a case, pipelines of β_1 have already been destroyed before the arrival of next tandem β_2. It is hard to know the initial element and the cache position of β_1. No reference pipeline can be found for the later pipelines. Hence, the method in comparison case is invalid. Fortunately, without reference pipelines, we still can detect the tandem structure by estimating a *virtual cache position* of β_1. The idea assumes β_1 is a repeating sequence. The virtual cache position is the position after the last element of β_1 is inserted.

Figure 3.8(a) illustrate the estimation case by taking an example of the string:

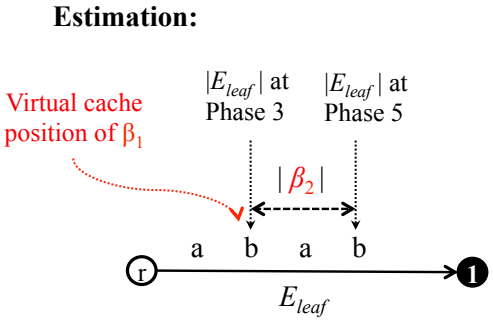
$$S = \textit{"ababa..."}.$$

An important observation is that the virtual cache position of β_1 always stays on the same edge in suffix tree, no matter how many identical tandem subsequences are inserted later. And, this edge is an edge of leaf node. In our example, "*ab*" is a tandem unit and the virtual cache position of β_1 is on the edge $E_{r:1}$. This situation can be verified by proving the following lemma.

LEMMA 3 *For a tandem structure $\beta_1, \ldots, \beta_k$, if β_1 is not a repeat, all successive tandems $\beta_2, \ldots, \beta_k$ are appended to the same edge of the last element of β_1. This edge is an edge of leaf node in the implicit suffix tree.*



(a) Cache position on Suffix Tree



(b) Estimation of Virtual Cache Position

Figure 3.8: Tandem Detection in Estimation Case

PROOF. Since β_1 is not a repeat, a path that exactly labels β_1 can be found from the root to a leaf node. The last element of β_1 is at the edge E_{leaf} of the leaf node. According to the property of implicit suffix tree, if the successive sub-sequence is a repeat, this sub-sequence will be appended to suffix tree directly, and no edge split will be happened. The structure of implicit suffix tree can remain unchanged. Because the successive tandems $\beta_2, \ldots, \beta_k$ are the repeating sequences of β_1, they will be appended to the same edge E_{leaf} directly. ■

According to Lemma 3, the following facts can be inferred:

- Because the pipeline p' containing β_2 has the same initial element as the pipeline p containing β_1, the cache position of p' will traverse the same path as p;
- The cache position of p' will move to the edge E_{leaf} of β_1 eventually;
- The edge E_{leaf} will be grown by appending the content of β_2, and will not be splitted;
- The length of E_{leaf} will be increased by the length of β_2.

Based on the above facts, after processing β_2, we can estimate the virtual cache position of β_1 by deducting the length of β_2 from the length of edge E_{leaf}. For a cache position $v(\textit{active node } O, \textit{active edge } E, \textit{active index } j)$, since the active node and edge are the same for both β_1 and β_2, we only need to compute the index j by the equation:

$$j = |E_{leaf}| - |\beta_2|. \tag{3.1}$$

Table 3.2. Recognition of Different Repeats

Pattern	**Conditions**
Maximal Repeat	Two different right elements η_r.
Supermaximal Repeat	All types of left (right) elements η_l (η_r) that occur only once.
Near-supermaximal Repeat	At least a pair of left (right) elements η_l (η_r) that occurs only once.
Tandem Unit	The cache position v of successive pipeline moving to the cache position v_{ref} of reference pipeline.
Full-tandem Near-supermaximal Repeat	Having near-supermaximal repeat β_{max} and tandem unit β_{tan} with the length $\beta_{max} = k\beta_{tan}$
Semi-tandem Near-supermaximal Repeat	Having near-supermaximal repeat β_{max} and tandem unit β_{tan} with the length $\beta_{max} > k\beta_{tan}$
Non-tandem Near-supermaximal Repeat	No tandem units found.

If the cache position of p' equals to the virtual cache position of p, a tandem structure can be discovered. The pipeline p' containing β_2 will be outputted as a tandem unit. Figure 3.8(b) shows the case of estimation on the leaf edge $E_{r:1}$.

Through the comparison and estimation of cache positions, our PipeMining algorithm can support the detection of tandem structure incrementally.

3.4.2 Recognition of Various Sequential Patterns

In Section 3.3.2 and 3.4.1, we have proposed different techniques to discover near-supermaximal repeat and tandem unit incrementally. The questions are how many kinds of sequential patterns that can be recognized by applying these techniques, and what are the conditions for detecting these patterns? We summarize different patterns with their recognition conditions in Table 3.1.

As discussed in Section 3.1, near-supermaximal has a more general definition and meaningful interpretation for mobility pattern. Our pattern recognition is based on the near-supermaximal repeat. Furthermore, considering the noise of sequence and abnormal mobility pattern, we further filter the near-supermaximal repeat in terms of the number of tandem units contained. Thus, patterns outputted from pipeline framework can be classified into different types, such as semi-tandem near-supermaximal repeat or non-tandem near-supermaximal repeat, to fulfill different application needs.

3.5 Maintenance of Pipelines and Patterns

In the pipeline framework, multiple pipelines are used to cache all suffixes of repeating sequence at the same time. So far, we only use the longest pipeline to discuss the detection of near-supermaximal repeat and tandem unit. For the rest of pipelines, we need to manage them as well. Moreover, during an incremental process, patterns need to be discovered, updated or deleted. In this section, we discuss the maintenance of pipelines and patterns.

3.5.1 Pipeline Maintenance

The life-cycle of a pipeline involves three stages: *creating*, *appending* and *destroying*. Creating and appending of a pipeline is simple. A pipeline is created when a new element is read from sequence. If this element is a part of repeat, it will be appended to all existing pipelines. However, the destruction of pipelines need to be considered case by case. Here, we discuss the destroying stage of pipeline.

(1) Destruction of Pipeline

In a pipeline framework, the pipeline on the top contains the longest repeating sequence γ. And, the rest of pipelines have the suffixes of γ respectively, as shown in Figure 3.6. Since repeats are usually contained or overlapped each other, for a new element η_{n+1}, it may not continue to be a repeat for the longest pipeline, but is still a part of repeat for other pipelines. For example, in a string

$$S = \text{“}x\mathbf{abc}y\mathbf{abc}z\mathbf{cde}m\mathbf{cde}n\mathit{abcdef}\text{”},$$

sub-sequences “*abc*” and “*cde*” are two different near-supermaximal repeats. When reading the last sub-sequence “*abcdef*”, its suffixes “*abc*”, “*bc*” and “*c*” are cached in pipelines. After adding the element “*d*”, sub-sequences “*abcd*” and “*bcd*”are repeats any more. Their corresponding caching processes need to be stopped. However, the pipeline containing “*cd*” can be reserved, because it is still a repeat matching the sub-sequence “*cde*”. Therefore, the destruction of pipeline should be considered separately. Only when the next element can not be appended to a pipeline, the corresponding pipeline need to be destroyed.

(2) Output of Near-supermaximal Repeat from Pipeline

Multiple pipelines may stop caching concurrently, if the next element is not a part of repeat. However, only the longest sub-sequence cached in the top pipeline is necessary to output, because we only look for the near-supermaximal repeat. The rest of shorter pipelines can be destroyed safely without any outputting. Therefore, we need to pass an ordering message to all pipelines before they start. Thus, each pipeline can know its position in the entire framework. For example, the pipeline on the top will receive a message with ordering number 1, and its successor will get the ordering number 2. At each round, the entire flow can be described as follows:

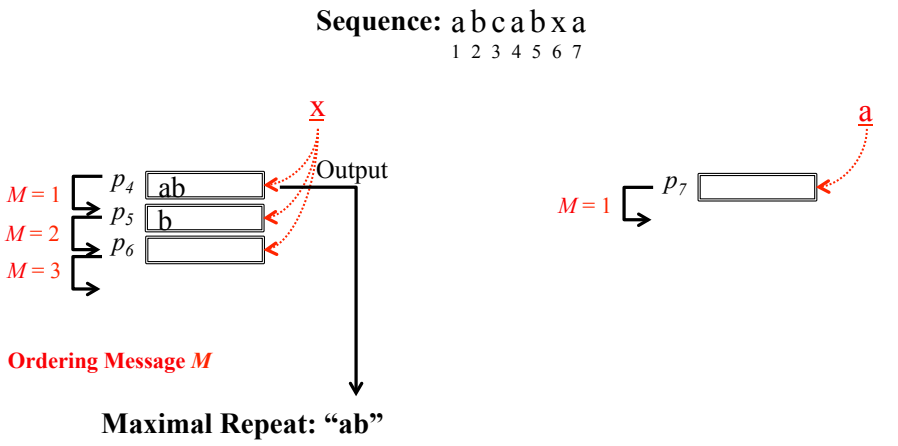


Figure 3.9: Message Passing for Maximal Repeat Detection

- Reset the ordering message to 1;
- Pass the ordering message to a pipeline on the top of pipeline framework;
- If this pipeline can not be appended and its ordering is 1, output its content as a near-supermaximal repeat directly;
- Increase the ordering number and pass the message to the successor pipelines;
- If the successor pipelines are also stopped and their ordering numbers are greater than 1, destroy these pipelines;
- If the successor pipelines can continue to be appended, keep them in the pipeline framework;

Figure 3.9 show the passing of ordering message through the pipeline framework. By doing so, we can make sure only one pattern will be outputted at each round.

(3) Output of Tandem Repeat from Pipeline

Once a tandem unit is detected, pipelines do not require to be destroyed. However, a cascade phenomenon exists in a tandem structure. For example, in a tandem sequence *S*= "*abababab*", "*ab*" is a tandem unit. Meanwhile, "*ba*" could be considered as a tandem unit as well. In such a case, only the first repeating sub-sequence is treated as a pattern, and the cascaded tandem units should be skipped. Here, we give a definition of cascaded tandem unit.

DEFINITION 8 (Cascaded Tandem Unit) *For a tandem unit* $\beta_1, \ldots, \beta_k$*, a sub-sequence is a cascaded tandem unit if it is overlap with two tandem units* β*, and has the same length of* β.

To avoid the mis-recognition of cascaded tandem unit as a pattern, another message passing mechanism is deployed during the tandem detection. The entire flow can be described as follows:

- If the first tandem unit is recognized in a pipeline, output this tandem unit as a pattern;
- Set a message to 1, and pass it to one successor pipeline;
- At the next round, if the successor pipeline also contains a tandem unit and the message received is 1, no output will be generated and pass a message with 1 to its successor;
- If the successor pipeline does not have tandem unit, pass a message with 0 to its successor;
- Wait for the next round.

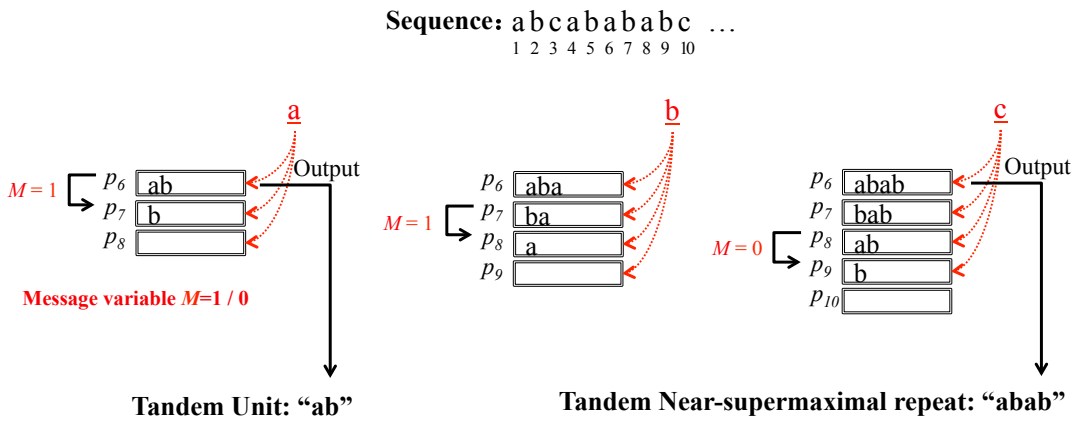


Figure 3.10: Message Passing for Tandem Repeat Detection

Figure 3.10 show the message passing at 3 phases of tandem structure detection. Since only one tandem unit will be generated when reading a new coming element, the message is only passed to one successor pipeline at each round.

3.5.2 Pattern Maintenance

Ukkonen's algorithm only makes the suffix tree be constructed incrementally so that it allows us to discover patterns in online environment. However, a pattern may not appear only once. It may have more instances in the later part of sequence. Hence, the matching of discovered pattern should also be implemented in an online fashion. Thus, we can recognize, update or delete a pattern incrementally.

(1) Structure of Suffix Pattern Tree

As a sequential pattern itself is a sequence as well, we can re-use the implicit suffix tree to maintain all discovered patterns. Our idea is to mark the length of pattern

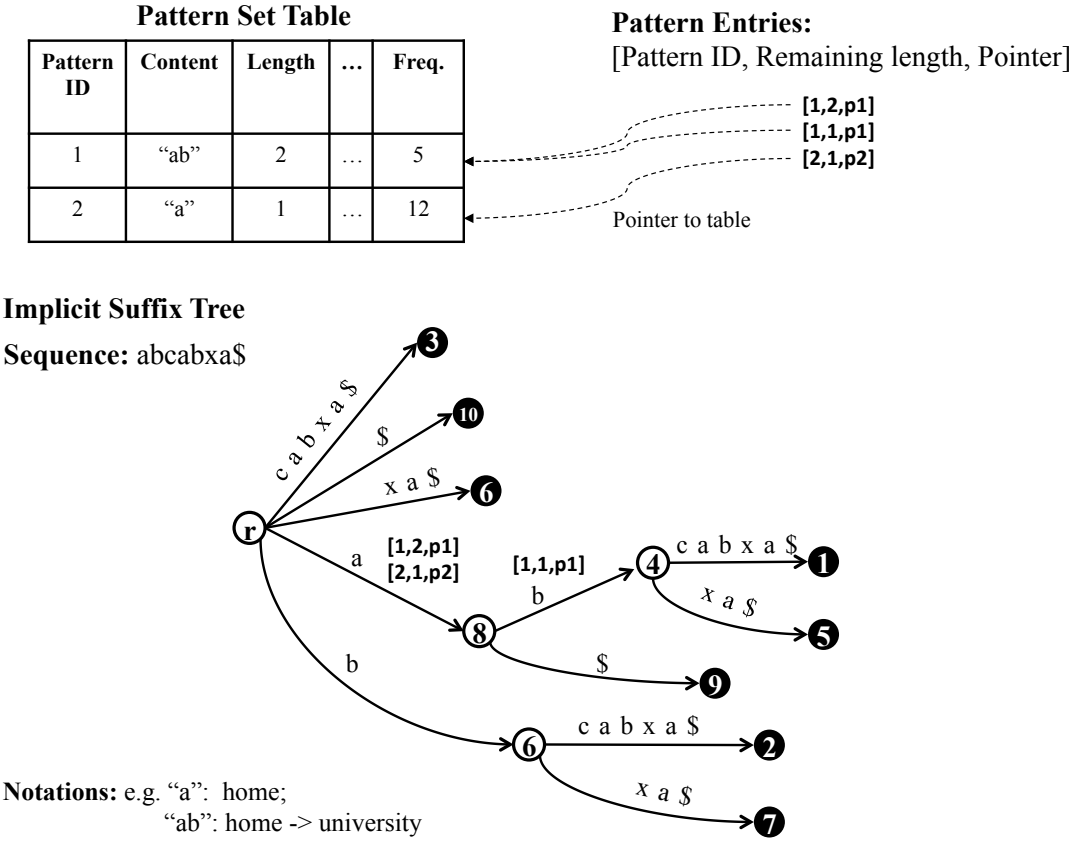


Figure 3.11: Suffix Pattern Tree

on the corresponding edges that are labeled the same content. We name this hybrid index structure as a *suffix pattern tree*. The suffix pattern tree is composed by three components, *implicit suffix tree*, *pattern entry* and *pattern set table*. Figure 3.11 illustrates the connection of implicit suffix tree and pattern set table via pattern entries. Pattern set table maintains all information about a pattern, such as pattern ID, content, length, frequency, position, etc.. Pattern entry contains three pieces of information, *pattern ID*, *remaining length*, and *a pointer*. The pointer refers to a corresponding record on the pattern set table.

In order to mark a pattern on suffix pattern tree, we need store a pattern entry on every corresponding edge that forms the pattern. The pattern entries are distributed from the root to the successive matched edges until no more element can be matched. In every pattern entry, the remaining length specifies how many characters are matched on the current edge as well as under its sub-tree. It can be calculated by deducting the accumulated length of its parent edges from the pattern length. For example, in Figure 3.11, pattern "*ab*" (ID=1) has 2 matched characters remained on edge $E_{r:8}$ and on the successive edge $E_{8:4}$ in the sub-tree. When traversing to the edge $E_{8:4}$, pattern "*ab*" remains only one element "*b*" on this edge.

(2) Matching and Deleting Patterns on Suffix Pattern Tree

When a repeating sub-sequence is cached into a pipeline, its caching position is also moving on the corresponding edges of suffix pattern tree. Once a caching position enters an edge, all pattern entries stored on this edge will be retrieved. We will deduct each remaining length of a pattern entry one by one along the moving of caching position. If a remaining length is reduced to zero finally, the corresponding pattern is matched.

In addition to the matching of pattern, we also keep counters for the pairs of left (right) elements. If the matched pattern does not satisfy the independent condition of the near-supermaximal repeat any more, this pattern should be removed from the pattern set table as well as the suffix pattern tree. With the facilitate of suffix pattern tree, discovery and maintenance of sequential patterns can be realized in an online manner.

3.6 Analysis of Time and Space Complexity

So far, we have introduced the techniques of detecting near-supermaximal repeat and tandem structure. Here, we are also interested to know how efficient our method can be. In this section, we will analyze the time and space complexity of our PipeMining algorithm.

3.6.1 Time and space complexity

Algorithm 3.1 shows the flow for the detection of near-supermaximal and tandem structure in PipeMining. Each time when a new element η_{N+1} is read, all pipelines need to be computed. As noted, the major workload of our algorithm is on the processing of pipelines. The question is how many pipelines need to be handled at each round. Since pipelines are responsible for caching all suffixes of a repeating sequence, the number of pipelines equals to the length of a repeating sequence. For a L-length repeating sequence, L pipelines require to be processed at each round, and time complexity is $O(L)$.

Detection of near-supermaximal repeat requires counting all right and left elements in order to verify the independent condition of near-supermaximal repeat. The number of left and right elements could be large, if a particular pattern appears many times in the entire sequence. Matching left (right) will increase the time complexity. A solution is to deploy a hash map on the right and left elements so that searching of right and left element can be completed in a constant time $O(1)$. Besides, pattern matching can be completed within $O(1)$ time complexity, as we deploy the pattern entries on the suffix pattern tree. Therefore, in general, the time complexity at each

Algorithm 3.1 PipeMining

Input: Next element: η_{N+1}, Suffix pattern tree: T, Pattern set table: $\boldsymbol{\beta}$, New Pattern: β_{new}, Pipeline framework: $\boldsymbol{P}$, Ordering Message: M_{order}, Message of Tandem Repeat: M_{tan},

Output: Collection of patterns returned: $\boldsymbol{\beta}_{ret}$

1: $p_{N+1} \leftarrow$ **AllocateNewPipeline**$(N+1)$
2: $\boldsymbol{\nu}_{ref} \leftarrow$ **FindReferenceCachePosition**$(\boldsymbol{P})$
3: $\boldsymbol{P} \leftarrow$ **AddToPipelineFramework**(p_{N+1})
4: $M_{order} \leftarrow 1$
5: **for** each $p_j \in \boldsymbol{P}$ **do**
6: $M_{order}, M_{tan} \leftarrow$ **GetMessages**(p_j)
7: $is_{repeat} \leftarrow$ **IsRepeatingElement**(T, η_{N+1})
8: **if** $is_{repeat} == true$ **then**
9: $p_j \leftarrow$ **AppendToPipeline**(η_{N+1})
10: **else if** $is_{repeat} == false\ AND\ M_{order} == 1$ **then**
11: $\beta_{new} \leftarrow p_j$
12: $\boldsymbol{\beta}_{ret} \leftarrow \beta_{new}$
13: **DestroyPipeline**(p_j)
14: **else**
15: **DestroyPipeline**(p_j)
16: **end if**
17: $\nu_j \leftarrow$ **GetCachePosition**(p_j)
18: $is_{tandem} \leftarrow$ **TandemDetection**$(\nu_j, \boldsymbol{\nu}_{ref})$
19: **if** $is_{tandem} == true\ AND\ M_{tan} == 0$ **then**
20: $\boldsymbol{\beta}_{ret} \leftarrow p_j$
21: $M_{tan} \leftarrow 1$
22: **end if**
23: $\beta_{old} \leftarrow$ **SearchExistingPattern**(ν_j, T)
24: $\boldsymbol{\beta}_{ret} \leftarrow$ **UpdateMatchedPattern**$(\beta_{old}, \eta_{N+1})$
25: **UpdateMessages**(M_{tan})
26: $M_{order} += 1$
27: **end for**
28: $T \leftarrow$ **InsertToSuffixTree**(η_{N+1})
29: **return** $\boldsymbol{\beta}_{ret}$

incremental step is $O(L)$, and the overall time complexity for a N-length sequence is $O(LN)$. Normally, in most cases, L is much smaller than N in a long sequence. Thus, PipeMining is linear to the time.

Sequence and patterns are stored in the suffix pattern tree and pattern set table respectively. As proved in [17], a suffix tree can be constructed in linear time and linear space with respect to the length of sequence. Pattern set table mainly stores the information of a pattern such as start position, left element, right element. Therefore, pattern set table is increased linear to the number of discovered patterns.

3.6.2 Parallel Execution

For huge customer based service providers, the mobility datasets they collect can be huge. To work on big dataset, it is better if PipeMining can be executed in a parallel environment in order to improve the processing efficiency. To realize the parallel execution, we need to consider both shared resources and programming logic. The shared resources are suffix pattern tree and pattern set table. They will be accessed by multiple pipelines, when comparing repeating sequence and existing patterns. However, the pipeline does not need to write on these shared resources immediately if any patterns are discovered. The updates of patterns can be written back to pattern set table after all pipelines are completed. Only read lock is sufficient to guarantee the synchronization.

From the aspect of programming logic, as shown in Algorithm 3.1, each pipeline can handle the near-supermaximal repeat detection, tandem structure detection, pattern matching and pattern maintenance independently. There is no dependency of other pipelines required during the comparison of caching position for checking the

maximal condition, independent condition and reference positions. The only interaction among pipelines is the message passing for tandem detection as discussed in Sub-section 3.5.1. Fortunately, the message passing is an asynchronous process. During the batch processing of pipelines, it does not need to be passed to another pipeline immediately after it is returned from one pipeline. We can save the message and deliver it to successor pipeline at the next round. Pipeline does not need to wait for any message while others are running.

Algorithm 3.2 Parallel Execution of PipeMining

Input: Next element: η_{N+1}, Suffix pattern tree: T, Pattern set table: $\boldsymbol{\beta}$, New Pattern: β_{new}, Pipeline framework: $\boldsymbol{P}$, Ordering Message: M_{order}, Message of Tandem Repeat: M_{tan},

Output: Collection of patterns returned: $\boldsymbol{\beta}_{ret}$

1: $p_{N+1} \leftarrow$ **AllocateNewPipeline**$(N+1)$
2: $\boldsymbol{\nu}_{ref} \leftarrow$ **FindReferenceCachePosition**$(\boldsymbol{P})$
3: $\boldsymbol{P} \leftarrow$ **AddToPipelineFramework**(p_{N+1})
4: $M_{order} \leftarrow 1$
5: **for** each $p_j \in \boldsymbol{P}$ **do**
6: $\quad M_{order}, M_{tan} \leftarrow$ **GetMessages**(p_j)
7: $\quad$ **CreateThread**$(p_j, M_{order}, M_{tan})$
8: $\quad M_{order} += 1$
9: **end for**
10: $\beta_{new}, \beta_{old}, M_{tan} \leftarrow$ **CollectResultFromPipeline**$(\boldsymbol{P})$
11: **UpdateMessages**(M_{tan})
12: $\boldsymbol{\beta}_{ret} \leftarrow$ **UpdatePatterns**$(\beta_{new}, \beta_{old})$
13: $T \leftarrow$ **InsertToSuffixTree**(η_{N+1})
14: **return** $\boldsymbol{\beta}_{ret}$

Therefore, our algorithm can be partitioned into two stages. In the first stage, we create threads for all pipelines to be executed concurrently. In the second stage, after all pipelines are completed, we update all discovered patterns to the pattern set table, and maintain the message of tandem detection for the next round. Algorithm 3.2 shows the flow of parallel execution, which demonstrates that our pattern mining algorithm can run in parallel in nature.

3.7 Experiments

To demonstrate the performance of PipeMining, in this section, we will introduce our conducted experiments. Our experiments were divided into two parts. The first part was to evaluate the time and storage performance of pipeline framework and suffix pattern tree. The second part was to further investigate various discovered patterns from amounts of moving trajectories.

3.7.1 Performance Evaluation

(1) Datasets and Experiment Settings

Running time and storage consumption are two important factors to evaluate an algorithm. Normally, the length of testing sequence should be long enough so that the performance of algorithm could be observed clearly. However, long location sequence (e.g. more than 100000 locations) is usually not available from the public as well as difficult to be collected. Therefore, in order to measure the performance, we used 11 pure-text English documents as our testing sequences [46]. The length of these sequences vary from 745 to 616041 characters. By removing all punctuations from these documents, the sequences only have 26 kinds of English characters. Detailed information of 11 text documents is listed in the following table. To compare the efficiency of PipeMining, we implemented a traditional algorithm proposed in [18]. The traditional algorithm supports mining maximal repeats and near-supermaximal repeats. However, as the algorithm searches maximal repeats on suffix tree directly, it does not support the incremental mining. Therefore, when reading a new coming element from sequence, the suffix tree was grown incremen-

Table 3.3. Lengths of Text Files

File	pg21782A	pg21782B	pg21782C	pg21782D	pg21782E	pg21782F
Length	746	934	1865	4221	8556	17623
File	pg21782G	pg0766A	pg0766B	pg0766C	pg0766D	
Length	39298	75045	150663	305058	616042	

tally using Ukkonen's algorithm, but the traditional algorithm need to be performed again to find any new maximal repeat.

Since pipeline framework can be executed parallelly, we also implemented PipeMining by multi-threads mode to simulate a parallel environment. As illustrated above, when a repeating sequence was read, it was cached into multiple pipelines. Each pipeline was then put into a new allocated thread for execution. Multiple threads were destroyed together after all pipelines completed their works. Afterwards, the discovered patterns from pipelines were collected and filtered. In this experiment, we controlled the number of threads that could run concurrently. Through tunning the number of threads, we wanted to understand how many threads to use could reach the best performance in parallel environment. All algorithms in our experiments were implemented in C++, and executed on MacOS X 10.10 with 2.3 GHz Intel Core i7 CPU and 16GB memory.

(2) Results

All repeats discovered from PipeMining were the same as the traditional algorithm proposed in [18]. Based on the same results, we then need to evaluate the efficiency of our algorithm. Firstly, we compare the running time of three settings including pipeline framework, pipeline framework using 5 threads and traditional algorithm. Table 3.4 lists three sets of time spent on processing 11 text documents respectively.

Table 3.4. Running Time Comparison

Length	Time (Pipeline)	Time (Pipeline-5Threads)	Time (Traditional Algo)
746	0.051s	0.15s	0.104s
934	0.053s	0.179s	0.171s
1865	0.118s	0.326s	0.679s
4221	0.337s	1.097s	3.867s
8556	0.682s	1.85s	14.69s
17623	1.541s	4.384s	56.469s
39298	4.088s	11.203s	296.049s
75045	8.04s	20.216s	1420.91s
150663	16.679s	42.773s	6153.79s
305058	35.984s	93.395s	26167.9s
616042	78.055s	202.148s	112855s

Figure 3.12 plots three curves for these experiment settings. The time axis of the figure is shown in log scale. According to this experiment result, we can find our method of pipeline framework outperformed the traditional algorithm of mining maximal repeats. Our algorithm achieved 2-1445 times faster than the traditional one. The longer the sequence length was, the more advantages our algorithm could demonstrate. However, the pipeline framework running in multi-thread mode (5 threads) was slower than normal settings by 2-3 times. The main reason is that the efficiency of pipeline processing was high enough in our C++ implementation. The time required for thread creation and mutual exclusive locking on shared resources could be more than the time spent on pipeline processing. Therefore, multi-thread mode increased additional time cost in our single machine deployment. It is possible that the multi-thread mode will be effective, if the pipeline framework is implemented in other high-level languages such as Java or Python.

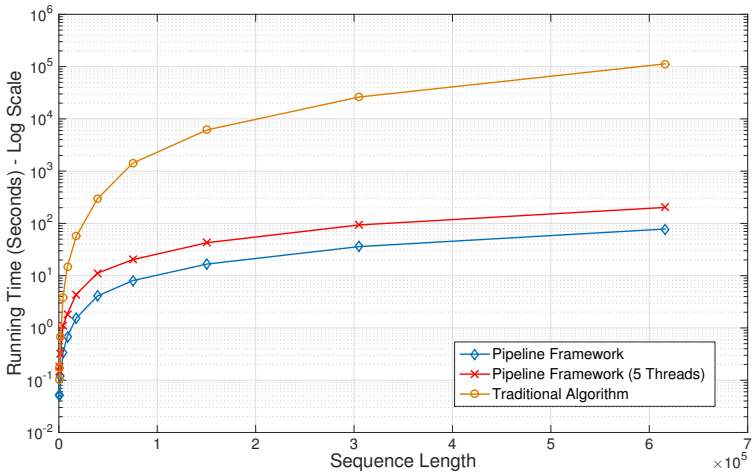
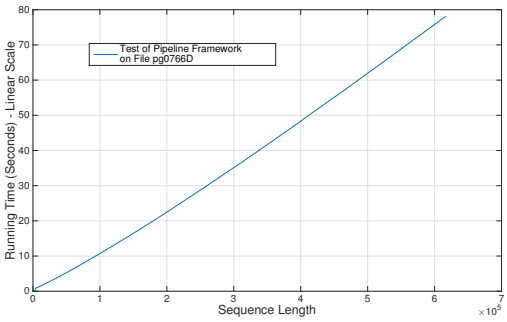


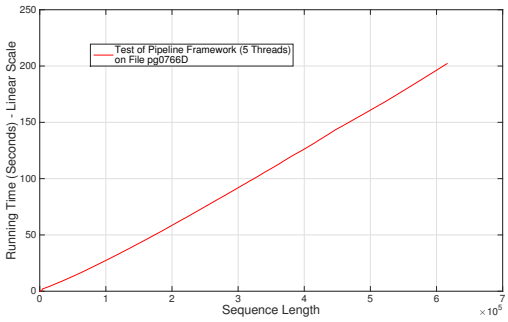
Figure 3.12: Running Time Comparison

Figure 3.13 also plots three sub-figures that demonstrate the running time on the largest text document "pg0766D" by three settings. The time axises of these sub-figures are shown in linear scale. It is obvious that the time required by our pipeline framework was linear to the length of sequence in both normal and multi-thread mode as shown in Figure 3.13(a) and Figure 3.13(b). This conclusion can be derived from the analysis in Section 3.6, which justifies the time complexity of PipeMining is $O(LN)$, where N is the length of a sequence and L is the average length of patterns. However, Figure 3.13(c) shows the traditional algorithm need exponential growth of the time with respect to the increase of sequence length. It is because the size of suffix tree became bigger when more elements were read from sequence. Time of searching suffix tree was increased as well. Therefore, it is impractical to adopt tradition algorithm to discover sequential patterns in a dynamic application.

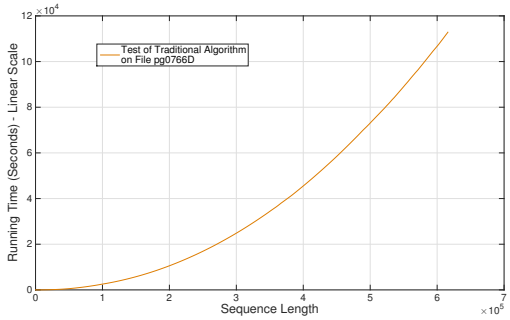
Additionally, Table 3.5 and Figure 3.14 demonstrate the number of nodes in suffix tree, the number of patterns, the memory consumption and the average pattern length over 11 text documents. Since suffix pattern tree maintains all information



(a) Pipeline Framework



(b) Pipeline Framework - 5 Threads



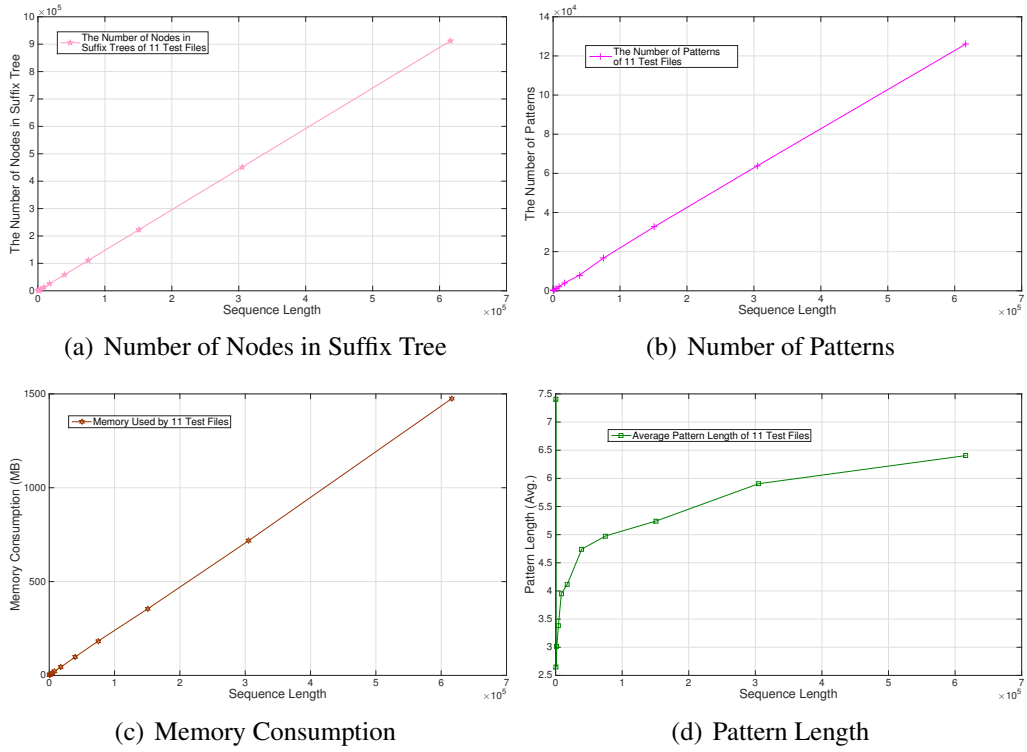
(c) Traditional Algorithm

Figure 3.13: Test of Running Time on File pg0766D

Table 3.5. Running Performance with Sequence Length

Length	Node(s)	Memory	Pattern(s)	Pattern Length (Avg.)
746	1113	3.0 MB	168	7.407
934	1299	3.4 MB	263	2.65
1865	2687	5.7 MB	512	3.022
4221	6146	12.1 MB	1050	3.386
8556	12567	22.5 MB	2029	3.949
17623	25834	43.9 MB	4003	4.12
39298	58142	97.3 MB	7982	4.743
75045	110939	183.1 MB	16656	4.977
150663	222538	354.3 MB	32609	5.241
305058	451433	717.3 MB	63850	5.905
616042	911822	1474.6 MB	125968	6.404

of sequence and patterns, the memory consumption depends on the size of suffix tree as well as the number of patterns discovered. From Figure 3.14(a) and Figure 3.14(b) we can find both the size of suffix tree and the number of patterns achieve linear growth to the sequence length. Therefore, memory consumption can be controlled in a linear range as well.



(a) Number of Nodes in Suffix Tree

(b) Number of Patterns

(c) Memory Consumption

(d) Pattern Length

Figure 3.14: Running Performance with Sequence Length

Finally, we explored how many threads to be allocated was the best for the efficiency of our pipeline framework. Figure 3.15 shows the change of running time with respect to the increase of the number of threads. We can find the pipeline framework work slowest in a single thread setting. With more threads joined, the running time began to decrease until 5 threads were created concurrently. It is obvious that multiple pipelines to be processed concurrently does benefit to the improvement of efficiency. However, as shown in this experiment, the number of threads

should be controlled. Too many threads to be used may increase the time for thread creation as well as have more chances to spend time on waiting for the unlocking of shared resources. We found an appropriate number of threads should be close to the average length of patterns. The average pattern length in file "pg0766D" was 6.404, which means around 6 pipelines were executed concurrently most times. Therefore, the advantage of multiple threads could be taken sufficiently. Note that the single thread in thread mode is different with the normal mode mentioned above, because the normal mode does not require any operation for maintaining thread pool in the real implementation.

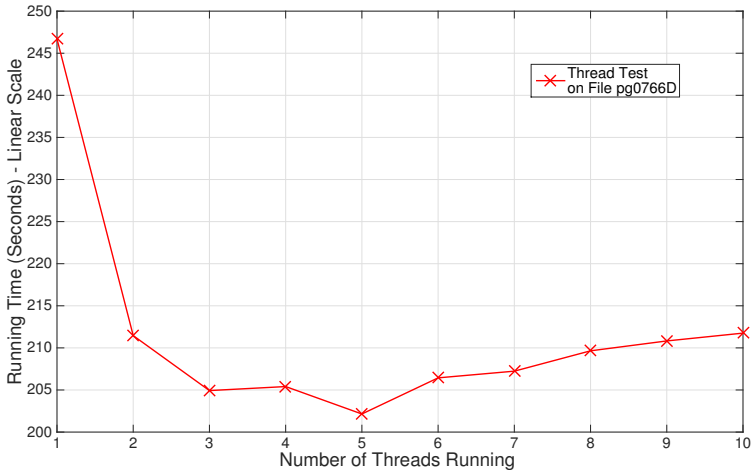


Figure 3.15: Running Time to the Number of Threads

3.7.2 Analysis of Mobility Patterns

(1) Datasets

In addition to the testing of computational performance, we are also interested in knowing how many and what kinds of mobility patterns can be discovered from real moving trajectories. To investigate the mobility patterns contained in people's

daily life, we adopted two sets of location datasets, LBSN check-in sequences and smartphone GPS sequences. LBSN check-ins were generated from two different location based social networks, including Gowalla [30] and Foursquare [47]. Each dataset contains more than ten thousand users, and cover around one year period. Each record contains the information of user ID, latitude, longitude, time and location ID. The location ID specifies a particular place where a user made a check-in, e.g. a restaurant, a hotel or a university.

It is possible that the check-ins may not include user's entire movements in a day, as they are reported by users actively. Hence, we also tested PipeMining on the smartphone GPS data. The real mobile dataset we used is available on CRAWDAD research communities [48] and provided by the research team of Yohan Chon et al [9]. 68 smartphone data donators participated in this project from November 2010 to August 2012. The average collection period was 94 days, and the median was 60 days. A mobile application, "Life Map", was deployed on smartphone to detect GPS, WiFi fingerprints, and smartphone usages (e.g., application, screen status, network traffic and battery level). In their pre-processing stage, they grouped nearby coordinates to a series of locations. The stay duration on a location was more than 10 minutes. We treated each individual trajectory as a location sequence with assigned location IDs, arrival and departure timestamps, and smartphone usage features. Detailed dataset information is listed in Table 3.6.

Each dataset contains various number of users. However, the fact is that not every user has sufficient long location sequence. If a user's trajectory is too short, few patterns will be discovered and they are trivial to our experiment results. Therefore, to guarantee the dataset quality, we filtered out all users who has less than 100

Table 3.6. Statistical Information of Mobility Datasets

Dataset	**Total Users**	**Remained Users**	**Sequence Length (Avg.)**	**Sequence Length (Max.)**	**Period**
Foursquare	11326	4417	244.20	2575	1 / 2011 - 7 / 2011
Gowalla	107092	12746	270.90	2175	2 / 2009 - 10 / 2010
Lifemap	68	68	521.96	3272	11 / 2010 - 8 / 2012

locations. We focused on the remained users after filtering. From Table 3.6, we can find dataset "Life Map" has the best data quality as the average sequence length is around 521, and every data donor has more than 100 locations.

(2) Results

To investigate the mobility pattern, we are interested in two aspects, pattern type and pattern length. The pattern type tells us whether a discovered pattern is full-tandem, semi-tandem, non-tandem, or is a tandem unit. Non-tandem pattern is the most important for us to understand a user's particular routine. Because every location in this routine is non-trivial, it represents a routine from one place to another. However, if a pattern is full-tandem, it implies this pattern may contain noisy information and the pattern itself is trivial. Secondly, the pattern length can tell us the reliability of a pattern. The longer a pattern is, the more reliable this pattern can reflect a user's real motivation. There is less chance that a long pattern can repeat many times. Therefore, through analyzing the pattern type and length, we can determine the reliability of a pattern, which will facilitate our understand of a user's real mobility behaviors.

Table 3.7. Number of Patterns Discovered per User (Avg.)

Dataset	Total Patterns	Full-tandem	Semi-tandem	Non-tandem	Tandem Unit
Foursquare	51.71	7.38	10.38	33.95	13.43
Gowalla	43.63	1.39	1.13	41.12	4.12
Lifemap	103.13	4.88	14.25	84.00	11.81

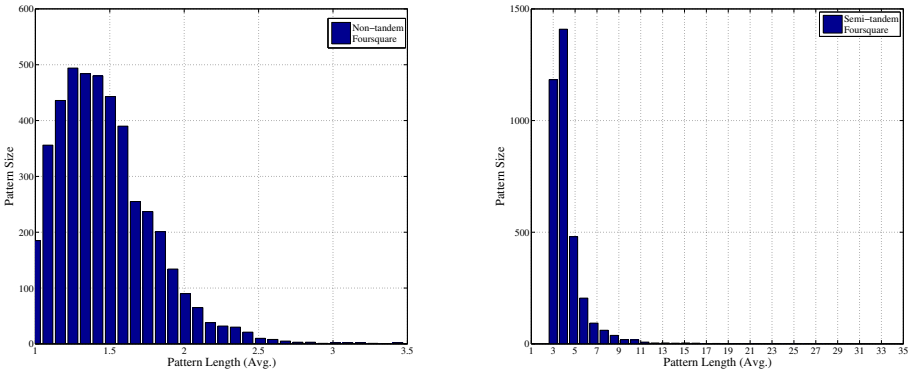
Table 3.8. Pattern Lengths of Different Pattern Types (Avg.)

Dataset	Length	Full-tandem	Semi-tandem	Non-tandem	Tandem Unit
Foursquare	244.20	3.50	4.22	1.47	1.49
Gowalla	270.90	2.49	3.74	1.36	1.22
Lifemap	521.96	3.16	4.41	2.37	1.65

Table 3.7 shows the average number of patterns for each user. First, we can observe the dataset "Life Map" contains the most patterns. Because this dataset have the longest average sequence length (avg. sequence length is 521), it makes sense that more patterns are likely to be mined. Secondly, among three datasets, full-tandem patterns only take a small proportion. The proportion in dataset "Foursquare" is the highest and is around 14%. For other datasets, only 2%-4% patterns are full-tandem. We can conclude dataset "Life Map" has the best data quality and dataset "Foursquare" contains more noisy data. Thirdly, we calculated a ratio between the number of non-tandem patterns and the sequence length. The ratio for "Foursquare" is $33.95/244.2 = 13.9\%$, the ratio for "Gowalla" is $41.12/270.9 = 15.17\%$ and the ratio for "Life Map" is $84/521.96 = 16.1\%$. By comparing three ratios, we can find smartphone records contain more pattern information than check-in datasets. It is possible that for a social network user, he or she would like to check in if this place is the first time to be visited. Therefore, many check-ins are unique and there is no chance to generate patterns from unique locations.

Table 3.9. Pattern Lengths of Different Pattern Types (Max.)

Dataset	Length	Full-tandem	Semi-tandem	Non-tandem	Tandem Unit
Foursquare	2575	7.04	5.19	2.94	3.39
Gowalla	2175	1.81	1.28	2.95	1.52
Lifemap	3272	4.81	5.59	5.75	3.16

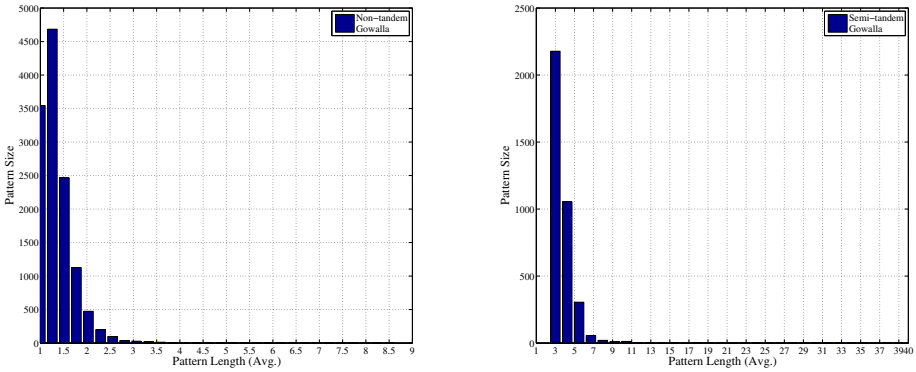


(a) Non-tandem Pattern

(b) Semi-tandem Pattern

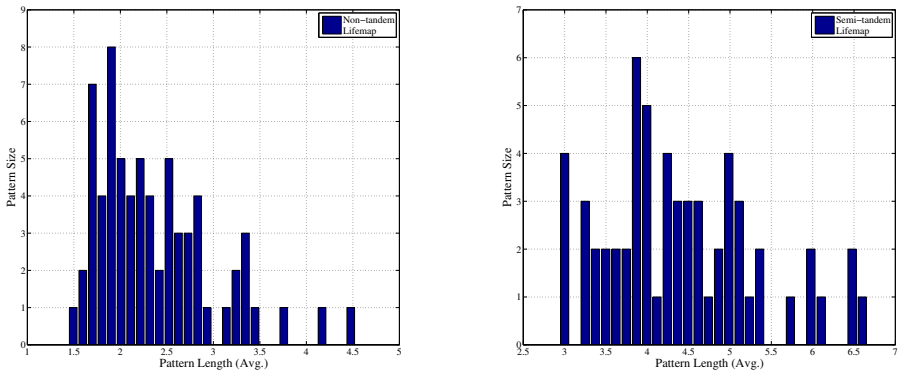
Figure 3.16: Distribution of Pattern Length on Foursquare Dataset

Table 3.8 lists the average pattern length with respect to different pattern types, and Table 3.9 lists the corresponding maximum pattern length. Similarly, we focused on the non-tandem patterns. According to the above tables, we can observe in "Life Map" dataset the average length of non-tandem pattern was 2.37 and maximum pattern length was 5.75. Generally speaking, a user may require to visit 2-5 locations to reach his or her destination. This observation matches our life experience in daily morning or evening commute. However, the length of non-tandem pattern in "Foursquare" and "Gowalla" was relatively short. It makes sense that people have less chance to check-in at the same consecutive locations. Figure 3.16, Figure 3.17 and Figure 3.18 show the distribution of pattern lengths for non-tandem and semi-tandem patterns. By comparing three datasets, we can find the length distribution of non-tandem patterns in "Foursquare" and "Gowalla" datasets are biased on 1



(a) Non-tandem Pattern (b) Semi-tandem Pattern

Figure 3.17: Distribution of Pattern Length on Gowalla Dataset



(a) Non-tandem Pattern (b) Semi-tandem Pattern

Figure 3.18: Distribution of Pattern Length on Lifemap Dataset

location. And, the length distribution of most semi-tandem patterns is biased on 4 locations. However, the pattern lengths of "Life Map" dataset are more equally distributed.

Therefore, based on above analysis, we can conclude that,

- More mobility patterns can be discovered in long location sequence. The longer location sequence we have, the better it is for building mobility model;

- Full-tandem pattern only take a small proportion in a sequence. They are trivial to our understand of people's mobility behaviors and can be discarded;
- People's mobility pattern usually contains 2-5 locations. A higher-order mobility model is suggested to describe the complicated mobility patterns.
- Check-in datasets have relatively few patterns and short pattern length. Smartphone records are more suitable to be the training data for mobility modeling and location prediction.

In general, mining sequential pattern in an incremental way is very important for mobility application. Through our experiments, we verified various mobility patterns exist in people's daily life. Especially, the non-tandem near-supermaximal patterns take a large proportion in all patterns. More patterns will be generated with the accumulation of more trajectories. Obviously, it is low efficiency to discover a new pattern if we need to re-scan all historical trajectories every time. Our algorithm provides 1000 times the speed of the traditional algorithm in mining non-tandem near-supermaximal patterns, which saves the time cost of building mobility model on smartphone side or server side. Furthermore, with the ability of pattern processing in parallel, our algorithm benefits to the location-based service providers in analyzing mobility behaviors of millions of users. Real-time recommendation system can be built based on discovering most frequent patterns, and can update analyzing result incrementally.

3.8 Conclusion

Finding atomic patterns are crucial for us to understand people's mobility behaviors as well as support mobility modeling and prediction. In this chapter, we have

proposed an incremental pattern mining algorithm for location sequence, named PipeMining. PipeMining is the first algorithm to mine near-supermaximal repeat and tandem structure concurrently in an incremental manner. By an incremental pattern discovery method, we can differentiate various pattern types including, full-tandem pattern, semi-tandem pattern and non-tandem pattern. With the knowledge of these pattern types, we can remove trivial and noisy patterns as well as investigate people's meaningful mobility patterns.

PipeMining realizes the pattern discovery via a pipeline method. Through experiments, we have verified that the running time and storage of our algorithm is linear to the length of sequence. It provides a possibility to scale up PipeMining to a big data scenario. Additionally, we also examined the mobility patterns discovered from three real datasets, including check-ins and smartphone records. We found people's mobility pattern usually contains 2-5 locations, and check-in datasets have relatively few patterns and short pattern length. Therefore, a higher-order mobility model is suggested to describe complicated mobility patterns, and smartphone records can be a good dataset for mobility model training.

Chapter: 4 Predicting Next Movement Location

An accurate mobility prediction model is the requisite for many location-based applications. For a private user, knowing his or her next visiting location can facilitate smartphone to estimate traveling time, or recommend popular restaurants around user's destination. For traffic management bureau, predicting next movement of collectives makes crowd control easier by setting up emergency measure in advance before big events.

Usually, a location service provider has millions of registered users. People's moving behaviors can be totally diverse. It is hard to develop different predictors based on different moving habits. Therefore, a general learning model is required, which aims to extract unique characteristic behind each user as well as reduce the work of feature engineering. The trained mobility model can be used to predict user's next movement. Secondly, people's moving paths may change from time to time during different days or months. A mobility model should adapt to the change of user's moving patterns quickly. Thirdly, in Chapter 2, we have reviewed several popular methods on the modeling of mobility behaviors, such as order-k Markov

model, Lempel-Ziv prediction model and hidden Markov model. The advantages of order-k Markov model are that it is efficient, and easy to be implemented. It does not require any complicated assumption, like domain knowledge and feature engineering work. However, the disadvantages are obvious as well. The parameter k is hard to be selected in different applications. Moreover, setting a parameter k arbitrarily may lose higher-order location dependency, which leads a mobility model inaccurate. Without considering any possible meaning of parsed sub-sequence, the predictor works as a black box and is hard to be interpreted from the view of mobility.

Considering the above limitations of traditional approaches, we propose a novel mobility model that has the following features:

- Less hyper-parameter setting and tunning;
- Online training and predicting supported;
- Quick adaption to the change of trajectory;
- Keeping original trajectory information as much as possible;
- Trained mobility model that can be further analyzed and interpreted.

To illustrate the model, our problem can be formalized as follow, and Table 4.1 lists the key notations used in this chapter.

PROBLEM 1 (Location Prediction) *Given a historical location sequence $S_{1:N}$ from time t_1 to t_N, predict the next location η_{N+1}.*

Table 4.1. Notations Used in Chapter 4

Notation	Description
$S_{1:N}$	Location sequence from time t_1 to time t_N
η_N	N-th location in $S_{1:N}$
β	Mobility pattern
γ	Recent traveling path
M	Pattern network model
$\mathcal{B}$	Mobility pattern set
G	Pattern network
ω	Pattern weight
θ	Connection strength
β_{act}	Active pattern
$\acute{\eta}$	Location in mobility pattern
η^{can}	Candidate predicted location
σ	Confidence of candidate location
ξ	Location uncertainty

4.1 Pattern Network

Our proposed method, called pattern network, is a general mobility model. The Pattern network can be used to learn features of moving trajectory and predict the next location. It works by discovering all atomic patterns from a location sequence incrementally, and connecting them as a network. Each pattern is represented as a node, and the occurrence order of two patterns determines their relation. Parameters of nodes and edges indicate the weight of individual pattern and connection strength in-between patterns. In this section, we will describe what kind of patterns are necessary, and how to construct a pattern network.

4.1.1 Partition of Location Sequence

The intuitive idea to model a user's mobility behavior is to find a series of motivations that drive his or her movements. For example, a short sub-sequence (*"Home"* $\rightarrow$ *"Bus Station"* $\rightarrow$ *"Office"*) may represent the action of *"Going to work"*. If a user's motivation can be determined, his or her next movement location can be inferred easily. The question is how can we find these motivations? Our assumption is that if a path is being visited repetitively, the appearance of this path may not be a coincidence. It may be related to a certain motivation. In other words, a path may be visited once by chance, but it is less likely to be visited more than once. Therefore, our strategy is to discover all repetitive sub-sequences and treat them as different motivations. In order to do so, we need to partition a location sequence into many sequential patterns using sequence mining method.

(1) Pattern of Near-supermaximal Repeat

A possible observation of above example is that the motivation of going to work may consist of more concrete motivations such as going to the bus station, reaching the destination and walking into the office. Our expected motivation to be inferred should be atomic but non-trivial. Then, what can be the most appropriate sequential pattern to reflect a user's motivation? Chapter 3 has introduced different definitions of sequential patterns for mobility. We choose the near-supermaximal repeat, because the longest repetitive traveling path can quantify a motivation maximally.

As introduced in Chapter 3, near-supermaximal repeat can be further classified into three types, *full-tandem near-supermaximal repeat*, *semi-tandem near-supermaximal repeat* and *non-tandem near-supermaximal repeat*. Since people may repeat a same routine continuously in a while, a near-supermaximal repeat will contain many *tandem units*, which makes it non-atomic. Differentiation of these repeats allows us to refine mobility patterns and choose suitable one in different scenarios. Usually, non-tandem near-supermaximal repeat is an essential pattern for building pattern network. If a particular sequence has lots of tandem structures, tandem unit will also be considered as one of mobility patterns. It is atomic as well as keeps the core component of full-tandem near-supermaximal repeat. Additionally, semi-tandem near-supermaximal repeat is interesting to be explored in pattern network. Although semi-tandem near-supermaximal repeat contains a tandem structure, it has an atomic presentation, and allows some tolerances as long as the major part of sub-sequence is non-tandem.

(2) Pattern Candidate

Besides repeating sub-sequences, we may have some fresh sub-sequences that appear only once so far. Especially, at the beginning of a location sequence, every location is new to the model. If these sub-sequences are ignored, we will lose some information of the original trajectory. In such a case, we treat a first-time visited path as a pattern candidate, which is defined as follow:

DEFINITION 9 (PATTERN CANDIDATE) *There is a pair of elements η_{left} and η_{right}. Both η_{left} and η_{right} have already appeared more than once. If there is a sub-sequence α in-between η_{left} and η_{right} that occurs only once, α is a pattern candidate.*

If any locations of pattern candidate are visited again in the later stage, these locations must be included in certain repeats. Thus, we should remove them from their belonging pattern candidate. Initially, a pattern candidate could be long. With more and more locations visited, the pattern candidate will shrink until no more one-time visited locations exist. All locations in the pattern candidate migrate to other near-supermaximal repeats.

By introducing pattern candidate and other near-supermaximal repeats, the entire location sequence can be partitioned as a chain of patterns. Since all patterns are connected end by end, it is easy for us to specify the relationship of a pair of patterns according to their relative positions. To conclude the pattern selection, we choose *non-tandem near-supermaximal repeat*, *semi-tandem near-supermaximal repeat*, *tandem unit* and *pattern candidate* as our building bricks for the construction of pattern network.

Figure 4.1 demonstrates an example of pattern discovery from a location sequence. We can find that a mobility pattern may contain multiple locations, and a particular location may belong to multiple mobility patterns as well. Each pattern is regarded as a hidden motivation that leads to a user's particular movement. Different with the arbitrary segmentation of sequence by Markov or LZ model, our method makes the parsed sub-sequence more meaningful.

4.1.2 Construction of Pattern Network

Usually, people's motivations do not stand alone. We can infer the relationship in-between successive mobility patterns. If a user is about to travel at weekend, several actions may be involved in his or her entire movement

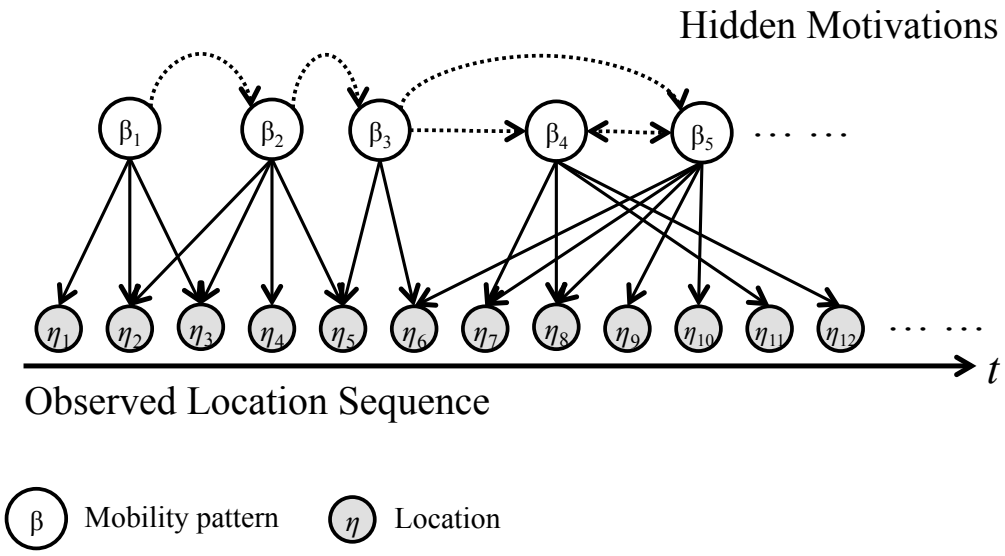


Figure 4.1: Discovery of Mobility Patterns from Location Sequence

in a day, such as *"Leaving home for car park"*, *"Driving to a restaurant"*, *"Driving to railway station"* and so on. As shown in Figure 4.1, a location sequence can be decomposed by the connections of many mobility patterns. Therefore, we can use a network structure to profile the relations of different patterns.

According to the occurrence order of pattern instances in history, connection of two patterns, (β_a, β_b), can be categorized into three major relations, *inclusion*, *overlap* and *adjacency*. Here, we will illustrate each connection respectively.

(1) **Inclusion**

If all elements of β_a appear in β_b with the same order, we call β_a is *contained* in β_b. Meanwhile, the length of β_a must be smaller than β_b, where$|\beta_a| < |\beta_b|$. Also, we

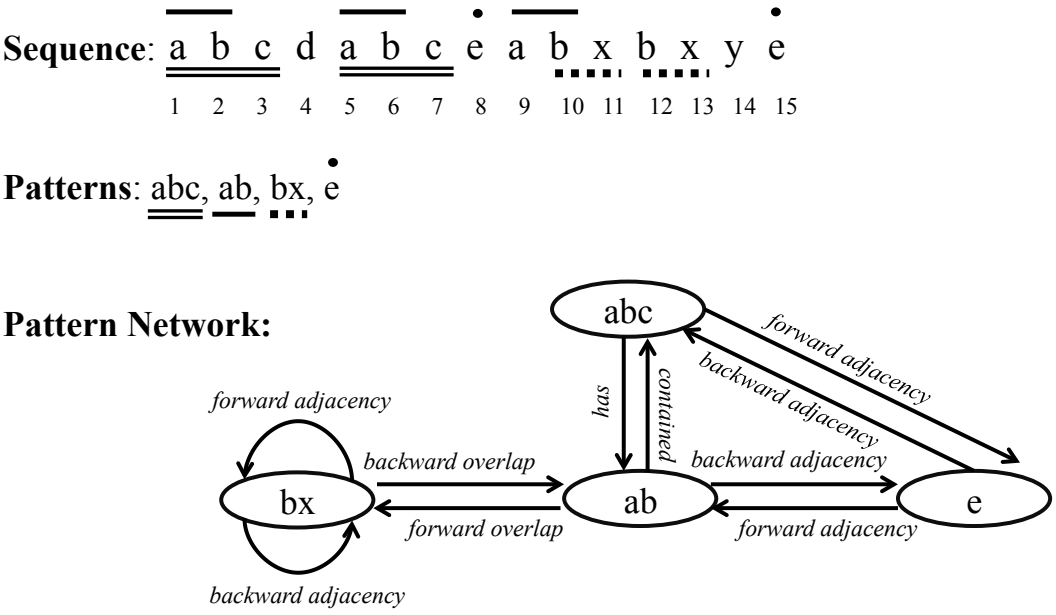


Figure 4.2: Relations of Patterns in a Pattern Network

can say β_b *has* β_a. In Figure 4.2, we can find an example that the pattern *"abc"* has the pattern *"ab"* at position 1-3, and 5-7 respectively.

(2) **Overlap**

Overlap specifies that two patterns share a common sub-sequence. According to the relative positions of β_a and β_b, overlap can be further classified as *forward overlap* and *backward overlap*. If β_b appears on the right side of β_a, β_a is forward overlap to β_b. While, we can also say β_b is backward overlap to β_a. At the position 9, 10, 11 in Figure 4.2, pattern *"ab"* is forward overlap to pattern *"bx"*, and pattern *"bx"* is backward overlap to pattern *"ab"*. Both pattern *"ab"* and *"bx"* share a common element *"b"*.

(3) **Adjacency**

Adjacency denotes the conjunction of two patterns. There is no gap or overlap in-between β_a and β_b. The adjacency can also be classified as *forward adjacency* and *backward adjacency* in terms of pattern's relative positions. If β_b appears on the right side of β_a, and the first element of β_b is right to the last element of β_a, β_a is forward adjacency to β_b. We can also say β_b is backward adjacency to β_a. In Figure 4.2, pattern *"e"* is backward adjacency to pattern *"abc"*, and is forward adjacency to pattern *"ab"* at position 8. It is worth to note two patterns *"bx"* are tandem together along with the positions 10-13.

Figure 4.2 illustrates a pattern network G. Different patterns are denoted by different line styles, which are discovered from a location sequence. The edge with arrow indicates the direction of two patterns, and each edge is labeled with a type of relation. Since we are interested in the prediction of next movement location, *contained*, *forward overlap* and *forward adjacency* are three main relation types that will be used to build pattern network.

Algorithm 4.1 shows the stages for mobility pattern discovery and pattern network construction. The construction of pattern network can be implemented in an incremental way. Every time when a pattern β is discovered, we only need to fetch all neighbor patterns which have the relations of contained, forward overlap or forward adjacency to β. A B+-Tree can be deployed on pattern's start and end positions to facilitate the neighbor search in $O(logN)$.

Algorithm 4.1 Online Pattern Network Construction

Input: Location: η_i, Pattern network: G, Pattern set: β
Output: Updated G

```
β ← DiscoverMobilityPattern(η_i)
if β exists then
    β_neighbors ← SearchNeighbors(β, β)
    for each β_j ∈ β_neighbors do
        relation ← ComputePatternRelation(β_j, β)
        G ← CreateEdge(β_j, β, relation)
    end for
end if
return G
```

4.1.3 Advantages of Pattern Network

Pattern network is a graph presentation of a location sequence. It connects all mobility patterns in history. Several advantages of pattern network can be summarized as follows:

- A pattern network models a mobility from location level to pattern level;
- A pattern network records all transitions of mobility patterns;
- A pattern network keeps original trajectory information as much as possible;
- A pattern network behaviors as a variable-order Markov model.

Figure 4.3 shows the relationships between locations and patterns. A pattern network models a trajectory from location level to pattern level. In the location level, every location is dependent on all previous locations inside its pattern. Obviously, the location transition inside a pattern is deterministic. The fixed sequential order determines which location is the next one. On the other hand, in the pattern network, one pattern connects to many neighbor patterns. The shift in-between two patterns can only be described in probability. We can not exactly tell which suc-

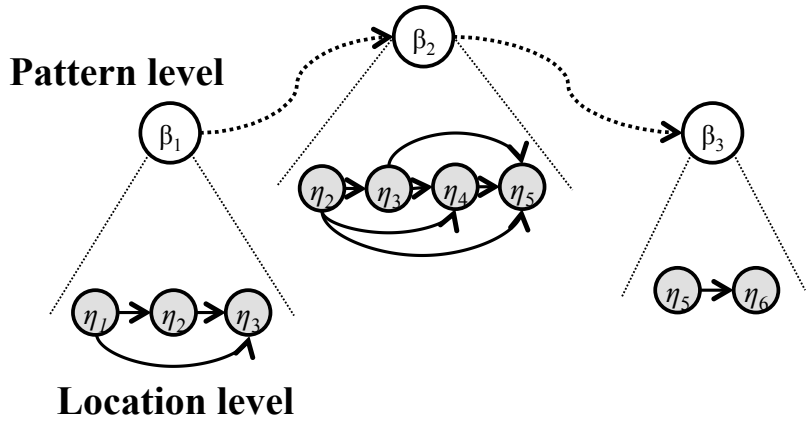


Figure 4.3: Markov Property in Pattern Network

cessive pattern must appear due to the lacking of completed ordering information. Ideally, if a model can remember all past information, it will be perfect for prediction. However, the computational and storage cost will be extremely high as well. The pattern network balances the prediction performance and modeling cost. For the aspect of storage, we limit the memorizing of location transitions within the length of a pattern. For the aspect of computation, only shift of patterns need to be estimated by calculating the probability. Normally, the amount of patterns is much smaller than the length of sequences, which reduces the complexity of computation. Once a successive pattern is inferred, the next location can be determined quickly by checking the sequential order in the next pattern.

Despite pattern network only presents the shift between two patterns, we still have chance to find a super-pattern that memorizes the successive shifts of multiple patterns. For example, if multiple successive patterns appear more than once, a new mobility pattern that contains these patterns will be generated as well. Inside this new pattern, all locations belonging to these patterns are recorded completely.

Therefore, the pattern network can store all sequential information over multiple patterns, and keep original trajectory information as much as possible.

To predict the next location, patterns are utilized to provide possible locations for inference. Since locations in a pattern follow a fixed sequential order, if a pattern is matched partially, the location next to the matched part has large chances to be happened. Since the lengths of patterns are different, the matched length can be variable. Therefore, when using these patterns for prediction, our pattern network model behaviors as a variable-order Markov model. The prediction is based on the variable number of previous locations, which overcomes the limitation of fixed order Markov model.

According to these advantages, pattern network models a location sequence from location level to pattern level, and has less information loss.

4.2 Training of Pattern Network

The pattern network transforms a location sequence to a network topology. In order to infer the next possible pattern, we need to know how important a mobility pattern is, and how likely it will shift to another. Therefore, we introduce two types of parameters, pattern weight ω and connection strength θ, as the attribute of node and edge respectively. With these parameters, we are able to learn features of location sequence. In this section, we will describe how to calculate these parameters and train our pattern network model.

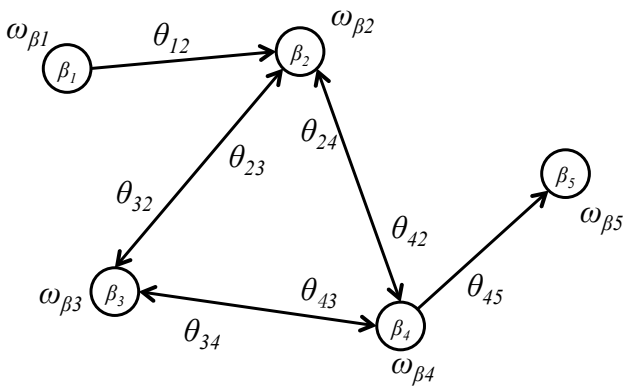


Figure 4.4: Pattern Network Model

4.2.1 Overview

A pattern network model consists of three major components, mobility pattern set β, network structure G and network parameters. The network parameters indicate the weights on all nodes and edges, named as pattern weight ω and connection strength θ respectively. Figure 4.4 shows an example of pattern network model $\mathbb{M}(\beta, G, \omega, \theta)$. It can be constructed incrementally through the following steps:

- A mobility pattern β_{new} is discovered from a location sequence;
- The new pattern β_{new} will be linked to existing patterns that contain, overlap or have adjacency to it;
- A pattern weight ω will be assigned to the new pattern;
- For each edge connecting with β, it will be assigned a connection strength θ.

What are the usages of pattern weight and connection strength? To evaluate the importance of node and edge, one naive method, like Markov or LZ model, is to count the appearances of a pattern, or the number of times a pattern shifting to another in history. However, simple statistic can not reflect the change of movement habit in time. For example, routine (*"Home"* $\rightarrow$ *"University"*) is a frequent pattern for a student, but after graduation, routine (*"Home"* $\rightarrow$ *"Company"*) will dominate a office worker's daily traveling. User's history records will affect current prediction greatly. Obviously, simple counting is not an effective way to model a dynamic problem. Therefore, besides counting, we introduce additional factors, pattern weight ω and connection strength θ, to indicate the current state of pattern and pattern shift. The pattern weight is a real number that will be increased or decreased if the locations of patterns are matched or not. Similarly, the connection strength, quantified as a percentage value, will be adjusted dynamically. Details of parameter computing and application will be introduced in the following sections.

To learn features of a trajectory, our model is trained by a feedback procedure including three stages: *prediction*, *verification*, and *propagation*. Firstly, if some patterns are matched with recent traveling path γ, these patterns will propose their next locations, and the final predicted location will be determined via a majority voting mechanism. In the verification stage, by knowing the actual next location η_{n+1}, pattern's prediction will be verified. Corresponding pattern's weight or connection strength will be increased if a correct prediction is made. Usually, people's mobility patterns are overlapped and nested. If a pattern happened recently, other connected patterns in history may have more chances to happen afterwards. Based on this observation, a portion of adjusted weights will be propagated to the neighbors of active pattern. The propagation stage make our pattern network adapt to the

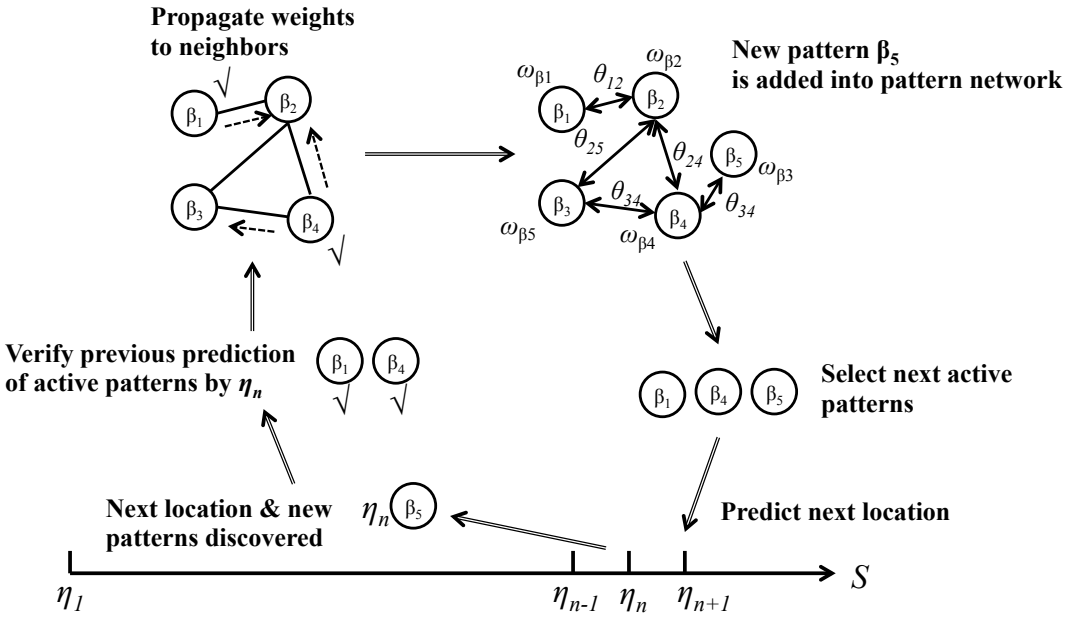


Figure 4.5: Overview of Training Pattern Network

recent trajectory quickly. Figure 4.5 shows the training flow of a pattern network via prediction, verification and propagation.

4.2.2 Prediction of Next Location

How to predict the next location η_{n+1} in pattern network? Consider a location sequence,

$$S_{1:n} = \eta_1, \ldots, \eta_m, \ldots, \eta_n, \;\; 1 \leq m \leq n. \tag{4.1}$$

Before a user reaching the next location η_{n+1}, if a recent traveling path fits a mobility pattern, we call this mobility pattern is an *active pattern*. Here, we give a formal definition as follow:

DEFINITION 10 (ACTIVE PATTERN) *For a mobility pattern and a location sequence, if a prefix of the mobility pattern β is exactly the same as a suffix of location sequence $S_{1:N}$, this pattern is an active pattern, denoted as β_{act}.*

In Chapter 3, mobility patterns are discovered and recognized via suffix pattern tree and pipeline framework. During an online pattern mining, multiple patterns are possible to be recognized if their prefixes are matched with a suffix of location sequence $S_{1:N}$. Therefore, multiple active patterns can be activated concurrently, and the matched sub-sequence can be viewed as a recent traveling path γ. To differentiate the location name in the historical sequence and in a mobility pattern, we use η to represent a location of sequence S, and $\acute{\eta}$ to represent a location in pattern β.

(1) Candidate Location

The idea for prediction is that if majority active patterns are followed by the same location η_{n+1}, this location η_{n+1} has the highest chance to be visited next. In our algorithm, each active pattern need to nominate a candidate location $\acute{\eta}^{can}$, and assign a confidence value σ. A majority voting procedure will select the final predicted location from this pool of candidate locations. The question is how can we choose a candidate location, and what is the confidence value?

There are two situations for choosing candidates according to how many locations are matched in an active pattern. Suppose there is a L-length active pattern $\beta_{act}[\acute{\eta}_1^{act} \dots \acute{\eta}_j^{act}\ \acute{\eta}_{j+1}^{act} \dots \acute{\eta}_L^{act}]$ whose prefix is matched with a recent sub-sequence, which can be denoted as $\beta[\acute{\eta}_1^{act} \dots \acute{\eta}_j^{act}] = S_{m:n}[\eta_m \dots \eta_n]$.

- *SITUATION 1:* If $1 \leq j < L$, the next location $\acute{\eta}_{j+1}^{act}$ can be nominated as a candidate, because the sequential order inside a pattern is fixed.
- *SITUATION 2:* If $j = L$, the current location is the end of a active pattern. At this time, there is no successive location inside a pattern can be proposed. Fortunately, pattern network provides the information to choose candidate from neighbor patterns that have adjacency, overlap or inclusion relations with the active pattern. If an active pattern has multiple neighbors, multiple candidates will be nominated. Table. 4.2 gives a detailed scheme of candidate selection from neighbor patterns, where $\beta[\acute{\eta}_1 \ldots \acute{\eta}_i \ldots]$ represents a neighbor pattern.

Table 4.2. Selection of Candidate Location from Neighbor Patterns

$\acute{\eta}^{can}$	**Relation**	**Neighbor Pattern**	**Condition**
$\acute{\eta}_1^{adj}$	Adjacency	$\beta_{adj}[\acute{\eta}_1^{adj} \ldots]$	$\acute{\eta}_1^{adj}$ is right to $\acute{\eta}_L^{act}$
$\acute{\eta}_{i+1}^{over}$	Overlap	$\beta_{over}[\acute{\eta}_1^{over} \ldots \acute{\eta}_i^{over} \ldots]$	$\beta_{over}[\acute{\eta}_1^{over} \ldots \acute{\eta}_i^{over}] = \beta_{act}[\acute{\eta}_j^{act} \ldots \acute{\eta}_L^{act}]$
$\acute{\eta}_{i+1}^{incl}$	Inclusion	$\beta_{incl}[\acute{\eta}_1^{incl} \ldots \acute{\eta}_i^{incl} \ldots]$	$\beta_{incl}[\acute{\eta}_1^{incl} \ldots \acute{\eta}_i^{incl}] = \beta_{act}[\acute{\eta}_1^{act} \ldots \acute{\eta}_j^{act}]$

(2) Confidence Value

Besides choosing the candidate location $\acute{\eta}^{can}$, each candidate will carry a confidence value σ to specify how much confidence $\acute{\eta}^{can}$ will be the next. For situation 1, the confidence value is calculated from three aspects of active pattern, the number of historical pattern appearances C_β, pattern weight ω_β and matched ratio R_β. The matched ratio quantifies the proportion of a pattern matched with the recent

traveling path, which can be expressed as below,

$$R_{\beta} = \frac{Number\ of\ matched\ locations}{Length\ of\ pattern}. \tag{4.2}$$

Similarly, for situation 2, the confidence value considers the number of historical pattern shifts $C_{\beta,\beta_{neighbor}}$, connection strength $\theta_{\beta,\beta_{neighbor}}$ and pattern shift percentage $R_{\beta,\beta_{neighbor}}$. The pattern shift percentage specifies how many times the shift happened between a pair of patterns over all neighbor patterns as shown below,

$$R_{\beta,\beta_{neighbor}} = \frac{Number\ of\ shifts\ from\ \beta\ to\ \beta_{neighbor}}{Total\ pattern\ shifts\ from\ \beta}. \tag{4.3}$$

Eq. 4.4 shows the calculation of confidence value for two situations, where $sigmod()$ is a sigmoid funciton, and $pow()$ represents R is the exponent of C. Let us take the situation 1 as example. The confidence value is based on the counting of pattern. However, as we discussed above, simple statistic is hard to describe a dynamic model. Therefore, we use a sigmoid function as threshold to control the influence of pattern counting. The output of sigmoid function ranges from zero to one, which depends on the scale of pattern weight. Since the pattern weight will be adjusted from time to time, the confidence value is able to reflect the popularity of a pattern in recent time. Furthermore, we assume the higher matched ratio an active pattern has, the more chances a proposed candidate location will be visited next. Pattern with higher matched ratio should have higher predictability. Hence, the popularity and the matched ratio of pattern are integrated with counting to become

a confidence value for a candidate location.

$$\sigma = \begin{cases} sigmoid(\omega_\beta) * pow(C_\beta, R_\beta) & \text{if } \acute{\eta}^{can} \in \beta \\ sigmoid(\theta_{\beta,\beta_{neighbor}}) * pow(C_{\beta,\beta_{neighbor}}, R_{\beta,\beta_{neighbor}}) & \text{if } \acute{\eta}^{can} \in \beta_{neighbor} \end{cases} \tag{4.4}$$

(3) Majority Voting

After nomination, we group all candidates with the same location together. Suppose we can obtain X candidate groups for different types of candidates, and each group has Y identical candidates. How can we predict which type of candidate is more likely to be occurred next? A majority voting procedure is deployed to select the predicted location from a pool of candidates. Our strategy is to average the confidence value of candidates in the same group as group confidence, and select the group with the highest group confidence. Thus, our problem of location prediction can be solved by Eq. 4.5 and Eq. 4.6, where $x \in [1,X], y \in [1,Y]$.

$$\sigma_x^{group} = \frac{\sum_{y=1}^{Y} \sigma_{xy}}{Y} \tag{4.5}$$

$$\eta_{n+1} = \acute{\eta}_x^{can} = \arg\max_x(\sigma_x^{group}), \;\; \xi \in [1,X] \tag{4.6}$$

(4) Prediction Uncertainty

Besides predicting the next location, we are also interested to know how reliable a prediction is. By grouping candidates and computing group confidence, a dis-

tribution of group confidence over X groups can be obtained. We can utilize this distribution to compute an uncertainty value ξ, which can help us to justify the performance of current prediction and tune the parameters of our model. The way to compute the prediction uncertainty can be derived from the entropy value of group confidence distribution. According to this distribution, the probability of a candidate group can be estimated by dividing its group confidence over the total confidence received from all candidates, as shown in Eq. 4.7.

$$P(\eta_{n+1} = \acute{\eta}_x^{can}) = \frac{\sigma_x^{group}}{\sum_{x=1}^{X} \sigma_x^{group}} \tag{4.7}$$

Then, we use a normalized entropy ξ to describe the distribution of group confidence, which can be computed as follow:

$$\xi = \frac{-\sum_X P(\eta_{n+1} = \acute{\eta}_x^{can}) \ln P(\eta_{n+1} = \acute{\eta}_x^{can})}{\ln X} \quad \xi \in [0,1] \tag{4.8}$$

We can find the entropy value ξ is an ideal indicator to evaluate the uncertainty of prediction as well as to indicate the changeability of next location. If many candidates can be found from active patterns and they have similar confidence, it implies the next location is highly changeable in history. In this case, the entropy value ξ will be close to 1, which means the current prediction is highly uncertain. On the other hand, if the selection of candidates has only a few options, our model will have high confidence. At this time, the entropy value ξ is close to 0. The reason we introduce a normalized entropy is that the number of X candidate groups may vary at each prediction. In order to make prediction uncertainty comparable, the entropy value is required to be normalized by lnX so that the range of ξ is between 0 and 1. Therefore, we use the normalized entropy value as a prediction uncertainty. Algorithm of prediction step is listed in Algorithm 4.2.

Algorithm 4.2 Location Prediction

Input: Location: η_n, Pattern set: β, Pattern Network: G
Output: Predicted next location: η_{n+1}, Prediction Uncertainty: ξ

$\beta_{act} \leftarrow$ **FindActivePatterns**(η_n, β)
for each $\beta_{act} \in \beta_{act}$ **do**
  if *Candidate in Active Pattern* **then**
    $(\acute{\eta}^{can}, \sigma) \leftarrow$ **NominateCandidateInPattern**(β_{act})
  else
    $\beta_{neighbors} \leftarrow$ **SearchNeighbors**(β_{act}, β, G)
    $(\acute{\eta}^{can}, \sigma) \leftarrow$ **NominateCandidateFromNeighbors**$(\beta_{act}, \beta_{neighbors})$
  end if
  $Group_x \leftarrow$ **PushIntoGroup**$(\acute{\eta}^{can}, \sigma)$
end for
$\sigma_x^{group} =$ **ComputeGroupConfidence**$(Group_1, \ldots, Group_X)$
$\acute{\eta}_x^{can} \leftarrow \arg\max_x(\sigma_x^{group})$
$\eta_{n+1} \leftarrow \acute{\eta}_x^{can}$
$\xi \leftarrow$ **ComputePredictionUncertainty**$(Group_1, \ldots, Group_X)$
return η_{n+1}, ξ

4.2.3 Verification

After reaching the next location η_{n+1}, a verification stage is necessary to adjust active pattern's weight or connection strength in terms of its proposed candidate location. Usually, many active patterns will be activated in the prediction stage. How can we make correct ones to be outstanding and receive more weights relatively? By doing so, user's recent moving habits can be reflect on those popular patterns. The pattern weight ω and connection strength θ can be updated by the adjustment weight $\Delta\sigma$ as follows:

$$\Delta\sigma = \begin{cases} \frac{\xi}{P(\acute{\eta}^{can})} & \text{if } \acute{\eta}^{can} \textit{ is correct} \\ -1 * \frac{1-\xi}{1-P(\acute{\eta}^{can})} & \text{if } \acute{\eta}^{can} \textit{ is incorrect} \end{cases} \tag{4.9}$$

$$\begin{cases} \omega_{\beta_{act}} = \omega_{\beta_{act}} + \Delta\sigma & \text{if } \acute{\eta}^{can} \textit{ is from active pattern} \\ \theta_{\beta_{act},\beta_{neighbor}} = \theta_{\beta_{act},\beta_{neighbor}} + \Delta\sigma & \text{if } \acute{\eta}^{can} \textit{ is from neighbor pattern} \end{cases} \tag{4.10}$$

Since the candidate location with the highest confidence value is the final predicted location, our purpose is to award or punish the predicted location. Thus, we especially introduce the prediction uncertainty ξ to tune the pattern. If an active pattern has great confidence, it will dominate the confidence distribution and ξ approaches to 0. For a correct prediction, there is no need to change the current distribution by highlighting any pattern. However, if the prediction of dominating pattern is wrong, we need to punish it by deducting more weights so that the original distribution can be changed. Therefore, the numerator is set to $1-\xi$. Similarly, if there is no active pattern dominating the distribution obviously, ξ is near to 1. Giving additional weight ξ to a correct pattern can increase its confidence on majority voting later. For a wrong prediction, as we are not fully sure which pattern will be popular, we can only change the distribution slightly. Thus, the value of numerator is small.

In our strategy, the verification procedure will compare every candidate location $\acute{\eta}^{can}$ with the actual one η_{n+1}. The denominator $P(\acute{\eta}^{can})$ have the similar effect to accelerate the adjustment. Especially for the pattern that takes small proportion in the distribution, its pattern weight will be increased or decreased heavily. Therefore, the verification stage can accelerate a low frequent pattern to become popular, if this pattern appears successively. Better than a simple statistical counting, introducing prediction uncertainty can not only reflect changeability of mobility trace, but also be used to tune mobility model appropriately.

4.2.4 Propagation

Location prediction is a dynamic process. How can we make the recent patterns become more important than earlier ones? Selecting patterns in a sliding window, or applying a decay function on pattern weights are traditional approaches to deal with a dynamic problem. However, these approaches rely on setting a fixed window length or an arbitrary decay parameter, which are inflexible when facing different mobility traces.

Note that people's mobility patterns are usually overlapped and nested. If a pattern happened recently, other connected patterns in history may have more chances to happen afterwards. In our model, pattern's weight can be further adjusted by a propagation step. After verification, we propagate the adjustment weight $\Delta\sigma$ to active pattern's neighbors via pattern network so that the weights of neighbor patterns can be adjusted as well. The advantages of weight propagation are that a pattern network model can adapt to the recent trajectory as quickly as possible, and the prorogation does not require any parameter.

The question is how many weights should be passed to neighbors? Our method is to use the connection strength θ with sigmoid function to control the proportion of propagation at each round. Eq. 4.11 shows the connection strength θ_{β_a,β_b} from pattern β_a to β_b. A stronger connection strength means shifts between two patterns happened many times in recent time, and this connected pattern is more likely to happen next. As shown in Eq. 4.11, strong connected neighbors will receive more

weights and increase its confidence on future location prediction.

$$\omega_{\beta_b} = \omega_{\beta_b} + \Delta\sigma_{\beta_a} \cdot sigmoid(\theta_{\beta_a,\beta_b}), \tag{4.11}$$

Weight is allowed to be propagated to neighbors' neighbors. The adjustment weight $\Delta\sigma$ can be propagated on the entire pattern network until $\Delta\sigma$ vanishes, or only propagate to 2-3 levels of neighbors. Algorithm of weight propagation is listed in Algorithm 4.3. Through prediction, verification and propagation, a pattern network model $M(\boldsymbol{\beta}, G, \boldsymbol{\omega}, \boldsymbol{\theta})$ is able to learn user's mobility features, make prediction as well as adapt recent trajectory effectively.

Algorithm 4.3 Weight Propagation

Input: Pattern: β, Adjustment weight: $\Delta\sigma_\beta$, Pattern set: $\boldsymbol{\beta}$, Pattern network: G
Output: Updated pattern set: $\boldsymbol{\beta}$

$\boldsymbol{\beta}_{neighbors} \leftarrow$ **SearchNeighbors**$(\beta, \boldsymbol{\beta}, G)$
for each $\beta_j \in \boldsymbol{\beta}_{neighbors}$ **do**
  $\theta_{\beta,\beta_j} \leftarrow$ **GetConnectionStrength**(β, β_j)
  $\omega_{\beta_j} += \Delta\sigma_\beta \cdot sigmoid(\theta_{\beta,\beta_j})$
  $\Delta\sigma_{\beta_j} \leftarrow \Delta\sigma_\beta \cdot sigmoid(\theta_{\beta,\beta_j})$
  $\boldsymbol{\beta} \leftarrow$ **UpdatePatternWeight**$(\beta_j, \omega_{\beta_j})$
  if $\Delta\sigma_{\beta_j} != 0$ **then**
    WeightPropagation$(\beta_j, \Delta\sigma_{\beta_j}, \boldsymbol{\beta}, G)$
  end if
end for
return $\boldsymbol{\beta}$

4.3 Experiments

In this section, we will compare the prediction accuracies of pattern network with other benchmark models, and investigate the properties of pattern network.

4.3.1 Datasets

To evaluate the performance of pattern network, we need to predict the next location of mobility trace. The dataset "Life Map" introduced in Chapter 3 is one of real mobility datasets that was contributed by 67 smartphone users. Besides, we also tested our model on another mobility dataset named "Reality Mining" [49]. The "Reality Mining" dataset was collected from 94 mobile phone users using Nokia 6600. This study was conducted by MIT Media Laboratory between September 2004 and June 2005. Among 94 data donators, 68 were colleagues working in the same building on campus, and the remaining 26 subjects were incoming students at the business school of university. We believed both "Life Map" and "Reality Mining" datasets were good samples for us to test different mobility models, because they records users' daily movements completely.

4.3.2 Experiment Settings

In our experiment settings, we introduced six models as baselines. These methods includes fixed-order and variable-order Markov models, in literature [24], which were used to evaluate the location prediction from WiFi data. Here, we briefly describe each method.

- **(1) Order-1 Markov Model**. In order-1 Markov model, one transition matrix is maintained, which contains the probabilities of transitions from one location to another. The probability is obtained by counting the appearances of the next location over all appearances of the current location. For a predic-

tion, the next location to be predicted is the one that has the highest transition probability;

- **(2) Order-2 Markov Model**. Order-2 Markov model works in the same way as order-1 Markov model. However, the location dependency is not only based on the current location but also considers one more previous location. Therefore, the predicted is the one with the highest transition probability dependent on previous and current locations;
- **(3) Decayed order-1 Markov Model**. To put more confidences on the recent learnt location transitions, we applied an exponential decay function on the order-1 Markov model. Transition probability will be adjusted by the following equation,

$$P'_n = P_n e^{-\lambda(n-n')} \tag{4.12}$$

, where P is the transition probability, n' is the current time and n is the last time when the same location transition appeared.

- **(4) Decayed order-2 Markov Model**. Decayed order-2 Markov model works in the same way as order-2 Markov model, but is applied an exponential decay function as well;
- **(5) Lempel-Ziv 78 (LZ78)**. The LZ78 is a variable-order predictor. It parses a sequence by extending one symbol from existing phrases at each step. If the extended phrase is new, this phrase will be added into a trie tree. Finally, a sequence is partitioned into many non-overlapping adjacent phrases. For the prediction, the algorithm will start from the root and traverse the tree according the matched phrases. Upon the completion of traverse, the next node with the maximum counts becomes the predicted location.

- **(6) Prediction by Partial Match (PPM)**. PPM is a variable-order predictor like LZ78. But different to LZ78, PPM requires to set an upper bound k. It maintains all previous occurrences of context at each level of k in a trie tree with associated probability values for each context. The prediction of PPM is similar to LZ78, and can handle the zero frequency problem with escape and exclusion mechanisms especially.
- **(7) Predictability Test**. Since the characteristic of mobility datasets can be totally different, some datasets can be predicted easily and some are difficult. To compare the performance of our model with ideal prediction result, we test the predictability for each dataset. The predictability test is based on order-1 Markov model. Give the current location, if the next location can be found in the historical transition recorded previously, we assume the next location is predictable.

In our experiment, each model was required to predict the ID of next location based on the current location ID or more previous ones. The performances of all models were evaluated in terms of their prediction accuracies, which can be calculated by:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Number\ of\ Total\ Predictions}. \tag{4.13}$$

Additionally, since the decayed Markov model was a parameterized method, we tuned the parameter λ from 0.2 to 1 to compare different prediction performances.

4.3.3 Performance Evaluation

Table. 4.3 and Figure 4.6 show the average location prediction accuracies over 68 users in "Life Map" dataset, and the best prediction accuracies of decayed $O(1)$

Table 4.3. Comparison of Prediction Accuracies on “Life Map”

Models	Total Predictions	Correct Predictions	Accuracy
Predictability Test	35425	15445	0.436
Pattern Network	35425	12147	0.343
LZ78	35425	10394	0.278
O(1) Markov	35425	9675	0.273
PPM	35425	9962	0.262
Decayed O(1) Markov	35425	8670	0.245
O(2) Markov	35425	7093	0.200
Decayed O(2) Markov	35425	6214	0.175

Table 4.4. Comparison of Prediction Accuracies on “Reality Mining”

Models	Total Predictions	Correct Predictions	Accuracy
Predictability Test	2901748	2588359	0.892
Pattern Network	2901748	1714570	0.600
O(1) Markov	2901748	1488166	0.524
PPM	2901748	1486886	0.510
O(2) Markov	2901748	1459303	0.503
LZ78	2901748	1346764	0.474
Decayed O(1) Markov	2901748	1323745	0.456
Decayed O(2) Markov	2901748	1307005	0.450

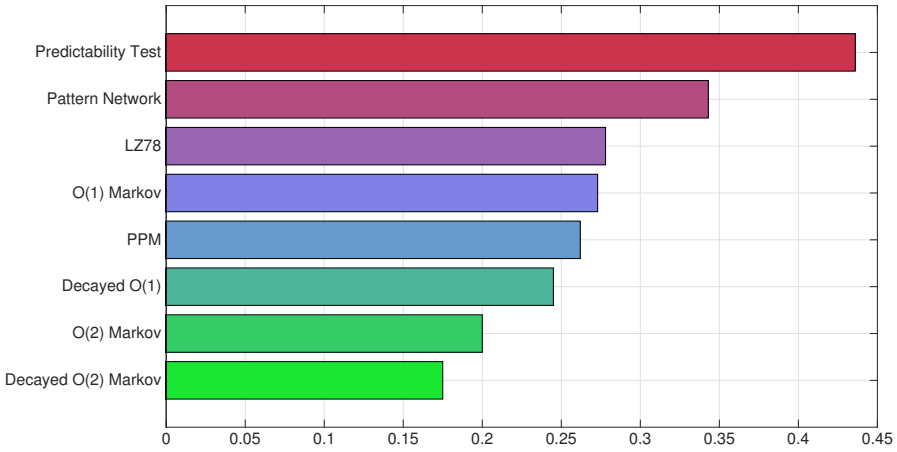


Figure 4.6: Comparison of Prediction Accuracies on “Life Map”

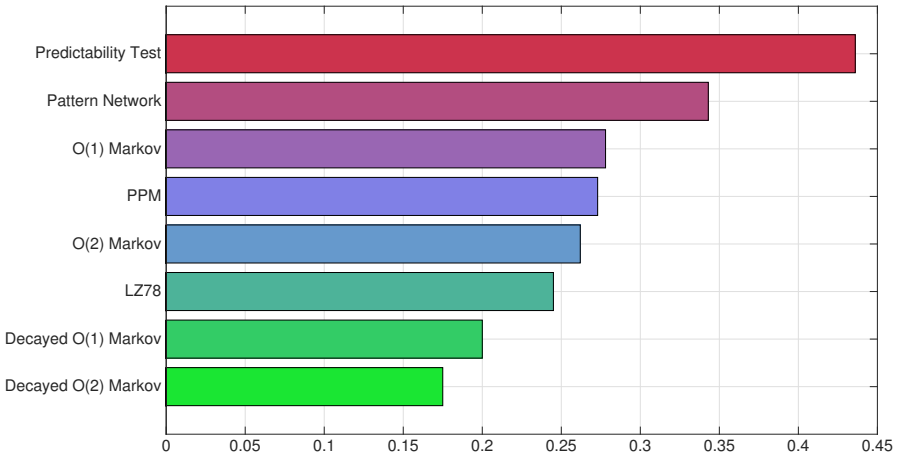
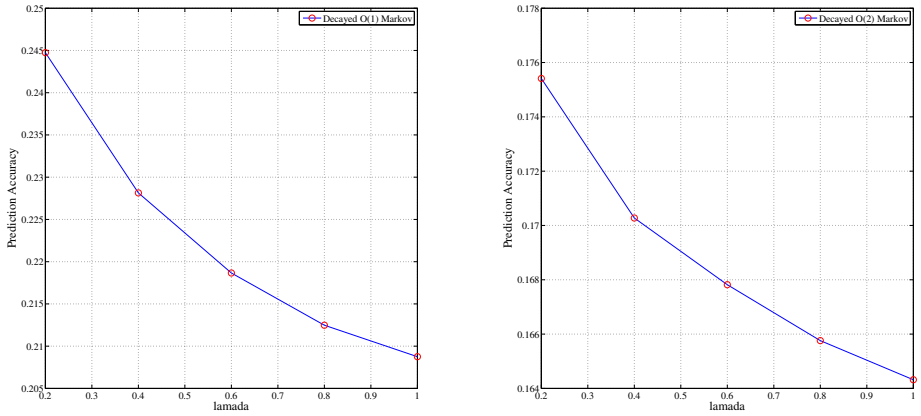


Figure 4.7: Comparison of Prediction Accuracies on "Reality Mining"

and $O(2)$ Markov models are listed in the table. By comparing the accuracies over seven models, we can find our pattern network outperformed other Markov models by 7% on average. However, the absolute prediction accuracies of seven models are not very high. Only around 20%-30% locations could be predicted successfully. When referring to the result of predictability test listed the table, we can find the "Life Map" dataset is hard to be predicted. Even the ideal result is only 43.6% accuracies. It is possible that people's movements changed frequently or many locations were only visited once, which increased the difficulty for prediction. Table. 4.4 and Figure 4.7 show the comparison of prediction accuracies on the dataset "Reality Mining". In general, our pattern network model is still better than others by 8%. Note that the absolute prediction accuracies are higher. The reason is that this dataset contains more repetitive sequences in trajectory, especially more tandem sequences such as $S =< 1 >< 2 >< 1 >< 2 >< 1 >< 2 >$. Since our pattern network model is able to detect tandem repeats, these tandem units can be used for better prediction.

Surprisingly, when comparing order-1 Markov model with order-2 Markov model in two datasets, we can observe that the order-1 Markov model is slightly better than the order-2 Markov model. It is possible that the predicted location from order-2 model is also the most frequent location recorded by order-1 model. Oppositely, the most frequent location in order-1 model may not be the frequent location in order-2 model. The reason is that the condition of order-2 dependency narrows down the searching range. Possible predicted location could be missed and incur a wrong prediction. Our pattern network model works in a variable-order fashion. The prediction of next location is based on multiple active patterns with various pattern lengths. Most frequent next location in order-1 Markov model can be found in the pattern network model. Meanwhile, some long patterns in the extreme case can also be captured. It overcomes the limitations of order-1 and order-2 model. Furthermore, although we introduced two variable-order models, LZ78 and PPM, their performances are not as good as we expect. Overall, for these benchmark models, no one is able to guarantee a good performance across different datasets.

Additionally, we found the performance of Markov models without decay function were better than the performance of decayed Markov model. According to the Figure 4.8, we can observe the prediction accuracies were decreased when increasing the value of λ from 0.2 - 1.0. The disadvantage of applying decay function is that for a location transition that appeared long time ago, its probability could be reduced to a very small value. Even the original probability is high enough, it will be decreased quickly so that the information historical transitions could be lost. While, our pattern network model realizes the weight adjustment by propagating weights to the neighbors of pattern, which are highly possible to appear in the near future.



(a) Decayed O(1) Markov Model (b) Decayed O(2) Markov Model

Figure 4.8: Prediction Accuracies of Decayed Markov Models on "Life Map"

Therefore, the strategy of weight propagation can not only make model adapt to the recent mobility, but also remains all historical transitions.

4.3.4 Pattern Network Analysis

In addition to the testing of prediction accuracy, we can also obtained a pattern network structure after model training. We are interested to know how large a pattern network can be generated for every user, and what is the proportion of different pattern relations. Here, we especially take the dataset "Life Map" as an example to investigate.

Table 4.5. Statistical Analysis of a Node in the Pattern Network (Avg.)

Num. of Nodes in Network	**Frequency**	**Pattern Weight**
159.54	5.34	24.80
Num. of Neighbor Nodes	**Num. of Relations**	**Num. of Node Shifts**
10.07	10.83	2.41

Table. 4.5 lists the statistical results of a node in the pattern network. For each user, he or she could have a pattern network that contains 159.54 nodes on average. For each node in a network, it appeared 5.34 times, and the average pattern weight is 24.80 after weight propagation. Furthermore, each node has around 10 connected neighbor nodes and the average number of relations is 10.83. We can find the number of relations is slightly larger than the number of connected nodes. The reason is that for the same neighbor node, it can be connected by different relations. For example, a pattern could be forward adjacency to another pattern as well as it can be contained in that pattern. Different instances of a pattern may generate different relations with respects to its neighbors. Finally, we can find the number of node shifts from one pattern to another is 2.41 on average.

Table 4.6. Statistical Analysis of an Edge in the Pattern Network (Avg.)

Num. of Edges in Network	**Frequency**	**Connection Strength**
1727.46	2.41	0.27
Num. of Contained	**Num. of Forward Adj.**	**Num. of Forward Overlap**
20.30%	51.27%	28.43%

Table. 4.6 shows the statistical results of an edge in the pattern network. On average, a pattern network includes total 1727.46 edges. Each edge, i.e. a pattern shift, appears 2.41 times, and the connection strength is 0.27. Among all relations, "Forward Adjacency" takes the largest proportion. It is possible that many patterns only contain 1-2 locations. Thus, the adjacency would be the most common relation of two short patterns. Also, we can find the relation "Contained" and "Overlap" have nearly 50% chances to be occurred. It proves that people's mobility patterns are

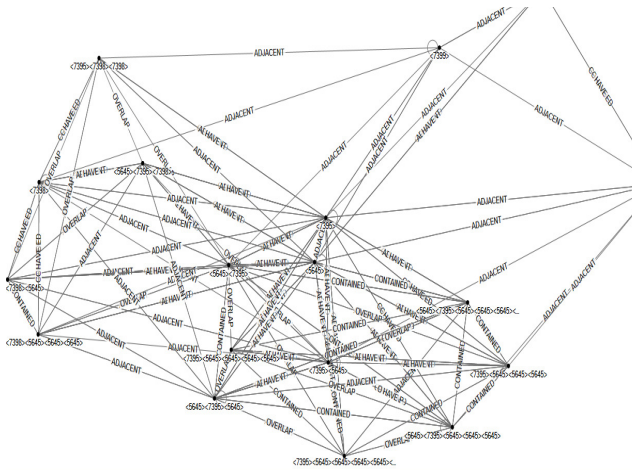


Figure 4.9: A Pattern Network of a Real Smartphone User

usually nested each other, which reflects the complexity of a moving trajectory. An example of a real user's pattern network is demonstrated in Figure 4.9.

4.4 Conclusions

High performance location predictor can facilitate many location-based services to recommend useful contents to smartphone users. In this chapter, we proposed a novel location predictor named pattern network. Pattern network is constructed by connecting repeating sub-sequences in a location sequence. The advantages are that (1) the location dependency are variable according to the different lengths of patterns; (2) Since each pattern is atomic and non-trivial, the pattern may implicit a meaningful path; (3) Besides memorizing location transition, the pattern network keeps information of pattern transition in higher level. This unique insight can not be identified in other methods, because Markov and LZ models only partition a sequence in location level.

When a location prediction is required, a set of candidate locations can be retrieved from the pattern network. The predicted location is selected by a majority voting among the candidates. Meanwhile, a location uncertainty value can be generated, which reflects the uncertainty of prediction at each step. By integrating the location uncertainty into a weight propagation procedure, our model can be trained to have better adaptability.

Our experiments show the pattern network model outperformed other fixed order or variable order models. Especially, the pattern network structure can help us to analyze a user's mobility behaviors. However, through our experiments, we also find there are still a large portion of locations that can not be well predicted, because they are so called zero frequency. Therefore, a location predictor should also consider the prediction of new or unusual locations.

Chapter: 5 Predicting New and Unusual Mobility Behavior

In many occasions, people may deviate from their normal mobility patterns from time to time, e.g.traveling to new places, going to hospital. In such circumstances, if a predictor still treats a high frequent visiting location as a person's next movement, obviously this prediction will be wrong. There are three main factors that make traditional predictors hard to support unusual mobility behavior prediction.

- Unusual mobility information is not contained in traditional predictors. Predictors only maintain frequent location transitions or patterns. They do not have additional records on exceptional cases, which makes the unusual mobility prediction difficult.
- Traditional predictors usually focus on how frequent a pattern happens, but they do not consider how regular a pattern is. Some patterns may occur many time, but also they are easy to be broken. For example, there is little chance for a office worker to change his or her routine on the way to company in the morning. If such a routine is detected, traditional predictors should work well, because a morning commute pattern has higher regularity. However,

office workers may have a habit for shopping at weekend, and will have great chances to go for traveling as well. In this scenario, prediction based on frequency may not be accurate.

- Some works [35] [36] [37] partition trajectory into multiple line segments. Based on special distance calculation, they group similar trajectories together and the outlier line segments are considered as unusual mobility behavior, because these segments are rare and sparse. Nevertheless, these methods do not consider any mobility pattern and the frequency of the pattern appearance. Outliers in space may not be caused from the breaking of patterns. Another method proposed in [6] uses hidden Markov model to train a discretized location sequence. By detecting a significant parameter change in HMM training, it recognizes this trajectory as an unusual behavior. However, these methods need to train a model by scanning historical data many times, which may not be suitable for online training and detection. Moreover, the method proposed in [7] treats a location of wrong predictions as a mobility change. But, this method is hard to be evaluated and interpreted. There is no solid definition of unusual mobility in traditional work.

In order to support unusual mobility prediction, we need to develop a novel predictor to conquer these limitations.

5.1 Definitions of Unusual Mobility Behavior

A good definition is important for our understanding of unusual mobility behavior as well as performance evaluation later. Firstly, in this section, we will give a formal definition about what is the unusual mobility behavior. Consider a user's moving

trajectory that consists of a series of time-ordered triples,

$$S_{1:N} = < \eta_1, t_1, F_1 >, \ldots, < \eta_N, t_N, F_N > \tag{5.1}$$

where η represents the name (ID) of a physical location, t is the timestamp and F is the meta feature. In this location sequence, various sequential repeats can be discovered as described in Chapter 4. These mobility patterns describe people's mobility behaviors including the daily routines of work, shopping, entertainment, etc. We have introduced many definitions of mobility patterns and online discovery methods in Chapter 3 or 4. Intuitively, if a pattern is matched at the beginning, but failed to be followed afterwards, it may indicate an unusual mobility behavior happening. When the next location is different from the earliest on-going pattern, we define it as a Point of Change.

DEFINITION 11 (Point of Change) *The next location is a Point of Change (POC) if it no longer matches the earliest on-going pattern.*

By this definition, a POC can be regarded as a place that leads to a pattern break. The question is why choose the earliest on-going pattern? As we have discussed in previous chapters, people's mobility patterns are usually overlapped or contained. A hidden motivation can be composed of many successive sub-motivations. The earliest on-going pattern represents the initial motivation in a person's mind. This motivation can be lasted for a long time, and may contain other patterns concurrently. If the earliest pattern is broken, we assume a user changed his mind from his or her initial motivation.

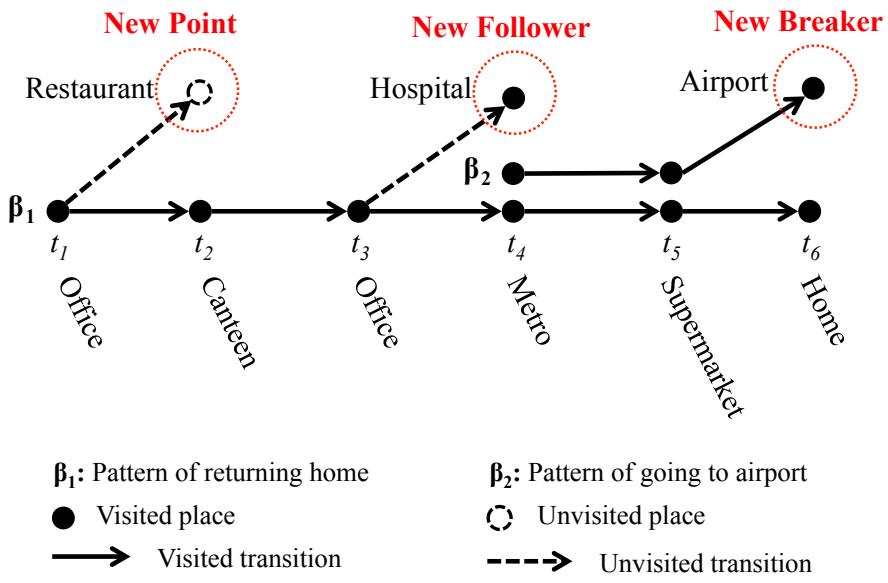


Figure 5.1: Three Types of Point of Change

Additionally, it is also interesting to find out what causes the pattern break. Is the next location a new place, or a visited place? Based on historical location sequence $S_{1:N}$, if the next location η_{N+1} never appears before, η_{N+1} is a new place. In another case, η_{N+1} may be a visited place but arrived via a new transition. Specifically, we can further differentiate POCs into three types, as follows:

DEFINITION 12 (New Point) *η_{N+1} is a new point, if η_{N+1} never appears in historical location sequence $S_{1:N}$.*

DEFINITION 13 (New Follower) *η_{N+1} is a new follower, if η_{N+1} is a visited place but the transition from previous location η_N is new.*

DEFINITION 14 (New Breaker) *η_{N+1} is a new breaker, if η_{N+1} is a visited place and the transition from previous location η_N is happened before.*

Figure 5.1 illustrates three cases of Point of Change.

- Case (1): Tom is at "*Office*" at time t_1, an on-going pattern β_1 is matched, because a prefix of β_1, "*Office*", is the same as current location. It implies Tom may follow this pattern to return home as usual. However, at lunch time, if Tom's next destination is an unvisited "*Restaurant*" instead of "*Canteen*", this restaurant breaks Tom's usual pattern. The unvisited restaurant is a new point.
- Case (2): If Tom goes to "*Hospital*" from office for the first time, it implies an urgent issue makes Tom break his evening commute. The "*Hospital*", which is not the first visit, becomes a new follower at time t_4.
- Case (3): It is possible that multiple patterns are matched concurrently as shown at time $t_4 - t_5$. If Tom follows a pattern β_2 to the "*Airport*", it breaks his earliest on-going pattern β_1. Due to the pattern β_2 occurred before, the "*Airport*" at time t_6 is a new breaker. It is worth noting that a POC only breaks the earliest on-going pattern. If β_2 starts from "*Office*" at time t_1, "*Airport*" is not a POC.

The advantages of the definition of POC are that (1) POC is easy to be understood. It fulfills people's normal cognition of pattern break and unusual mobility behavior. (2) POC is easy to be detected. By finding all repeats in historical location sequence, we are able to detect POCs based on patterns. (3) POC allows different location predictors to be evaluated in terms of a common ground truth. Because the definition of POC is objective and it is not derived from the prediction result of any predictor, a location sequence has only one unique POC label sequence to indicate the status of each location at each time. According to our definitions, we can find

new point, new follower and new breaker are mutually exclusive. By defining three types of POC, we are able to apply different location-based services in different situations.

In our mobility prediction scenario, we focus on building a user-specific model that has the capability to answer the following question:

PROBLEM 2 (POC Prediction) *Given a historical location sequence $S_{1:N}$ from time t_1 to t_N, predict whether the next location η_{N+1} is a Point of Change (POC). If it is, what is the type of POC, new point, new follower or new breaker?*

Table 5.1 lists the key notations used in this chapter. In the rest of this chapter, we will present our solutions to the above problem.

Table 5.1. Notations Used in Chapter 5

Notation	Description
$S_{1:N}$	Location sequence from time t_1 to time t_N
η_N	N-th location in $S_{1:N}$
t_N	Arrival timestamp on η_N
τ_N	Stay duration on η_N
F_N	Meta feature vector on η_N
β	Mobility pattern
γ	Recent traveling path
M	ST-Pattern Network Model
$\mathcal{B}$	Mobility pattern set
G	Pattern network
ω	Pattern weight
θ	Connection strength

5.2 Regularity of Mobility Pattern

The regularity of a pattern is very important for us to infer how many possibilities a user will change his or her mobility under the current circumstance. It is found in previous studies [50, 51] that there is a strong transition regularity in people's mobility behaviors. This suggests that if the current traveling path follows some strong regular patterns, the chance of POC occurrence in the next move may be small. Thus, our strategy to predict POC is to match mobility patterns as many as possible from historical location sequence, and evaluate their regularities. In this section, we will introduce how to calculate the regularity of a mobility pattern.

5.2.1 Mobility Patterns for POC Detection

The prediction of POC is to utilize mobility patterns to estimate the changeability of a person's movement. Studies [50, 51] suggests that spatial patterns are more robust than temporal patterns. Thus, the first step is to look for any spatial pattern from a location sequence, and then verify its regularity based on temporal information.

In Chapter 3, we have introduced different mobility patterns as well as their online discovery methods. Different mobility patterns can be used to fulfill different requirements of applications. In Chapter 4, mobility patterns are found as *semi-tandem near-supermaximal repeat*, *tandem unit* and *pattern candidate*. In order to improve prediction accuracy, we need to portrait a user's movement by every possible cases and keep sufficient trajectory information. However, requirements for POC prediction are slightly different. Here, we make a comparison between two applications.

- We do not need to consider the sub-sequence that occurs only once. Because every location is new, there is no pattern break inside this sub-sequence. Therefore, pattern candidate is not required in POC prediction.
- Individual tandem unit is not necessary. In POC prediction, we are only interested in the break of the earliest mobility pattern. A tandem unit must be contained in other longer patterns. If a long pattern is broken, its tandem unit will be broken as well. Therefore, tandem unit is redundant in this case.
- We hope the conditions of mobility pattern should be as simple as possible so that POC can be detected easily.

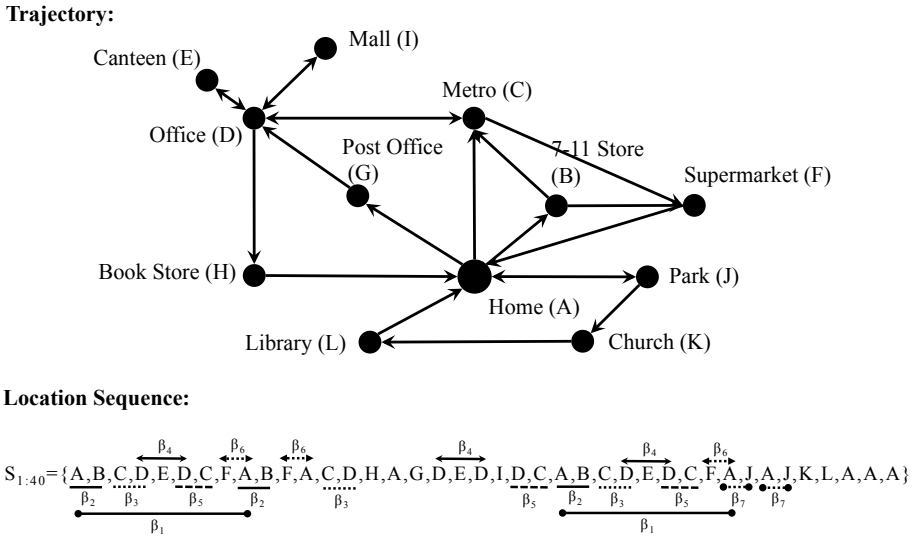


Figure 5.2: Mobility Patterns

Considering above reasons, we only choose the *near-supermaximal repeat* as mobility pattern in the scenario of POC prediction. Definition of POC and our POC prediction model will be implemented based on the near-supermaximal repeat.

Figure 5.2 shows one individual's mobility trajectory for several days. For example, the routine for work β_1 consists of multiple near-supermaximal repeats including

$$\beta_2 = (\textit{Home} \rightarrow \textit{7-11 Store});$$

$$\beta_3 = (\textit{Metro} \rightarrow \textit{Office});$$

$$\beta_4 = (\textit{Office} \rightarrow \textit{Canteen} \rightarrow \textit{Office});$$

$$\beta_5 = (\textit{Office} \rightarrow \textit{Metro});$$

$$\beta_6 = (\textit{Supermarket} \rightarrow \textit{Home}).$$

Note that the above mobility patterns $\beta_2 - \beta_6$ are not only contained in β_1, but also have other independent instances. It is worth noting some mobility patterns consist of a single location such as $\beta_8, \beta_9, \beta_{10}$. In the next sub-sections, we will introduce how to evaluate the regularity of mobility pattern from spatial and temporal aspects.

5.2.2 Spatial Likelihood

As demonstrated in Figure 5.2, a location sequence can be composed by many nested mobility patterns. However, when a user is traveling from the current location η_N to the next η_{N+1}, how can a system know which pattern a user is intended to follow? To answer this question, we compute the spatial and temporal similarity between recent traveling path and previous mobility patterns to infer its likelihood. Here, we give the definition of recent traveling path γ.

DEFINITION 15 (Recent Traveling Path) *A recent location sequence* $[\eta_i, \ldots, \eta_N]$ *is called a recent traveling path, if it matches the prefix of a mobility pattern.*

At location η_N, it may have multiple mobility patterns that are matched concurrently. These recent traveling paths need to be compared with their corresponding matched patterns respectively. From the spatial aspect, the likelihood that a particular pattern β is being followed can be estimated based on its matched path γ. Since the sequential order of a pattern is fixed, the spatial likelihood can be simply computed by the length of recent path:

$$P_s(\beta|\gamma) = \frac{|\gamma|}{|\beta|}. \tag{5.2}$$

If $P_s(\beta|\gamma) = 1$, it means that an instance of pattern β is fully observed, and thus we are ensure the user is following a mobility pattern β.

5.2.3 Temporal Likelihood

Temporal similarity is another aspect we can use to infer the likelihood of an ongoing pattern. The time a user arrives at a location and the period he stays at that location are two basic temporal features of his/her mobility behavior. In a sequential pattern, a stay duration may be dependent on its previous locations. Also, start time of a pattern may vary from time to time. For example, Tom arrives office at 10AM this morning rather than 9AM as usual, but his one-hour lunch time is fixed from 12AM to 1PM at canteen. To guarantee 8 hours working, his stay duration at office in the afternoon will be 6 hours instead of 5 hours. Therefore, it is necessary to set up a temporal model for each mobility pattern so that temporal likelihood can be calculated.

(1) Stay Duration

To model the time dependence, we derive stay duration τ of a location from its arrival and departure timestamps. Additionally, in-transit duration between two consecutive locations can be obtained as well. For a fixed length mobility pattern β, durations can be represented by a $2|\beta|-1$ dimensional random duration vector, $\tau_\beta = [\tau_1, \ldots, \tau_{2|\beta|-1}]$, where $|\beta|$ is the length of the pattern β. Stay durations and in-transit durations in a vector are interleaved with the same sequential order of mobility pattern.

We believe if a pattern is highly regular on time, the stay and in-transit duration should be consistent each time. Therefore, the duration vector $\boldsymbol{\tau}_\beta$ is modeled by a multivariate Gaussian distribution, $P(\boldsymbol{\tau}_\beta) \sim N_{2|\beta|-1}(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta)$:

$$P(\boldsymbol{\tau}_\beta) = \frac{1}{\sqrt{(2\pi)^{2|\beta|-1}|\boldsymbol{\Sigma}_\beta|}} \exp\left(-\frac{1}{2}(\boldsymbol{\tau}_\beta - \boldsymbol{\mu}_\beta)^\top {\boldsymbol{\Sigma}_\beta}^{-1}(\boldsymbol{\tau}_\beta - \boldsymbol{\mu}_\beta)\right). \tag{5.3}$$

Eq. 5.3 is the probability density function for the multivariate Gaussian distribution of duration vector. Parameter $\boldsymbol{\mu}_\beta$ is a mean vector of variables $\boldsymbol{\tau}$, where $\boldsymbol{\mu}_\beta = [E(\boldsymbol{\tau}_1), \ldots, E(\boldsymbol{\tau}_{2|\beta|-1})]^\top$. Parameter $\boldsymbol{\Sigma}_\beta$ is a symmetric covariation matrix in which the entry in the i-th row and j-th column expresses the covariation between random variable $\boldsymbol{\tau}_i$ and $\boldsymbol{\tau}_j$, i.e. $\boldsymbol{\Sigma}_\beta = Cov(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j), i, j = 1 \ldots 2|\beta| - 1$. Once an instance of β is identified, the mean vector $\boldsymbol{\mu}_\beta$ and covariation matrix $\boldsymbol{\Sigma}_\beta$ need to be updated correspondingly.

Given the duration vector $\boldsymbol{\tau}_\gamma$ of a recent path γ, temporal likelihood of on-going pattern β can be computed according to the multivariate Gaussian model. However, the recent path γ may only match with β partially, durations beyond matched part are empty values. Therefore, we only need to consider a $2|\gamma| - 1$ length duration vector instead of $2|\beta| - 1$, where $|\gamma|$ is the length of recent path and $|\gamma| \leq |\beta|$. To calculate $P(\boldsymbol{\tau}_\gamma)$, the original multivariate Gaussian distribution needs to be marginalized out over these empty variables $\tau_{2|\gamma|}, \ldots, \tau_{2|\beta|-1}$. The calculation of marginal probability

is illustrated in follows,

$$P(\boldsymbol{\tau}_\gamma) = \int p(\tau_1, \ldots, \tau_{2|\gamma|-1}) d\tau_{2|\gamma|} \ldots d\tau_{2|\gamma|-1} = \\ \frac{1}{\sqrt{(2\pi)^{2|\gamma|-1}|\hat{\boldsymbol{\Sigma}}_\beta|}} \exp\left(-\frac{1}{2}(\boldsymbol{\tau}_\gamma - \hat{\boldsymbol{\mu}}_\beta)^\top \hat{\boldsymbol{\Sigma}}_\beta^{-1}(\boldsymbol{\tau}_\gamma - \hat{\boldsymbol{\mu}}_\beta)\right). \tag{5.4}$$

where $\hat{\boldsymbol{\mu}}_\beta = [\mu_1, \ldots, \mu_{2|\gamma|-1}]^\top$ is the marginalized mean vector of β, and $\hat{\boldsymbol{\Sigma}}_{\beta(i,j)} = Cov(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j), i, j = 1 \ldots 2|\gamma| - 1$ is the marginalized covariation matrix of β. Eq. 5.4 shows the marginal distribution of a multivariate Gaussian is also a Gaussian distribution. We only need to re-compute the trained mean vector and covariation matrix of pattern β by the length of $2|\gamma| - 1$.

(2) Start Time

If a pattern is regular, does it start at a fixed time? To better infer the temporal likelihood, we choose the arrival time of the first location as a pattern's start time. The random variable t_γ is used to represent the start time of γ. However, a single Gaussian distribution may not be sufficient to characterize the distribution of arrival time in 24 hours. For example, location "*Metro*" may be visited twice during morning and evening commute in a working day. Therefore, we use a histogram to model possible multiple peaks of the visiting at different times per day. The frequency density is calculated by Eq. 5.5,

$$P(t_\gamma) = \frac{histo(t_\gamma)}{TotalHisto} \tag{5.5}$$

where $histo(t_\gamma)$ is the frequency of bin that t_γ belongs to, and the time interval is set to 30 minutes in our scenario. Due to the sparsity of pattern instances, some

bins may be empty. To estimate the frequency density of empty bin, we make a linear interpolation from two nearest valued bins, which are earlier and later than bin $histo(t_\beta)$ respectively.

As we assume the arrival time is independent on stay durations, the overall temporal likelihood of start time and stay duration can be computed as the product of frequency density $P(t_\gamma)$ and probability density $P(\tau_\gamma)$:

$$P_t(\beta|\gamma) = P(t_\gamma) \cdot P(\tau_\gamma). \tag{5.6}$$

Finally, to verified whether a mobility pattern is on-going can be computed by its spatial and temporal likelihood with the recent traveling path. We can also use the likelihood value to compute the regularity of a pattern. If the likelihood of every pattern instance is high, this pattern has strong regularity.

5.3 The POC Prediction Model

In Chapter 4, we have proposed a novel model called pattern network to predict next location. In this section, we extend the pattern network to learn the historical occurrences of POCs, which is called a ST-Pattern Network. There are three major improvements:

- Temporal information is integrated into ST-Pattern Network;
- Pattern weight is calculated based on the regularity of pattern.
- ST-Pattern Network is able to support multi-class prediction.

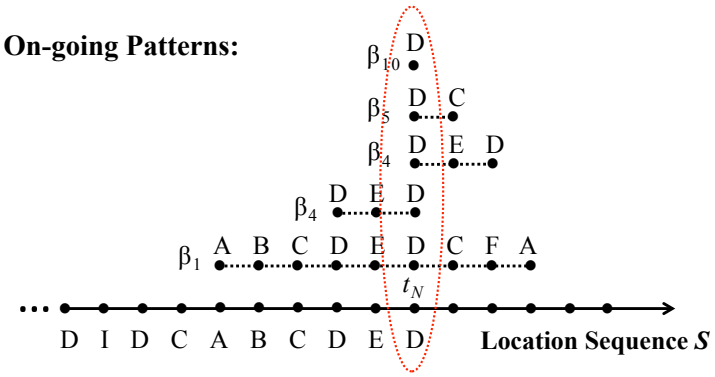
Next, we will describe our method for the training ST-Pattern Network. The spatial and temporal likelihood introduced above will be used to compute the chance of POC occurrence and the pattern weight on pattern network.

5.3.1 POC Training

An observation of mobility change is that under a strong regularity pattern, the chance of POC occurrence will be small. For example, morning commute "*Home*" $\rightarrow$ "*Office*" in a working day is less likely to be broken. On the contrary, routine "*Office*" $\rightarrow$ "*Home*" may be likely to change since people may go to pub or party after work. Therefore, the idea of model training is to learn under what kinds of mobility patterns POCs are most likely to happen. Our method is to enumerate different mobility patterns, and train related on-going patterns in terms of historical POC occurrences.

(1) Model Training

Every time when a user is moving, his/her trajectory will be updated incrementally. How can we train a model efficiently and deploy an online prediction? Let us look at an example in Figure 5.3. Figure 5.3 has the same trajectory as in Figure 5.2, and assumes a user is at location "*Office(D)*" at time t_N. Hence, prefixes of patterns β_1, β_4, β_5 and β_{10} are matched with recent paths. These on-going patterns are only a part of patterns that are activated during user's movement, which are denoted as β_{on}. Therefore, at each location we only need focus on these on-going patterns. If a POC happens, we mark the corresponding positions on on-going patterns.



POC matrix $C_{\beta 1}$:

	A	B	C	D	E	D	C	F	A
New Point	2.5	1.6	0	0	0	0	0	0	0
New Follower	1.2	2.8	0.8	0	0	0	0	0	0
New Breaker	3.6	3.2	2.3	1.2	0	0	0.5	0	0
Non-POC	10.2	8.3	8.2	6.5	6.2	5.9	5.8	5.3	4.8

Figure 5.3: POC Matrix

For each mobility pattern, it maintains a POC matrix C to record the possible positions of POC occurrences. Figure 5.3 shows the structure of POC matrix C_{β_1} for the pattern β_1. Each row represents a pattern sequence and each column contains four 4 types including new point, new follower, new breaker and not-POC. An entry of a matrix indicates whether the next location is the POC or not. For example, if a new follower happens after location 'A', a value will be credited on the entry $C_{\beta_1}[\textit{New Follower}, A]$.

Naturally, an entry can be filled by a counting value. However, on-going patterns may be partially matched as shown in Figure 5.3. We can not exactly tell which pattern a user is intended to follow based on observed recent paths. Therefore, our strategy is to add a spatial-temporal likelihood value on the corresponding entry of POC matrix.

Given the start time t_γ, duration vector τ_γ and matched length $|\gamma|$ of a recent path γ, the spatial-temporal likelihood can be computed as follows:

$$P(\beta_{on}|\gamma) = P_s(\beta_{on}|\gamma) \cdot P_t(\beta_{on}|\gamma). \tag{5.7}$$

We assume the spatial and temporal likelihoods are independent. The value of entry is accumulated as Eq. 5.8, where $c \in \{New\ point, New\ follower, New\ breaker, not\text{-}POC\}$:

$$C_{\beta_{on}}[c, |\gamma|] = \sum P(\beta_{on}|\gamma). \tag{5.8}$$

Thus, we can train POC matrices of all patterns from the historical location sequence, which will be used for future POC prediction.

(2) Pattern Weight Calculation

Furthermore, we can check the regularity of a mobility pattern. A regular pattern should have a large weight value on POC prediction, because the likelihood of its POC matrix is more reliable. Here, we measure a pattern's weight ω by its spatial-temporal likelihood:

$$\omega_\beta = \omega_\beta + \sum P_t(\beta_{on}|\gamma). \tag{5.9}$$

Pattern weight is accumulated only when its instance is completely observed. Thus, we can ensure a pattern is happening, and its regularity can be updated by its new spatial-temporal likelihood. Note that for a pattern being fully observed, the spa-

tial similarity is 1. The algorithm of online POC learning is summarized in Algorithm 5.1.

Algorithm 5.1 Online POC Training

Input: Location: η_i, POC type: c_{i+1}, Pattern set: β
Output: Updated β

1: $(\gamma, \beta_{on}) \leftarrow$ **FindRecentPaths**(η_i, β)
2: **for** each $\gamma \in \gamma$ **do**
3: $P(\beta_{on}|\gamma) \leftarrow P_s(\beta_{on}|\gamma) \cdot P_t(\beta_{on}|\gamma)$
4: $C_{\beta_{on}} \leftarrow$ **GetPOCMatrix**(β_{on})
5: $\omega_{\beta_{on}} \leftarrow$ **GetPatternWeight**(β_{on})
6: $C_{\beta_{on}}[c_{i+1}, |\gamma] += P(\beta_{on}|\gamma)$
7: **if** $|\beta_{on}| == |\gamma|$ **then**
8: $\omega_{\beta_{on}} += P(\beta_{on}|\gamma)$
9: **end if**
10: $\beta \leftarrow$ **UpdatePatternWeightandPOCMatrix**$(\beta_{on}, C_{\beta_{on}}, \omega_{\beta_{on}})$
11: **end for**
12: **return** β

(3) Weight Propagation

The idea of weight propagation is based on the assumption that if a pattern happened recently, other connected patterns may have more chances to happen afterwards. Similarly, in POC prediction, we hope the POC matrices of possible next patterns have more impacts on the future prediction. The weight propagation is also implemented in ST-Pattern Network model.

Figure 5.4 illustrates a ST-Pattern Network G, which records partial pattern connections ($\beta_1 - \beta_7$) from $S_{1:40}$ in Figure 5.2. The network is constructed incrementally according to the historical occurrences of two patterns. After that, once a mobility pattern is fully observed again, the pattern's weight will be increased by the current spatial-temporal similarity. Meanwhile, this spatial-temporal similarity need to be propagate to pattern's neighbors as well as neighbor's neighbors until the variation

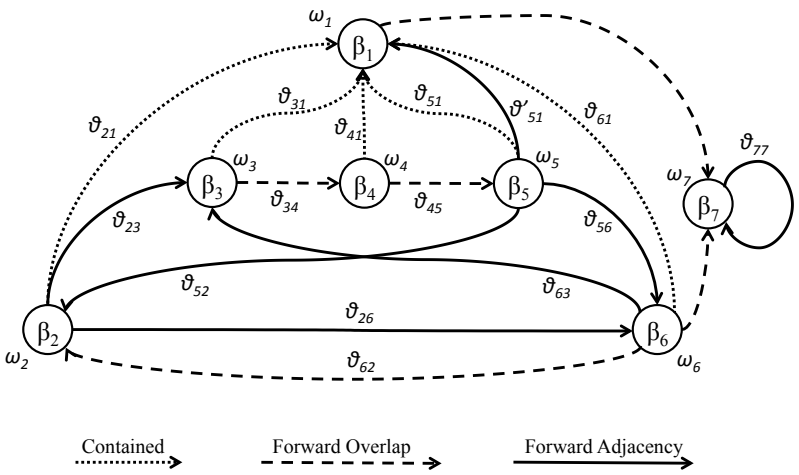


Figure 5.4: A ST-Pattern Network

is vanished. Same as in Chapter 4 we use the connection strength θ to control the proportion of propagation at each round. After weight propagation, the ST-Pattern Network is ready for POC prediction next.

5.3.2 POC Prediction

Given a trained ST-Pattern Network model, we want to know whether the next location η_{N+1} will be a Point of Change. If it is, what is the type of this POC? Therefore, we pose a POC prediction as a binary classification problem and the POC type prediction as a multi-class problem.

At time t_N, suppose K recent paths γ_i and corresponding on-going patterns β_{on}^i are found, where $i = 1, \ldots, K$. Each on-going pattern maintains a trained POC matrix $C_{\beta_{on}^i}$. To predict whether the next location is a POC or not, we take the following operations. Suppose the length of a recent path is $|\gamma_i|$. Firstly, we extract a column

vector from the $|\gamma_i|$-th column of POC matrix $C_{\beta_{on}^i}$. There are total K column vectors needed to be extracted from K POC matrices. Secondly, we group these K vectors to form a $4*K$ prediction matrix D. Finally, we transform the prediction matrix D to a $2*K$ matrix D_{binary} by summing the rows of new point, new follower and new breaker. Now prediction matrix D_{binary} can only be used to predict POC and not-POC.

Besides the matrix D_{binary}, we also consider the weights of all on-going patterns and their current spatial-temporal likelihoods. If a recent path is very similar to an on-going pattern and the weight of this pattern is high, this pattern will greatly dominate final POC prediction. Therefore, we put weights of on-going patterns into a K length column weight vector W. Similarly, all current spatial-temporal likelihoods are grouped into a K length column likelihood vector L. Now, we can combine prediction matrix D_{binary}, weight vector W and likelihood vector L to predict POC as follows:

$$V_{binary} = D_{binary} \cdot (W^{\top} \cdot L). \tag{5.10}$$

Finally, POC of the next location can be predicted by finding the class label of entry in V_{binary} that contains the maximum value, as shown in Eq. 5.11, where $c_{binary} \in \{POC, not\text{-}POC\}$:

$$c_{binary} = \arg\max_j (V_{binary}[j]), \quad j = 1 \ldots |V_{binary}|. \tag{5.11}$$

Similarly, to further predict the type of POC, we only need to remove the row of not-POC in prediction matrix D, and obtain a $3 * K$ prediction matrix D_{multi}. Same as Eq. 5.10 and Eq. 5.11, we combine the prediction matrix D_{multi}, weight vector W and likelihood vector L to predict the type of POC (New point, New follower and New breaker). The algorithm of POC prediction is listed in Algorithm 5.2.

Algorithm 5.2 POC Prediction

Input: Location: η_N, Pattern set: β
Output: POC of next location: c_{N+1}

$(\gamma, \beta_{on}) \leftarrow$ **FindRecentPaths**(η_i, β)
for each $\gamma \in \gamma$ **do**
  $C_{\beta_{on}} \leftarrow$ **GetPOCMatrix**(β_{on})
  $\omega_{\beta_{on}} \leftarrow$ **GetPatternWeight**(β_{on})
  $P(\beta_{on}|\gamma) \leftarrow P_s(\beta_{on}|\gamma) \cdot P_t(\beta_{on}|\gamma)$
  $D \leftarrow$ **InsertColumnToMatrix** $C_{\beta_{on}}.column(|\gamma|)$
  $W \leftarrow$ **InsertElementToVector** $\omega_{\beta_{on}}$
  $L \leftarrow$ **InsertElementToVector** $P(\beta_{on}|\gamma)$
end for
$D_{binary} \leftarrow$ **TransformToBinaryPrediction**(D)
$V_{binary} \leftarrow D_{binary} \cdot (W^\top \cdot L)$
$c_{N+1} \leftarrow \arg\max_j(V_{binary}[j])$
if c_{N+1} *is POC* **then**
  $D_{multi} \leftarrow$ **TransformToMulticlassPrediction**(D)
  $V_{multi} \leftarrow D_{multi}(W^\top L)$
  $c_{N+1} \leftarrow \arg\max_j(V_{multi}[j])$
end if
return c_{N+1}

5.4 Experiments

In this section, we will evaluate our ST-Pattern Network model proposed in this chapter. Several traditional Markov models are used as benchmarks to compare POC prediction performances. Additionally, we further investigate the smartphone usages on or before POC location to see whether people's mobility change will affect their smartphone usage behaviors.

5.4.1 Dataset

For our experiments, the real mobile dataset "Life Map" is used to predict the unusual mobility behaviors in a moving trajectory, since this dataset contains timestamps for each visited location. As shown in previous chapters, "Life Map" is a high quality dataset that records all movements of 68 data donators. However, the dataset does not have any information about mobility change and POCs. To label the ground truth of POCs, firstly, we discovered near-supermaximal repeats as patterns from individual sequence. Secondly, according to the definitions Def. 11-Def. 14, if an earliest on-going pattern can not be followed, we labeled the next location as POC, and determined its type, e.g. "*New Point*", "*New Follower*", or "*New Breaker*". For all matched locations inside a pattern, they are labeled as "*not-POC*". It is possible that at the beginning of sequence no near-supermaximal repeat is formed. For those locations that are not covered by any pattern, we labeled them as "*Out-Pattern*", which would not be predicted in our experiments. Features of the dataset are listed in Table 5.2. From Table 5.2, we can find during two-month period each smartphone user visited 522 locations on average. The number of POCs is greater than the number of not-POCs by two times. That means that people's mobility patterns are frequently changed in their daily life. Predicting unusual mobility change is necessary for the improvement of recommendation system.

Table 5.2. POCs Labelled in Dataset

Num. of Samples	**Avg. Seq Length**	**Num. of Locations**
68	522	35493
Num. of POCs	**Num. of not-POCs**	**Num. of Out-Patterns**
15202	8795	11496
Num. of New Points	**Num. of New Followers**	**Num. of New Breakers**
4361	2406	8435

5.4.2 Experiment Settings

To evaluate the performance of our ST-Pattern Network model, we used several popular approaches in mobility prediction as benchmarks. Here, we briefly describe each method.

(1) Pure temporal model. Pure temporal model is based on the assumption that people's mobility behavior will happen at a particular time. For example, people may go out for travel only at weekend. To configure the temporal model, we partition a week into 7*48 time slots with each 30-minute time granularity. For each time slot, we count all categories of locations that happened in this time slot. To predict the future POC, we find the time slot that the arrival time belongs to, and select the category with the maximum count as the possible category for the next location.

(2) 1-order spatial model. 1-order spatial model is a 1-order Markov chain that assumes the property of next location is only dependent on one previous location. For each particular location, we count the category of its next location every time. Thus, in terms of current location, we can predict the category of next location by finding category with the maximum count.

(3) 1-order ST model. 1-order ST model is a mixed model of pure temporal and 1-order spatial model. Training of temporal and spatial models is separated. The prediction is made by a majority voting mechanism. This model considers both space and time of mobility change concurrently.

(4) 2-order spatial model. Similar with 1-order spatial model, we extended 1-order Markov chain to a 2-order fashion. Due to the complex of mobility behavior, statistics on one location may not be sufficient to differentiate various cases. Therefore,

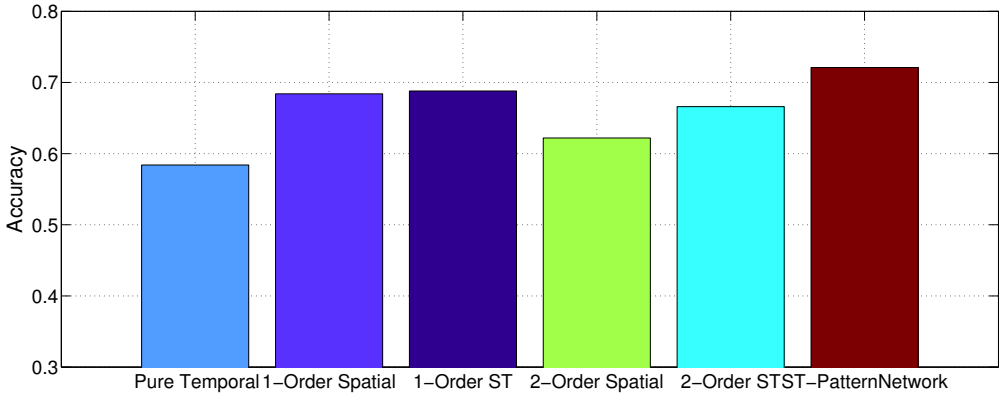


Figure 5.5: Accuracy on Binary POC Prediction

a 2-order Markov chain was deployed to refine a spatial model.

(5) 2-order ST model. Additionally, we also extended a 1-order ST model to a 2-order ST model.

The performances of all models in our experiment were mainly evaluated in terms of accuracy and F-measurement, which can be measured by:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \tag{5.12}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP+FP+FN}, \tag{5.13}$$

where TP is true positive, TN is true negative, FP is false positive and FN is false negative.

5.4.3 POC Prediction Performance

Table 5.3. Comparison of Binary POC Prediction

Model	Precision	Recall	F-measure	Accuracy
Pure Temporal	0.682	0.643	0.662	0.584
1-Order Spatial	0.732	0.791	0.761	0.684
1-Order ST	0.723	0.823	0.770	0.688
2-Order Spatial	0.730	0.640	0.682	0.622
2-Order ST	0.720	0.773	0.745	0.666
ST-Pattern Network	0.739	0.867	**0.798**	**0.721**

When performing POC prediction, we first predict whether the next location is a POC or not-POC. Table 5.3 lists the performances of five benchmark models and our ST-Pattern Network model on a binary POC prediction, including precision, recall, F-measure and Accuracy. Also, accuracies of six models are shown in Figure 5.5. In general, pure temporal performs the worst among total six models. It shows only time information is not sufficient to infer the occurrence of POC, because mobility change may vary across different time slots. Secondly, we found models in 2-order fashion are not better than 1-order models, which decreases 6% prediction accuracy. It is possible that having additional location dependence incurred the overfitness of model. Finally, our ST-Pattern Network model performs better than other benchmark models and has 4%-14% improvement on prediction accuracy. The main reason is that we discovered all possible mobility patterns to train people's mobility change under different cases. For a long pattern, it can be viewed as a high-order Markov chain and guarantees to capture POC occurrence

in some extreme cases. On the other side, deploying on-going patterns as many as possible to learn the POCs can capture more general cases and avoid the overfitness of model in some rare situations, which is the major problem of 2-order model. Therefore, our ST-Pattern Network model balances two issues and yields a better performance.

Table 5.4. Comparison of Multiclass POC Prediction

Model	Precision	Recall	F-measure	Accuracy
Pure Temporal	0.432	0.304	0.343	0.533
1-Order Spatial	0.506	0.359	0.400	0.600
1-Order ST	0.482	0.335	0.380	0.574
2-Order Spatial	0.474	0.351	0.387	0.601
2-Order ST	0.461	0.324	0.366	0.564
ST-PatternNetwork	0.496	0.353	0.388	0.582

If the next location was predicted as a POC, we further predict its type. Table 5.4 lists the average performances on multi-class POC prediction. The overall accuracy is lower than in binary case probably due to the higher complexity of multi-class prediction than binary prediction. To differentiate the type of POC, our model is close to 1-order and 2-order spatial model. It shows the type of POC, new point, new follower and new breaker, does not have strong difference on spatial and temporal aspects in this dataset.

5.4.4 Patterns for POC Analysis

Besides the performance of model, we were also interested to know how the mobility patterns involve in our life. In our experiment, we discovered all patterns in our dataset, and summarized their properties in Table 5.5. In terms of our near-supermaximal repeat definition, total 7847 patterns were found. On average, each user generated 115 patterns over 522 locations in 2 months. The number of mobility

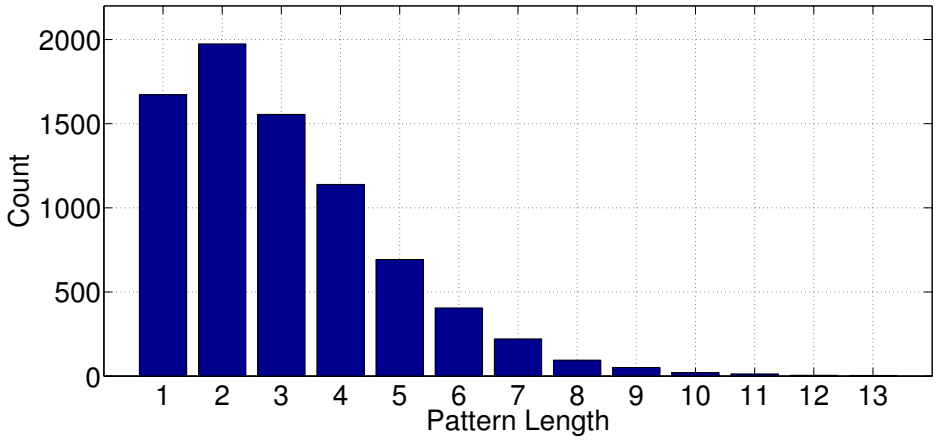


Figure 5.6: Distribution of Patterns with Respect to Pattern Length

Table 5.5. Mobility Patterns in the Dataset

Total Num. of Patterns	**Num. of Patterns per User**	**POCs on Pattern (Avg.)**
7847	115	63
Max. Pattern Length	**Avg. Pattern Length**	**Frequency of Pattern (Avg.)**
13	3	7.5

patterns generated is relatively high compared with the length of location sequence. It proves our trajectory usually consists of many patterns. Among 7847 patterns, the longest pattern contains 13 different locations, and the average pattern length is 3. Figure 5.6 shows the distribution of all patterns with respect to their pattern lengths. Most of our patterns are comprised by 2-3 locations. The number of patterns is dramatically decreased when the pattern length increases. It can be understood that long patterns are rare in our daily life. We may get to a destination through only 2-3 location transitions. Each pattern appears 7.5 times in a 2-month trajectory, which may indicate that most patterns would occur as least once in a week.

How many times a pattern is broken or not followed by next location? Surprisingly, we found each pattern was broken 63 times due to POCs on average. Figure 5.7 displays the POC distribution with respect to the pattern length. We can see 7-length

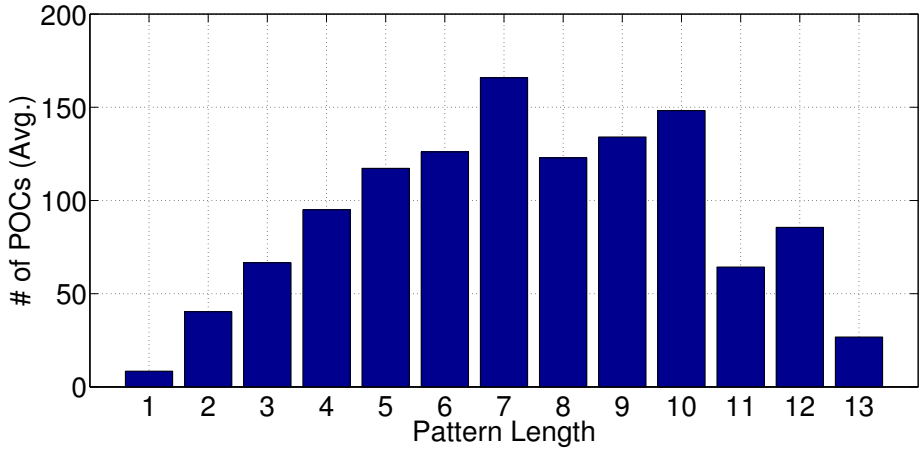


Figure 5.7: Distribution of Occurred POCs with Respect to Pattern Length

pattern has most possibility to be broken. The longer pattern is, the more difficult a pattern can be followed always. For the pattern with more than 7 locations, the number of POCs are decreased. The reason is that a long pattern has less chances to happen as well, which is proved in Figure 5.6.

5.4.5 Smartphone Usage Analysis

Our dataset not only has spatial-temporal information, but also contains smartphone usage records. In our experiment, we further investigate how people's mobility change will affect their smartphone usage behaviors. Real-time smartphone's battery consumptions, screen activities, were recorded completely by deployed mobile apps. We compared the volumes of battery consumption and the screen on and off at POC locations.

Table 5.6 shows three features, i.e. battery consumption, on screen count and on screen duration. The battery consumption was calculated by battery level (0%-100%) decrease over a minute. The on screen duration recorded the time duration

Table 5.6. Smartphone Usages on POC / not-POC locations

Usages	Battery (per min)	On Screen Count	On Screen Duration
New Point	0.148%	0.209	15.84s
New Follower	0.109%	0.166	21.40s
New Breaker	0.105%	0.168	13.99s
not-POC	0.099%	0.153	14.31s

between on and off screen. We found on average the battery consumption increased 21% on POC locations. Similarly, the on screen count and the duration increased 19% and 20% respectively. These features proved that people may be intended to use their smartphones more frequently on unusual situations. Based on these observations, in our future work, we can merge these meta features to improve the prediction of POC.

5.5 Conclusions

Breaking previous mobility patterns and forming new patterns are taking place in people's daily life from time to time. These mobility changes will bring difficulties for traditional location predictors and recommendation systems to estimate a user's next movement and profile personal life mode accurately. If unusual mobility behavior can be predicted, it will provide traditional systems a new way to understand people's real intentions, e.g. asking heuristic question in advance instead of recommending blindly.

In this chapter, we first give a formal definition of mobility change, called Point of Change (POC), to describe people's new and unusual mobility behaviors. POC indicates a location that a user may move to the next location that is different with his or her previous path. Detecting POC is very useful in our real life, because when

a user is in an unfamiliar path, he or she has more chances to use navigation, search Internet by smartphone. If a smartphone system can predict POC, it can pre-load information of map, nearby famous spots or transportation. Moreover, it is possible that in an unfamiliar place, a user may lose Internet connection. Smartphone can pre-load or backup important information in advance.

POCs can be learned by discovering the pattern breaking from all mobility patterns in historical trajectory. Our ST-Pattern Network model can online predict future POCs and adapt to recent trajectory via a pattern based network. Finally, our experiments show the POC prediction accuracies can be significantly improved by our model over traditional Markov models. Additionally, our results indicated people use smartphones more frequently on POC locations. We expect future recommendation system provides more assistances for people once a POC is predicted.

Chapter: 6 Conclusions

Although user's various behaviors can be recorded by smartphone, only moving trajectories can be accessed easily due to the privacy issue. Without additional information, how to mine a user's behaviors and predict his or her next movements are two crucial problems for the location-based service providers. Additionally, considering the huge volume of trajectories generated by mass of users, pattern mining and prediction algorithms should be efficient. In this thesis, we have shown our systematical works to solve above challenges on mobility pattern mining, next location prediction and unusual mobility prediction.

In Chapter 2, we have reviewed many existing works related to the mobility prediction. Firstly, we categorized different presentations of moving trajectory. Normally, real-world coordinate-based sequence and location-based sequence are two popular presentations when we collect trajectories from smartphone or location-based social networks. Several sequential pattern mining algorithms, e.g. GSP and PrefixSpan, have been reviewed, which allow to discover frequent item set from multiple sequential transactions efficiently. However, a user's moving trajectory is usually continuous. Arbitrary partitioning of a trajectory may lead to the loss of potential mobility patterns. While, sequence mining algorithms based on suffix tree or suffix

array are able to discover repeats on a whole sequence. Besides sequence analysis, many other mobility patterns are expected to be detected in different applications such as periodic patterns, spatial-temporal patterns or knowledge-based patterns. Nevertheless, defining and mining these patterns require lots of domain knowledges and feature engineering works. People with different culture background or living in different countries may have completely different habits. Knowledge-based patterns may not work for many general scenarios. For the location prediction, we examined Markov based predictor and Lempel-Ziv predictor. Both of them have their advantages and limitations. Markov based predictor works very efficiently but has strong assumption on the number of dependent previous locations. The order of Markov model is fixed and requires to be specified in advance. While, Lempel-Ziv predictor realizes the location prediction based on variable-length sub-sequences, but any possible pattern across two parsed sub-sequences is lost. Furthermore, prediction of unusual mobility pattern is a new challenge for location-based services. People may only have the demands of recommendations when they arrive a new place or change their regular behaviors. However, only [7] attempted to predict the temporary departures from routine, but they did not provide a formal definition of unusual mobility behavior, which leaves the room for us to achieve unusual mobility prediction.

In Chapter 3, we first proposed a set of definitions for discovering mobility pattern from a location sequence. By comparing the family of maximal repeats, near-supermaximal repeat is selected as an atomic pattern, as we expect a mobility pattern should be non-trivial and longest possible. Furthermore, due to the noise of trajectory or abnormal mobility behavior, a tandem structure may be contained in a near-supermaximal repeat. To differentiate the patterns with different tandem

structure, we classify a mobility pattern into three categories, including full-tandem pattern, semi-tandem pattern and non-tandem pattern. By removing the full-tandem patterns, noisy information can be filtered out and guarantee the quality of patterns. So far, no algorithm can support the discovery of near-supermaximal repeat and the detection of tandem structure concurrently. Our work achieved the online discovery of various patterns via a pipeline method. Every pipeline caches a suffix of a repeating sequence. The near-supermaximal repeat and tandem structure can be detected quickly by comparing the caching positions of pipelines on the suffix tree. Through our experiments, we have verified that the running time and the storage of our algorithm is linear to the length of sequence. Such performance allows our pattern mining algorithm to support a big data scenario.

In Chapter 4, we utilized various discovered patterns as atomic patterns to train our mobility model for the location prediction. As we observed that people's mobility patterns are usually contained or overlapped each other, all patterns can be connected into a pattern network as long as two patterns appear together in history. Our pattern network model works by three steps, prediction, verification and weight propagation. Location prediction can be achieved by selecting the active patterns and looking for a set of candidate locations from the pattern network. For the candidate with the highest confidence, it is treated as the next predicted location. In the verification step, all active patterns that nominated the correct candidate will be rewarded, and the increased pattern weight will be propagate to their neighbors via the network. Through above steps, the pattern network model can adapt to the recent trajectory as well as remain all historical location transition information. Our experiments shows the pattern network model outperformed other fixed order and

variable order Markov models. Meanwhile, the generated pattern network could be a very useful tool for us to analyze user's mobility behaviors.

In Chapter 5, we gave a formal definition of mobility change, called Point of Change (POC), to describe people's new and unusual mobility behaviors. Our motivation to predicting POCs is that a large portion of locations in a trajectory are unique or do not appear in a pattern. Usually, these locations can not be successfully predicted by traditional methods. Therefore, we extended the pattern network to the spatial-temporal version so that the unusual mobility behaviors can be learnt and predicted. Based on the advantages of pattern network, our ST-Pattern Network model can predict future POCs incrementally and adapt to recent trajectory via a weight propagation step. Our experiments show the POC prediction accuracies can be improved by our model over traditional Markov models. Additionally, we analyzed the smartphone usages on POC locations, our results indicated people utilize their smartphones more intensively on the POC locations. This observation suggests that future recommendation systems can provide more assistances for the people in unusual case.

6.1 Future Works

Our works can be extended to three directions in the future.

(1) Investigating Recurrent Neural Network for mobility prediction

Recently, application of recurrent neural network (RNN) becomes a hot topic in natural language processing. RNN has the ability to memorize long-term historical information with a cell unit called LSTM [52]. By connecting LSTMs to form a

neural network, RNN is able to predict a sequence or value at a particular time point. Therefore, RNN provides a new prospective for mobility prediction as well.

To use RNN for mobility prediction, we design three layers network including input layer, hidden layer with LSTM units and output layer. At each time point, an index of location is inputted into the input layer until the entire sequence or pattern is read. For model training, the next location of the inputted sequence can be used as its label. With more and more sequences or patterns being learned, the RNN is able to memorize various location transitions from short-time to long-term dependency. To predict the next location, we can simply input the current location sequence, and the corresponding output at the last time stamp is our expected prediction result.

To compare the RNN with our pattern network model, we find the RNN is not only suitable to predict the next location, but also can be used to predict near-future location. We can simply use near-future location as the training label. Moreover, the RNN is easier to cope to uncertainty data, because neural network contains more parameters to memorize the probability of location transition, and the output is generated by triggering the threshold of certain hidden units. Oppositely, the pattern network uses explicit sequential patterns to build a mobility model. Therefore, the prediction is limited on existing patterns, and has less ability to handle uncertainty data.

However, to achieve high prediction accuracy, RNN requires huge volume of data for model training, and the training processing is very time-consuming even working on distributed system. The model training brings new challenges for applying RNN in mobility scenario, because people's trajectories are updated frequently. Including any new trajectory requires the model to be trained until it is converged

again. Fortunately, the strategy of training pattern network model is by propagating the pattern weights to patterns' neighbors. Hence, it does not need many iterations, and deep propagation, which makes our pattern network model more efficient.

Considering the power of neural network and deep learning, it is worth to investigate how to modify RNN into a spatial-temporal model by incorporating people's movement information.

(2) Discovering and maintaining multiple users' patterns on one suffix pattern tree

Currently, a suffix pattern tree is only constructed from an individual trajectory. Pattern discovery and maintenance are achieved on a single suffix tree. However, in a big data scenario, the amount of users to be managed could be million. If every user is required to maintain a suffix pattern tree, the overall workloads of computing and storage consumption will be significantly increased. A better strategy is to merge multiple suffix pattern trees into one tree. It will be great if the online near-supermaximal repeat discovery and tandem structure detection could also be achieved on this hybrid suffix pattern tree. Fortunately, a generalized suffix tree [53] has been proposed, which is able to manage a set of strings and can be constructed in linear time with linear storage requirement. Therefore, one possible future work is to explore how to realize the online pattern discovery on the generalized suffix tree. Moreover, since multiple users will access the suffix tree concurrently, it is possible for us to find a group of users who are visiting the same place at the same time. Thus, group behaviors are likely to be detected.

(3) Applying graph algorithms on the pattern network to investigate user's mobility behaviors

Pattern network transforms the trajectory from a sequence to a graph. It contains complete historical information of location transitions and pattern shifts. The graph presentation of trajectory provides an opportunity for us to investigate people's mobilities via many graph algorithms. We may use community detection algorithm to find a set of connected nodes to represent a user's compound life pattern. Or, we can use PageRank algorithm to find the most important pattern. The locations inside this pattern may be the residence place or working place for a user. Moreover, it is impossible to measure the trajectory similarity of two users if they are living in different cities. Because the locations in both trajectories are totally different, direct comparison of two sequences can not be performed. However, pattern network provides a way to compare two users based on their pattern network topologies. It is possible that similar users may have the similar mobility behaviors, and these characteristics can be reflected on their pattern networks. Therefore, pattern network could be a powerful tool to analyze and compare multiple users' mobility behaviors in the future.

References

[1] Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, ICDE '01, pages 215–, Washington, DC, USA, 2001. IEEE Computer Society.

[2] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. Mining periodic behaviors for moving objects. In *KDD*, pages 1099–1108, 2010.

[3] Huiping Cao, Nikos Mamoulis, and David W. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, pages 82–89, 2005.

[4] Yang Ye, Yu Zheng, Yukun Chen, Jianhua Feng, and Xing Xie. Mining individual life pattern based on location history. In *Mobile Data Management*, pages 1–10, 2009.

[5] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-Lszl Barabsi. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

[6] Anirudh Kondaveeti. Spatio-temporal data mining to detect changes and clusters in trajectories. ARIZONA STATE UNIVERSITY.

[7] James McInerney, Sebastian Stein, Alex Rogers, and Nicholas R. Jennings. Breaking the habit: Measuring and predicting departures from routine in individual human mobility. *Pervasive and Mobile Computing*, 9(6):808–822, 2013.

[8] Jie Bao, Yu Zheng, David Wilkie, and Mohamed F. Mokbel. A survey on recommendations in location-based social networks. *submitted to GeoInformatica*, August 2013.

[9] http://lifemap.yonsei.ac.kr/wps/index.php/about-lifemap/.

[10] Bolei Zhou, Xiaogang Wang, and Xiaoou Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents. In *CVPR*, pages 2871–2878, 2012.

[11] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, and Yan Huang. T-drive: Driving directions based on taxi trajectories. In *ACM SIGSPATIAL GIS 2010*. Association for Computing Machinery, Inc., November 2010.

[12] Gianluca Antonini, Santiago Venegas Martinez, Michel Bierlaire, and Jean Philippe Thiran. Behavioral priors for detection and tracking of pedestrians in video sequences. *INT. J. COMPUT. VIS*, 69(2):159–180, 2006.

[13] Xiaogang Wang, Kinh Tieu, and Eric Grimson. Learning semantic scene models by trajectory analysis. In *In ECCV (3) (2006*, pages 110–123, 2006.

[14] Pradeep Kumar, Pradeep Kumar, P. Radha Krishna, and S. Bapi Raju. *Pattern Discovery Using Sequence Data Mining: Applications and Studies*. IGI Global, Hershey, PA, USA, 1st edition, 2011.

[15] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, pages 3–17, 1996.

[16] Peter Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (Swat 1973)*, SWAT '73, pages 1–11, Washington, DC, USA, 1973. IEEE Computer Society.

[17] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[18] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York,

NY, USA, 1997.

[19] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 319–327, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[20] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. of Discrete Algorithms*, 2(1):53–86, March 2004.

[21] Zhenhui Li, Jingjing Wang, and Jiawei Han. Mining event periodicity from incomplete observations. In *KDD*, pages 444–452, 2012.

[22] Alicia Rodriguez-Carrion, Carlos García-Rubio, and Celeste Campo. Performance evaluation of lz-based location prediction algorithms in cellular networks. *IEEE Communications Letters*, 14(8):707–709, 2010.

[23] Quan Yuan, Ionut Cardei, and Jie Wu. Predict and relay: an efficient routing in disruption-tolerant networks. In *MobiHoc*, pages 95–104, 2009.

[24] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive wi-fi mobility data. In *INFOCOM*, 2004.

[25] Alicia Rodriguez-Carrion, Celeste Campo, and Carlos García-Rubio. Recommendations on the move. In *Recommender Systems for the Social Web*, pages 179–193. 2012.

[26] Yohan Chon, Elmurod Talipov, Hyojeong Shin, and Hojung Cha. Mobility prediction-based smartphone energy optimization for everyday location monitoring. In *SenSys*, pages 82–95, 2011.

[27] Lu Xin, Wetter Erik, Bharti Nita, Andrew J. Tatem, and Bengtsson Linus. Approaching the limit of predictability in human mobility. volume 3. Macmillan

Publishers Limited.

[28] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, MPM '12, pages 3:1–3:6, New York, NY, USA, 2012. ACM.

[29] Wesley Mathew, Ruben Raposo, and Bruno Martins. Predicting future locations with hidden markov models. In *UbiComp*, pages 911–918, 2012.

[30] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD*, pages 1082–1090, 2011.

[31] Hong Cheng, Jihang Ye, and Zhe Zhu. What's your next move: User activity prediction in location-based social networks. In *SDM*, pages 171–179, 2013.

[32] Jing Yuan, Yu Zheng, and Xing Xie. Discovering regions of different functions in a city using human mobility and pois. In *KDD*, pages 186–194, 2012.

[33] Mao Ye, Dong Shou, Wang-Chien Lee, Peifeng Yin, and Krzysztof Janowicz. On the semantic annotation of places in location-based social networks. In *KDD*, pages 520–528, 2011.

[34] Yohan Chon, Yunjong Kim, and Hojung Cha. Autonomous place naming system using opportunistic crowdsensing and knowledge from crowdsourcing. In *IPSN*, pages 19–30, 2013.

[35] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *ICDE*, pages 140–149, 2008.

[36] Xiaolei Li, Zhenhui Li, Jiawei Han, and Jae-Gil Lee. Temporal outlier detection in vehicle traffic data. In *ICDE*, pages 1319–1322, 2009.

[37] Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. Roam: Rule- and motif-based anomaly detection in massive moving object data sets. In *SDM*. SIAM, 2007.

[38] L. Gereneser and G. Molnar-Saska. Change detection of hidden markov models. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 2, pages 1754–1758 Vol.2, Dec 2004.

[39] Cheng-Der Fuh. Asymptotic operating characteristics of an optimal change point detection in hidden markov models. *The Annals of Statistics*, 32(5):2305–2339, 10 2004.

[40] Jens Stoye and Dan Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theor. Comput. Sci.*, 270(1-2):843–850, January 2002.

[41] Jr. Langdon, G.G. A note on the ziv - lempel model for compressing individual sequences (corresp.). *Information Theory, IEEE Transactions on*, 29(2):284–287, Mar 1983.

[42] J. Rissanen. A universal data compression system. *IEEE Trans. Inf. Theor.*, 29(5):656–664, September 2006.

[43] http://www.foursquare.org.

[44] Alberto Apostolico and Franco P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22:297–315, 1983.

[45] Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf. Process. Lett.*, 12(5):244–250, 1981.

[46] D. Schmidt. A c implementation of ukkonen's suffix tree-building algorithm, with test suite and tree print. 2013.

[47] Huiji Gao, Jiliang Tang, and Huan Liu. Exploring social-historical ties on location-based social networks. In *ICWSM*, 2012.

[48] http://www.crawdad.org.

[49] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: Sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4):255–268, March 2006.

[50] Yohan Chon, Hyojeong Shin, Elmurod Talipov, and Hojung Cha. Evaluating mobility models for temporal prediction with high-granularity mobility data. In *PerCom*, pages 206–212, 2012.

[51] Huiji Gao, Jiliang Tang, and Huan Liu. Mobile location prediction in spatio-temporal context. In *Nokia Mobile Data Challenge Workshop*, 2012.

[52] F.A. Gers and J. Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, 12(6):1333–1340, Nov 2001.

[53] P. Bieganski, J. Riedl, J.V. Cartis, and E.F. Retzel. Generalized suffix trees for biological sequence data: applications and implementation. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 5, pages 35–44, Jan 1994.