國立交通大學

資訊科學與工程研究所

博 士 論 文

探勘智慧型手機中應用程式使用行為之研究

A Study on Mining Apps Usage Behavior in Smartphones

研 究 生：廖忠訓

指導教授：彭文志　教授

中 華 民 國 一 百 零 二 年 八 月

探勘智慧型手機中應用程式使用行為之研究
A Study on Mining Apps Usage Behavior in Smartphones

研 究 生：廖忠訓　　Student：Zhung-Xun Liao

指導教授：彭文志　　Advisor：Wen-Chih Peng

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

博 士 論 文

A Dissertation

Submitted to Department of Computer Science

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

August 2013

Hsinchu, Taiwan, Republic of China

中華民國一百零二年八月

探勘智慧型手機中應用程式使用行爲之研究

學生: 廖忠訓　　指導教授: 彭文志

國立交通大學資訊科學與工程研究所博士班

摘要

隨著智慧型手機的普及，愈來愈多的行爲應用程式(mobile applications)被開發及設計。使用者可以由網路上下載這些Apps來處理各種需求，例如：相機、地圖、瀏覽器、音樂播放器，…等。並且，由於手機的行動性，這些下載、執行及移除的行爲可以發生在任何的時間及任何的地點。因爲，智慧型手機的使用記錄便成爲一個複雜的時間空間資料。在本論文中，主要將探討四個主題：(1)以個人化特徵挑選進行Apps使用預測、(2)以時間性履歷進行Apps使用預測、(3)使用者Apps動態喜好模型、(4)探勘以多維度序列爲基礎之使用者Apps使用樣式。

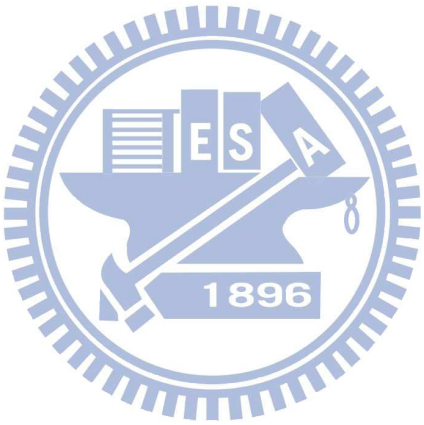
在第一個主題中，我們將收集智慧型手機上的各種感測器，包含硬體感測器，例如：時間、加速度、地理位置、等…；軟體感測器，例如：Apps的使用順序。這些感測裝置的讀數可以有效的利用來預測使用者的Apps使用情況。然而，智慧型手機上的感測裝置非常多，收集這些讀數不只會造成儲存空間上的浪費，也會花費額外的能源來進行感測。因此，我們在這個主題中，會以個人化的觀點進行特徵挑選，只有有利於預測該使用者的Apps使用的感測裝置會進行感測。如此便用大量的降低儲存訓練資料所需的空間及感測時消耗的能量。

在第二個主題中，我們只參考時間的因素來進行Apps使用預測。我們利用傅立葉轉換來得到Apps的使用週期，接著根據這些找出來的週期爲每一個App進立他的時間性履歷。此時，由於時間資訊已經被傅立葉轉換所消除，我們必需將具有相同週期但發生在不同時間的行爲區隔開。在這裡，我們利用階層式分群法來將類似的行爲分爲一群，並且視其爲一種使用行爲。最後，我們提出一個分數計算的系統，可以計算出每一個App在目前時間可能被執行的機會。但由於這個機率的運算需要對App的使用機率密度進行積分，而積分的計算對於手機來說是一項耗費能源的動作。因此，我們提出一個以柴比雪夫不等式爲基礎的分數計算方式，可以不需要進行積分，便能計算出App可能被使用的機率。

在第三個主題中，我們發現使用者的喜好是會隨著時間而改變。但大部份的使用者不會對他們所下載的Apps進行評分，而不可能在他們每次改變喜好時，不停的重新評分。在這個主題中，我們以上一個時

間點的喜好搭配上目前時間點的使用次數來計算目前時間點的喜好。然而，使用次數並無法完整的反應出使用者的喜好。例如：對於某些使用者來說，通訊Apps，像是Line、Whatsapp的使用次數必定比生產力工具來得多。於是，在這個主題中，我們以線性迴歸來代表使用者喜好變化的趨勢，而該使用者喜好的變化，便能以迴歸線的斜率來表示。

在第四個主題中，我們設計了一個特殊的序列樣式，稱爲多維度序列樣式。由於Apps可能歸屬於各種不同的類別，因爲，我們可以將Apps的使用記錄看成是一個多維度(類別)的序列。而最常出現的多維度序列樣式，便可以用來代表這個使用者的使用行爲。在這裡，我們提出一個傳遞式的探勘方式，只需要在第一個維度進行序列樣式探，接著再透過傳遞的方式來組合其他維度相對應的樣式，便能組合出一個多維度的序列樣式。除此之外，我們還提出一個增加效率的資料結構，可以只找出最短的樣式，再利用此資料結構來延長樣式的長度。



A Study on Mining Apps Usage Behavior in Smartphones

Student: Zhung-Xun Liao Advisor: Dr. Wen-Chih Peng

Department of Computer Science

National Chiao Tung University

ABSTRACT

Smartphones have played an important role nowadays. There are more and more mobile applications (Apps) designed for smartphones. Users could download and execute different Apps for different purposes, such as camera, maps, browser, mp3 player, and so on. Furthermore, users could buy (download), launch, close and remove Apps in any location and any time due to the powerful mobility of smartphones. Therefore, the usage behavior of smartphone obviously could be seen as a complex spatio-temporal data. In this thesis, we will focus on 1) identifying users personal features for predicting their mobile Apps usage, 2) predicting the Apps to be launched regarding the usage trace, 3) modelling the dynamic preference of Apps usage, and 4) discovering users mobile usage patterns which are represented as multi-domain sequential patterns.

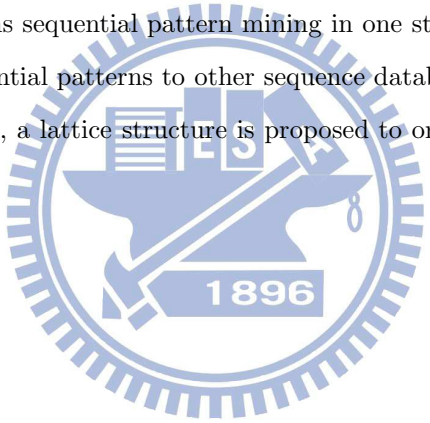
In the first work, we predict Apps usage for users according to their personalized features which are collected from sensors attached on smartphones. We claim that the Apps usage behavior would be affected by the hardware sensors, such as time, GPS, Accelerometer, etc. and the software sensors, such as the Apps usage sessions. Thus, we could predict user's Apps usage in advance through collecting those sensor readings. However, to collect all of the sensors readings is impractical and inefficient. Here, we only select a set of most useful sensors for every individual user. Therefore, the training data size and the sensing energy could be reduced.

In the second work, the temporal profiles is discovered for mobile Apps. We identify the periodicity of Apps via Fourier transform and consequently, the temporal profiles are thus constructed according to the usage periods of Apps. Furthermore, due to the temporal information is eliminated after we perform the Fourier transform, we have to identify the different sub-patterns which share the same period. Thus,

a hierarchical clustering is adopted to group similar sub-patterns and different groups are considered as different usage behaviors. Finally, we propose a scoring system based on Chebychev inequality which calculate the usage probability without performing integral on the usage density probability function.

In the third work, we observe that a user's preference to the mobile Apps (s)he has installed is dynamic. However, users seldom rate their Apps and even re-rate them when their reference is changed. In this work, we collect the mobile Apps usage trace of a user and model the current preference according to previous preference and the current usage counts. However, the usage count does not reflect the preference directly. For example, for some users, the usage count of an IM App is definitely higher than that of a productive App. Therefore, we model the usage trend by linear regression and thus the preference change is based on the slope of the regression line.

In the forth work, we design a novel sequential pattern across multiple sequence databases to model the mobile Apps usage behavior and proposed an efficient algorithm, called PropagatedMine. The proposed PropagatedMine performs sequential pattern mining in one starting sequence database, and then propagate the discovered sequential patterns to other sequence databases. Furthermore, to reducing the amount of propagated patterns, a lattice structure is proposed to organize and composes multi-domain sequential patterns.



致 謝

大家都說當學生是最幸福的時光，而我一不小心就過得太幸福了，人生有多少個六年呢?扣掉前前後後的時間，眞正能有自己的想法、做自己想做的事的時間，大概只有五到六個吧，而其中一個，我把他拿來念博士班。發表了七篇論文，在美國住了一年，結交了許多志同道合的好朋友。未來還有多少個六年可以如此燦爛輝煌?

謝謝我的指導教授彭文志老師，從進來念博士班開始就一直不斷的鼓勵我，讓我相信只要不停的嘗試，努力的成果終將會被看見。這六年來，他用過去的經驗給我許多寶貴的建議，從他身上我學到了要勇敢的嘗試並且不怕失敗，這不僅僅是學術研究上的成長，更是對人生態度的成熟。另外，也很感謝他帶我認識許多國內外傑出的教授，尤其是Data Mining的大師Prof. Philip Yu，讓我有機會能到芝加哥進行一年的訪問，體驗不同的文化與生活，開拓了眼界。

Prof. Philip Yu是一位標準的學者，熱愛研究但不是那種只侷限在自己領域裡的教授，也因爲這樣的特性，使得他總是能夠在很短的時間內，激發出不同的思維，這也是他和許多號稱大師的教授不同之處。每當我的研究有缺陷，思考得不夠清楚時，他總是能夠從其他的視野看到研究的價值，進而修改研究的方向，或補充不足之處，而不會因爲遇到問題就全盤推翻所有的研究成果，這樣的態度也改變了我看待事情的角度，懂得去欣賞每件事物的美，學習保留他們的優點。在美國的這一年是博士班生涯裡最自由自在的時光，可以專心的做自己的研究，同時，也讓我重拾對研究的熱情。

最重要的，要感謝我的家人，每次過年過節回家，你們總是不厭其煩的問我什麼時候要畢業，雖然場面都會因此變得很尷尬，但你們的關心卻支持著我，讓我無後顧之優，安心的把博士班念完。感謝實驗室裡每一位陪我渡過漫漫長路的學弟妹們，因爲你們讓苦澀變的甘甜。最後，我要感謝那些看不起我及我做的研究的人，有一句話是這麼說的，「當你需要勇氣時，上帝就給你阻礙」，我知道你們是上帝派來給我勇氣的，謝謝你們。

人生的每個選擇都決定了未來不一樣的路，這六年來遇過好幾個關鍵的抉擇，換論文方向、國科會千里馬計畫到芝加哥訪問、進入HTC實習，遇到的每個人，做的每個決定都成就了現在的我及這本論文。沒有人知道如果當初做了別的決定或是往另一個方向走，現在的我會在哪裡?如果當初選擇先畢業而沒有去經歷這些看似荒唐又浪費時間的事，現在的我又會是甚麼模樣?然而，我喜歡現在的自己，謝謝你們讓我成爲這樣的我。

謹以本論文獻給你們，因爲你們豐富了我的人生，讓這篇論文有了生命。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the increasing number of mobile Apps developed, they are now closely integrated into daily life. Users install more and more Apps on their smartphones. Therefore, predicting Apps usage is a prerequisite for helping users 1) find the Apps they want to use and improve the user experiences; 2) pre-load complex resources, such as the graphics and GPS positioning, and launch Apps faster; 3) remove the useless Apps from memory to save energy consumption. Furthermore, the Apps usage reflects the activity users are doing. We can better capture the users behavior via discovering their Apps usage. Many research works indicated that the Apps usage highly depends on the context information, such as time, location, mobile sensor readings, and the Apps usage.

Figure 1.1 depicts the overview of this dissertation, where the first and second works focus on predicting Apps usage by considering feature selection and temporal information respectively. The third work models the dynamic preferences of users, and the forth work discovers the sequential patterns across multiple categories of Apps. In the future, work 3 and work 4 could be utilized to enhance the accuracy of Apps usage prediction. The brief introduction of each chapter in this dissertation is given as follows:

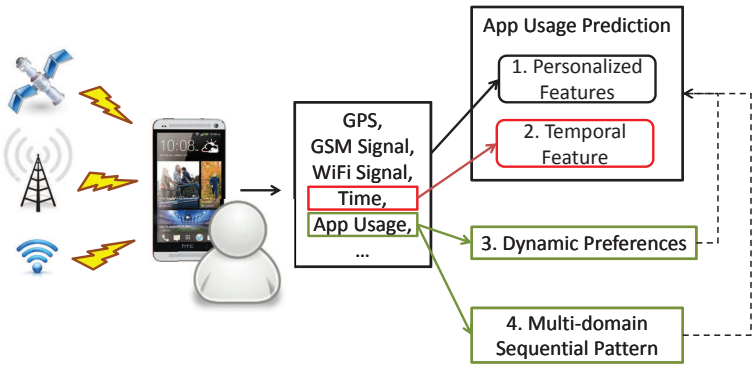


Figure 1.1: Overview of this dissertation.

1.1 On the Feature Discovery for App Usage Prediction in Smartphones

We develop a framework to predict mobile Apps that are most likely to be used regarding the current device status of a smartphone. Such an Apps usage prediction framework is a crucial prerequisite for fast App launching, intelligent user experience, and power management of smartphones. By analyzing real App usage log data, we discover two kinds of features: The Explicit Feature (EF) from sensing readings of built-in sensors, and the Implicit Feature (IF) from App usage relations. The IF feature is derived by constructing the proposed App Usage Graph (abbreviated as AUG) that models App usage transitions. In light of AUG, we are able to discover usage relations among Apps. Since users may have different usage behaviors on their smartphones, we further propose one personalized feature selection algorithm. We explore minimum description length (MDL) from the training data and select those features which need less length to describe the training data. The personalized feature selection can successfully reduce the log size and the prediction time. Finally, we adopt the kNN classification model to predict Apps usage. Note that through the features selected by the proposed personalized feature selection algorithm, we only need to keep these features, which in turn reduces the prediction time and avoids the curse of dimensionality when using the kNN classifier. We conduct a comprehensive experimental study based on a real mobile App usage dataset. The results demonstrate the effectiveness of the proposed framework and show the predictive capability for App usage prediction.

1.2 Mining Temporal Profiles of Mobile Applications for Usage Prediction

As many research works indicate the prediction ability of temporal information, we take only the temporal information into account and see how is the performance. Due to the proliferation of mobile applications (abbreviated as Apps) on smart phones, users can install many Apps to facilitate their life. Usually, users browse their Apps by swiping touch screen on smart phones, and are likely to spend much time on browsing Apps. We design an AppNow widget that is able to predict users' Apps usage. Therefore, users could simply execute Apps from the widget. The main theme of this chapter is to construct the temporal profiles which identify the relation between Apps and their usage times. In light of the temporal profiles of Apps, the AppNow widget predicts a list of Apps which are most likely to be used at the current time. AppNow consists of three components, the usage logger, the temporal profile constructor and the Apps predictor. First, the usage logger records every App start time. Then, the temporal profiles are built by applying Discrete Fourier Transform and exploring usage periods and specific times. Finally, the system calculates the usage probability at current time for each App and shows a list of Apps with highest

probability. In our experiments, we collected real usage traces to show that the accuracy of AppNow could reach 86% for identifying temporal profiles and 90% for predicting App usage.

1.3 Mining Usage Traces of Mobile Apps for Dynamic Preference Prediction

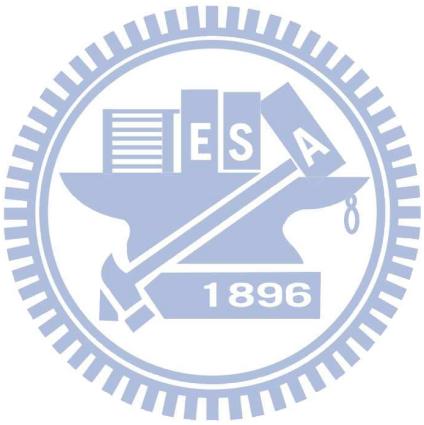
Due to a huge amount of mobile applications (abbreviated as Apps), for Apps providers, the usage preferences of Apps are important in recommending Apps, downloading Apps and promoting Apps. We predict and quantize users' dynamic preferences by exploring their usage traces of Apps. To address the dynamic preference prediction problem, we propose Mode-based Prediction (abbreviated as MBP) and Reference-based Prediction (abbreviated as RBP) algorithms. Both MBP and RBP consist of two phases: the trend detection phase and the change estimation phase. In the trend detection phase, both algorithms determine whether the preference of an App is increasing or decreasing. Then, in the change estimation phase, the amount of preference change is calculated. In particular, MBP adopts users' current usage mode (active or inactive), and then estimates the amount of change via our proposed utility model. On the other hand, RBP calculates an expected number of usage as a reference, and then builds a probabilistic model to estimate the change of preference by comparing the real usage and the reference. We conduct comprehensive experiments using two App usage traces and one music listening log, the Last.fm dataset, to validate our proposed algorithms. The experimental results show that both MBP and RBP outperform the usage-based method that is based solely on the number of usages.

1.4 Mining Sequential Patterns Across Multiple Sequence Databases

Given a set of sequence databases across multiple domains, we aim at mining multi-domain sequential patterns, where a multi-domain sequential pattern is a sequence of events whose occurrence time is within a pre-defined time window. We first propose algorithm Naive in which multiple sequence databases are joined as one sequence database for utilizing traditional sequential pattern mining algorithms (e.g., PrefixSpan). Due to the nature of join operations, algorithm Naive is costly and is developed for comparison purposes. Thus, we propose two algorithms without any join operations for mining multi-domain sequential patterns. Explicitly, algorithm IndividualMine derives sequential patterns in each domain and then iteratively combines sequential patterns among sequence databases of multiple domains to derive candidate multi-domain sequential patterns. However, not all sequential patterns mined in the sequence database of each domain are able to form multi-domain sequential patterns. To avoid the mining cost incurred in algorithm IndividualMine, algorithm PropagatedMine is developed. Algorithm PropagatedMine first performs one sequential pattern mining from one sequence database. In light of sequential patterns

mined, algorithm PropagatedMine propagates sequential patterns mined to other sequence databases. Furthermore, sequential patterns mined are represented as a lattice structure for further reducing the number of sequential patterns to be propagated. In addition, we develop some mechanisms to allow some empty sets in multi-domain sequential patterns. Performance of the proposed algorithms is comparatively analyzed and sensitivity analysis is conducted. Experimental results show that by exploring propagation and lattice structures, algorithm PropagatedMine outperforms algorithm IndividualMine in terms of efficiency (i.e., the execution time).

The rest of the paper is organized as follows. Chapter 2 discusses the explicit and implicit features and personalized feature selection algorithm. Chapter 3 reduces the Apps usage problem to only consider the temporal information. Chapter 4 proposes two dynamic preferences prediction algorithms. Chapter 5 states the individualMine and propagatedMine algorithms to discover sequential patterns across multiple domains. Finally, this dissertation is concluded in Chapter 6.



Chapter 2

On the Feature Discovery for App Usage Prediction in Smartphones

2.1 Introduction

With the increasing number of smartphones, mobile applications (Apps) have been developed rapidly to satisfy users' needs [76, 15, 63, 72]. Users can easily download and install Apps on their smartphones to facilitate their daily lives. For example, users use their smartphones for Web browsing, shopping and socializing [46, 4]. By analyzing the collected real Apps usage log data, the average number of Apps in a user's smartphone is around 56. For some users, the number of Apps is up to 150. As many Apps are installed on a smartphone, users need to spend more time swiping screens and finding the Apps they want to use. From our observation, each user has on average 40 launches per day. In addition, the launch delay of Apps becomes longer as their functionality becomes more complicated. In [73], the authors investigated the launch delay of Apps. Even simple Apps (e.g., weather report) need 10 seconds, while complicated Apps (e.g., games) need more than 20 seconds to reach a playable state. Although some Apps could load stale content first and fetch new data simultaneously, they still need several seconds to complete loading.

To ease the inconvenience of searching for Apps [41, 64] and to reduce the delay in launching Apps [73], one possible way is to predict which Apps will be used before the user actually needs them. Although both the iOS and Android systems list the most recently used (MRU) Apps to help users relaunch Apps, this method only works for those Apps which would be immediately relaunched within a short period. Another common method is to predict the most frequently used (MFU) Apps. However, when a user has a lot of frequently used Apps, the MFU method has very poor accuracy. In our experiments, these two methods are the baseline methods for comparison.

Recently, some research works have addressed the Apps usage prediction problems [73, 41, 64]. In [41],

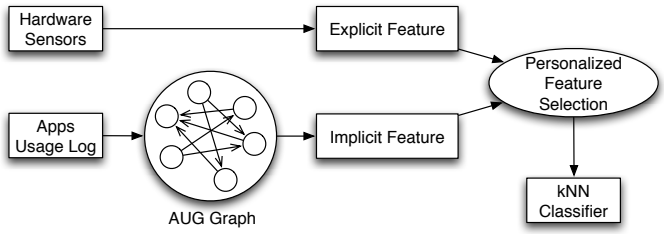


Figure 2.1: Overview of kNN-based App Prediction framework.

a temporal profile is built to represent the usage history of an App. The temporal profile records the usage time and usage period of the App. Then, when a query time is given, the usage probability of each App could be calculated through comparing the difference between the temporal profile and the query time. However, since they only consider the periodicity feature of Apps, some Apps with no significant periods cannot be predicted by their temporal profiles. In [73], the authors adopted three features to predict Apps usage: time, location, and used Apps. Based on those three features, they designed and built a system to remedy slow App launches. However, they always use these three features to predict different users' usage, which is impractical as users could have different usage behavior. For example, the location information could be less useful for those users who have lower mobility. We claim that the features which are able to accurately predict Apps usage are different for different users and different Apps. The authors in [64] collected 37 features from accelerometer, Wi-Fi signal strength, battery level, etc., and proposed a Naive Bayes classification method to predict Apps usage. However, a Naive Bayes classification method needs sufficient training data to calculate the conditional probability, which does not always hold. Therefore, the system would fail to predict Apps if there are not exactly the same instances existing in the training dataset. In addition, they still apply all the same features to each user, instead of selecting personalized features for different users with different usage behaviors.

In this chapter, we adopt the concept of minimum description length (MDL) to select personalized features for different users and propose a kNN-based App Prediction framework, called KAP, to predict Apps usage. Once we distinguish the useful and useless features, only the useful features need to be collected. Therefore, the size of the log data could be reduced. The overall framework is shown in Figure 2.1. KAP investigates features from both explicit and implicit aspects. The explicit feature is a set of sensor readings from built-in hardware sensors, such as GPS, time, accelerometers, etc. On the other hand, the implicit feature is referred to as the correlations of Apps usage. To capture these correlations, the implicit feature is represented as the transition probability among Apps.

For the explicit feature, we focus on three types of hardware sensors: 1) device sensors, such as free space, free ram, and battery level, 2) environmental sensors, such as time, GSM signal, and Wi-Fi signal, and 3) personal sensors: acceleration, speed, heading, and location. We claim that the usage of different Apps is related to different types of sensors. Obviously, the advantages of selecting sensors for the explicit

feature is that it reduces the effect of noisy data and also saves power and storage consumption for logging data and performing the prediction.

For the implicit feature, we calculate the transition probability for each App. However, the previous works [73, 64] only take the usage order into account, and not the time duration between Apps. We claim that the length between Apps usage means different things. For example, users may take pictures via a camera App and upload those pictures to Facebook. However, some users may upload pictures immediately, while others would upload them when they have a Wi-Fi connection. Therefore, the time duration between camera and Facebook use depends on different users and different usage behaviors. To model the usage relation among Apps, an Apps Usage Graph (AUG), which is a weighted directed graph, is proposed. The weight on each edge is formulated as an exponential distribution to describe the historical usage durations. Based on AUG, the implicit feature of each training instance is derived by traversing the AUG. Consequently, the implicit feature of each testing case is derived by an iterative refinement process.

With both explicit and implicit features, KAP adopts a kNN classification model to predict Apps usage which is represented as class labels. In the experimental study, the proposed KAP framework outperforms both baseline methods and achieves accuracy of 95%. We also show that the personalized sensor selection for the explicit feature is efficient and effective. In addition, the implicit feature is useful for improving the prediction accuracy of KAP.

The major contributions of this research work are summarized as follows.

- We address the problem of Apps usage prediction by discovering different feature sets to fulfill different users' Apps usage behavior, and propose the concept of explicit and implicit features for Apps usage prediction.
- We estimate the distribution of the transition probability among Apps and design an Apps Usage Graph (AUG) to model both Apps usage order and transition intervals. Two algorithms are proposed to extract the implicit features from the AUG graph for training and testing purposes respectively.
- We propose a personalized feature selection algorithm in which one could explore MDL to determine a personalized set of features while still guaranteeing the accuracy of the predictions.
- A comprehensive performance evaluation is conducted on real datasets, and our proposed framework outperforms the state-of-the-art methods [64].

The rest of this chapter is organized as follows. Section 2.2 investigates the related works which discuss the conventional prediction problem and Apps usage prediction. Section 2.3 introduces the explicit and implicit features. Section 2.4 presents the mechanism of personalized feature selection. Section 2.5 conducts extensive and comprehensive experiments. Finally, this chapter is concluded with Section 4.6.

2.2 Related Works

To the best of our knowledge, the prediction problem of Apps usage in this chapter is quite different from the conventional works. We focus on not only analysing usage history to model users' behavior, but on personalizing varied types of features including hardware and software sensors attached to smartphones. The proposed algorithm selects different features for different users to satisfy their usage behavior. Although there have been many research works solving the prediction problem in different domains, such as music items or playlist prediction [10], dynamic preference prediction [44, 37], location prediction [38, 62, 50], social links prediction [17, 45], and so on, the prediction methods are only based on analysing the usage history. In [33], the author selected features from multiple data streams, but the goal is to solve the communication problem in a distributed system.

Currently, only a few studies discuss mobile Apps usage prediction. Although the authors in [49] adopted location and time information to improve the accuracy of Apps usage prediction, the total number of Apps is only 15. Concurrently, in [31], the authors stated that the prediction accuracy could achieve 98.9%, but they still only focus on predicting 9 Apps from a set of 15. In [73], the authors solved the prediction problem through multiple features from 1) location, 2) temporal burst, and 3) trigger/follower relation. However, they did not analyze the importance of each feature. Therefore, for different users, they always use the same three features to predict their Apps usage. In [64], the authors investigated all possible sensors attached to a smartphone and adopted a Naive Bayes classification to predict the Apps usage. However, collecting all possible sensors is inefficient and impractical. Moreover, the useful sensors for different users could vary according to users' usage behavior. We claim that for different users, we need to use different sets of features to predict their usage. In this chapter, we collect only the subset of all features which are personalized for different users.

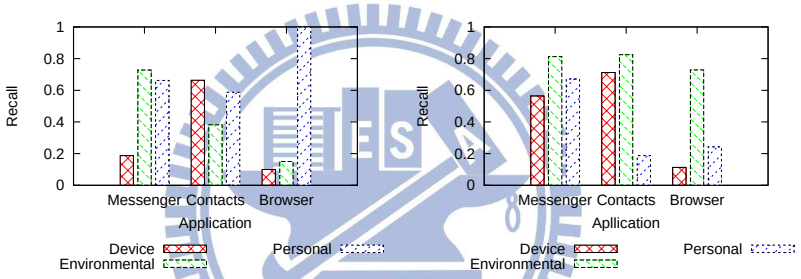
This chapter is the first research work which discusses how to select suitable sensors and features for different users to predict their Apps usage. Through the personalized feature selection, we could perform more accurate predictions for varied types of usage bahavior, reduce the dimensionality of the feature space, and further save energy and storage consumption. In addition, the proposed KAP framework derives the implicit feature by modelling the usage transition among Apps.

2.3 Explicit and Implicit Features

In this chapter, we separate the features into two main categories: the explicit feature and the implicit feature. The explicit feature represents the sensor readings which are explicitly readable and observable. The implicit feature is the Apps usage relations.

Table 2.1: Hardware sensors for the explicit feature.

Sensors	Contextual Information
Location	Longitude Latitude Altitude Location Cluster
Time	Hour of day Day of week
Battery	Battery Level Charging status
Accelerometer	Avg. and std. dev. of {x, y, z} Acceleration changes speed Heading
Wi-Fi Signal	Received signal
GSM Signal	Signal Strength
System	Free space of each drive Free RAM



(a) $User_1$ (b) $User_2$

Figure 2.2: Varied recalls of predicting Apps usage via different types of sensors for different users.

2.3.1 Explicit Feature Collection

Table 2.1 shows the hardware sensors we use for the explicit feature. As different models of smartphones could have different sets of hardware sensors, we only list the most common ones whose readings are easy to record. It is totally free to add or remove any hardware sensors here.

To show the prediction ability of different types of mobile sensors, we randomly select two users from the collected dataset and perform kNN classification via the three types of sensors respectively to predict their Apps usage. Figure 2.2 shows the prediction recall of "Messenger", "Contacts", and "Browser" for the two users. As can be seen in Figure 2.2, personal sensors would be a good explicit feature for predicting $user_1$'s Apps usage, while environmental sensors are good for $user_2$. The reason is that $user_2$ probably needs a Wi-Fi signal to access the Internet.

2.3.2 Implicit Feature Extraction

The implicit feature formulates the usage transitions among Apps in a usage session. As mentioned in [73], users use a series of Apps, called a usage session, to complete a specific task. For example, one user could use "Maps" when travelling to a sightseeing spot, then use camera to take photos, and upload

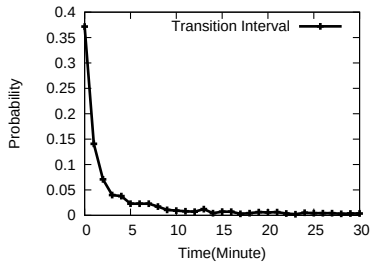


Figure 2.3: The PDF of the duration of two consecutive App launches.

those photos to Facebook. Thus, the series of using "Maps", "Camera" and "Facebook" is called a usage session, denoted as "Map" $\xrightarrow{\delta_1}$"Camera" $\xrightarrow{\delta_2}$"Facebook", where δ_1 and δ_2 represent the transition intervals.

The implicit feature of "Facebook" in this usage session is thus $< p_{MF}(\delta_1), p_{CF}(\delta_1 + \delta_2), p_{FF}(\infty) >$, where $p_{MF}(\cdot)$, $p_{CF}(\cdot)$, and $p_{FF}(\cdot)$ are probability models which represent the probability of using "Maps", "Camera" and "Facebook" respectively before using "Facebook" with the transition interval as the random variable. Note that because there is no "Facebook" to "Facebook" in this usage session, the transition interval is thus set to ∞ and then the probability would be 0.

The probability model could be estimated from a user's historical usage trace. In this section, we introduce an Apps Usage Graph (AUG) which models the transition probability among Apps for a single user. For training purposes, the implicit features for the training usage sessions are derived by traversing the AUG. However, for testing purposes, since we do not know which is the App to be invoked, the derivation of the implicit feature for the training usage session cannot be utilized directly. Therefore, an iterative refinement algorithm is proposed to estimate both the next App and its implicit feature simultaneously. The following paragraphs will illustrate the details of the AUG construction and the implicit feature derivation for both the training and testing usage sessions.

Apps Usage Graph (AUG)

For each user, we construct an Apps Usage Graph (AUG) to describe the transition probability among Apps. An AUG is a directed graph where each node is an App, the direction of an edge between two nodes represents the usage order, and the weight on each edge is a probability distribution of the interval between two Apps. Since two consecutive launches could be viewed as a Poisson arrival process, we can formulate the intervals between two launches as an exponential distribution. For example, Figure 2.3 shows the probability density function (PDF) of two consecutive launches which exactly fulfils the exponential distribution where most transitions (e.g., 0.45%) are within 1 minute.

Here, Equation 2.1 formulates the exponential density function of the launch interval being in $[x, x+1)$. The parameter $\alpha = \hat{p(0)}$ is derived by assigning $x = 0$ in Equation 2.1, and could be calculated by $p(0)$, the real probability derived from the training data. Then, β is solved by minimizing the difference between the estimated probability $\hat{p(i)}$ and the real probability $p(i)$ as shown in Equation 2.2 for every

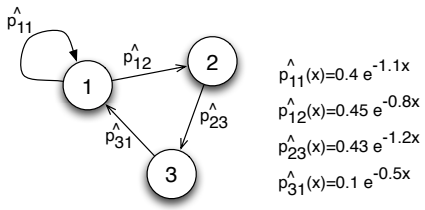


Figure 2.4: An example of the Apps Usage Graph (AUG).

interval i.

Empirically, we do not need to fit every interval when obtaining the exponential model. For example, in Figure 2.3, only the first 5 intervals already cover more than 75% of the training data. Therefore, we can iteratively add one interval until the data coverage reaches a given threshold. We will discuss the impact of the data coverage threshold in the experiments section.

$$\hat{p(x)} = \alpha \exp^{-\beta x} \tag{2.1}$$

$$\begin{aligned} \beta &= \underset{\beta}{\operatorname{argmin}} \sum_i |\hat{p(i)} - p(i)| \\ &= \underset{\beta}{\operatorname{argmin}} \sum_i |p(0) \exp^{-\beta i} - p(i)| \end{aligned} \tag{2.2}$$

For example, Figure 2.4 shows an AUG with three Apps. From Figure 2.4, the probability of two consecutive usages of App_1 with an interval of 0.3 minutes (i.e., $App_1 \xrightarrow{0.3} App_1$) is 0.4, and $App_1 \xrightarrow{1.5} App_2$ is 0.2. Although AUG only takes two consecutive Apps into account, such as p_{12} and p_{23}, the probability of p_{13}, could be calculated by $p_{12} \times p_{23}$.

Implicit Features for Training

For each training case, the implicit features are derived by looking up the AUG. Suppose the currently used App (i.e., class label) is App_t, the implicit feature is thus, $< p'_{1t}, p'_{2t}, ..., p'_{nt} >$, where p'_{it} represents the probability of transiting from App_i to any random Apps and then to App_t. The probability of $p'^{(s)}_{it}$ is defined as in Equation 2.3 which is the summation of every probability from App_i to App_t. Note that we use a superscript, s, to indicate how many Apps are between App_i and App_t, and App_{m_k} is the k-th App after App_i. Once we derive the implicit feature in a reverse time order, the sub-problem of estimating $p'^{(s-k)}_{m_k,t}$ is already solved. The calculation of the implicit feature for App_i stops when the transition probability falls below a given threshold, min_{tp}. In our collected dataset, the transition probability falls to 0.1% when we look backward to more than 5 Apps, which is the default parameter for min_{tp}. Algorithm 1 depicts the derivation of the implicit feature for a training case with App_t as its class label.

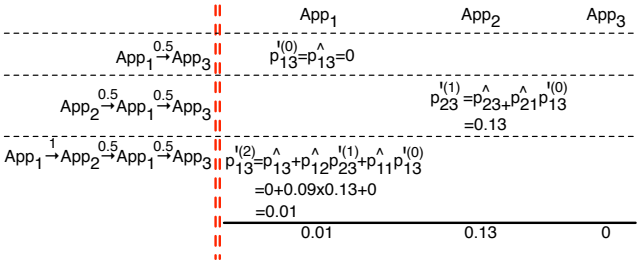


Figure 2.5: Steps of obtaining the implicit feature of App_3 in the training case, $\cdots \rightarrow App_1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3$.

$$p'^{(s)}_{it} = \hat{p_{it}} + \sum_k \hat{p_{i,m_k}} \times p'^{(s-k)}_{m_k,t} \tag{2.3}$$

Algorithm 1: Deriving the implicit feature of App_t for training.

Input: App_t: a training App
Output: IF_t: the implicit feature of App_t

```
foreach App_i prior than App_t do
    IF_t[i] ← IF_t[i] + p_it(δ̂_it) ;
    foreach App_m between App_i and App_t do
        IF_t[i] ← IF_t[i] + p_im(δ̂_jm) × IF_m[t] ;
    end
end
return IF_t
```

For example, suppose we have an AUG as shown in Figure 2.4 and a usage trace as $\cdots \rightarrow App_1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3 \rightarrow \ldots$. Figure 2.5 shows the process of obtaining the implicit feature of App_3. We first estimate $p'^{(0)}_{13}$ from $App_1 \xrightarrow{0.5} App_3$, then $p'^{(1)}_{23}$ from $App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3$, and finally update $p'^{(2)}_{13}$ from $App1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3$. Note that $p'^{(0)}_{13}$ is reused for calculating $p'^{(1)}_{23}$, and $p'^{(1)}_{23}$ and $p'^{(0)}_{13}$ are reused for updating $p'^{(2)}_{13}$. The implicit feature of App_3 is $< 0.01, 0.13, 0 >$.

Implicit Features for Testing

Since the App to be predicted for current invocation, App_t, is unknown for testing, the derivation process of implicit features for training does not work. We propose an iterative refinement algorithm to estimate both App_t and its implicit feature, IF_t, for testing. Suppose θ_i is the probability of $App_t = App_i$, the implicit feature IF_t is calculated as in Equation 2.4 which is a linear combination of the IF of each App_i. In addition, $M = [IF_1^T, IF_2^T, \ldots]$ represents the transition matrix among Apps, where IF_1^T, IF_2^T, ... are column vectors. Then, the value of θ_i could be updated by Equation 2.5, which is the probability of staying in App_i after one-step walking along the transition matrix M. We keep updating θ_i and IF_t iteratively, until App_t is fixed to one specific App. In our experiments, the iterative refinement process converges in about 3 iterations. Algorithm 2 depicts the derivation of the implicit feature for testing.

$$IF_t = \sum_{App_i} \theta_i \times IF_i \tag{2.4}$$

$$\theta_i = \sum_{App_m} IF_t[m] \times M[m][i] \tag{2.5}$$

Algorithm 2: Deriving the implicit feature for testing.

Input: t: a testing case
Output: IF_t: the implicit feature at t

```
while iter < threshold do
    foreach θ_j do
        IF_t ← IF_t + θ_i × IF_i ;
    end
    foreach App_i prior than time t do
        θ_i ← θ_i + IF_t[m] × M[m][i] ;
        Normalize θ_i ;
    end
    iter ← iter + 1 ;
end
return IF_t
```

For example, suppose the testing case is $\cdots \rightarrow App_1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_t$. First, we initialize θ_i as $< 1/3, 1/3, 1/3 >$, which gives equal probability to each App, and the transition matrix $M = \begin{bmatrix} 0.49 & 0.6 & 0.01 \\ 0 & 0 & 0.13 \\ 0 & 0 & 0 \end{bmatrix}$, which is derived by calculating the IF of each App shown in Equation 2.3. Note that the last row is all zero because there is no App_3 transiting to any other Apps. Then, the implicit feature is $< 0.37, 0.04, 0 >$ in the first iteration. Next, θ_i is updated to $< 0.18, 0.22, 0.01 >$, and normalized as $< 0.44, 0.54, 0.02 >$ according to one-step walk in M with the calculated implicit feature as the prior probability. Then, we can obtain the implicit feature as $< 0.53, 0.01, 0 >$ in the second iteration.

2.4 Personalized Feature Selection

The goal of the personalized feature selection is to use as fewer features as possible to guarantee an acceptable accuracy. Due to the energy and storage consumption of collecting sensors readings and Apps transition relations, we should select useful features for different users in advance. Furthermore, through the personalized feature selection, we could avoid the curse of dimensionality on performing the kNN. We first apply the personalized feature selection on the training data, and then only the selected features are required to be collected in the future.

Here, we propose a greedy algorithm to select the best feature iteratively. We adopt the concept of Minimum Description Length (MDL) [59, 60] to evaluate the goodness of the features. For different

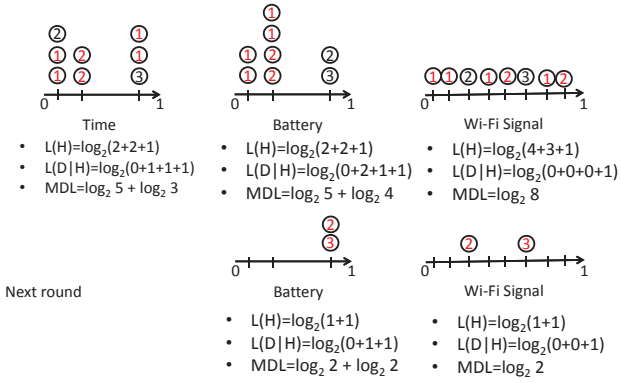


Figure 2.6: An example of feature selection where the red data points are correctly predicted.

features, we can have varied projections of the training data. We claim that if a feature needs fewer bits to describe its data distribution, it is good for predicting the data. Therefore, in each iteration, the feature with the minimum description length is selected. Then, those data points which are correctly predicted are logically eliminated from the training data, and the next feature is selected by the same process repeatedly. We define the description length of the hypothesis, which is shown in Equation 2.6, as the length of representing the training data. $NG(App_i)$ is the number of groups of App_i. The description length of Data given the hypothesis is the total number of miss-classified data which is formulated as in Equation 2.7.

$$L(H) = \sum_i \log_2 NG(App_i) \tag{2.6}$$

$$L(D|H) = \sum_i \log_2(missClassified(App_i) + 1) \tag{2.7}$$

For example, given 8 data points in the training data and three features as shown in Figure 2.6. In the first round, Time is the feature with minimum description length. Those data points marked as red are correctly predicted and will be removed. Therefore, in the second round, only two data points are left, and the feature of Wi-Fi signal is selected due to its minimum description length.

The selection process stops when a percentage of ρ of the training data is covered. We also discuss the impact of ρ in the experimental section. Note that the number of features affects the energy and storage consumption and is set according to the capability of the smartphones. Algorithm 3 depicts the process of personalized feature selection. After the selection, only the readings of the sensors which are selected will be collected as the explicit feature in the future. In addition, only the selected Apps will be used to construct AUG.

Algorithm 3: Personalized feature selection.

Input: D_z: the training data
Output: PF: the personalized features

Let $N_z \leftarrow |D_z|$;
while $|D_z| < \rho N_z$ **do**
  foreach *feature* f **do**
    Calculate DL_f: description length for feature f ;
  end
  $PF \leftarrow PF \cup \{\underset{f}{\text{argmax}}\, DL_f\}$;
  Let D_a be the set of accurately predicted data points ;
  $D_z \leftarrow D_z - D_a$;
end
return PF

2.5 Experimental study

In this section, we conduct a comprehensive set of experiments to compare the performance of the proposed KAP framework with other existing methods including 1) most frequently used (MFU) method, 2) most recently used (MRU) method which is the built-in prediction method in most mobile OS, such as Android and iOS, 3) SVM, 4) App Naive Bayes [64], 5) Decision Tree, and 6) AdaBoost. In the following, we first discuss the collected dataset, then introduce the metrics employed to evaluate the performance, and finally deliver the experimental results.

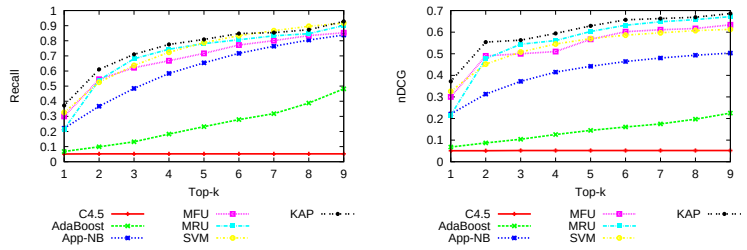
2.5.1 Dataset Description

In this chapter, we use a real world dataset collected by a mobile phone company which installed a monitoring program on every volunteer's smartphone. In this dataset, we have totally 50 volunteers including college students and faculty from June 2010 to January 2011. For each user, we separate the dataset into three parts, where each part consists of three months, and we use the first two months as training data, and the last one month as testing data. Totally, there are more than 300 different Apps installed on their smartphones, and the average number of Apps on one smartphone is 56.

2.5.2 Performance Metrics

In this chapter, we use two performance metrics: 1) average recall and 2) nDCG [29] score.

Average Recall: Since there is only one App being launched in each testing case, recall score is thus adopted as one performance metric which evaluates whether the used App is in the prediction list. The recall score of one user is defined as $\sum_{c_i \in C} \frac{I(App_{c_i}, L_{c_i})}{|C|}$, where C is the set of testing cases, App_{c_i} is the ground-truth, and Lc_i is the prediction list at the i-th testing case. $I(\cdot)$ is an indicator function which equals 1, when $App_{c_i} \in L_{c_i}$, and equals 0, otherwise. Finally, the average recall is the average of the recall values of all users.



(a) Recall (b) nDCG

Figure 2.7: Impact of the number of prediction, k.

nDCG Score: To evaluate the accuracy of the order of the prediction list, we also test the nDCG score of the prediction results. The IDCG score is fixed to 1 because there is only one used App in the ground-truth. The DCG score is $\frac{1}{\log_2(i+1)}$ when the used App is predicted at position i of the prediction list. Then, nDCG is the average of $\frac{DCG}{IDCG}$ for all testing cases.

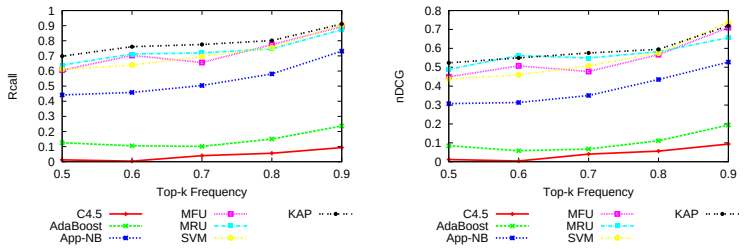
2.5.3 Experimental Results

To evaluate the performance of predicting Apps usage by the proposed KAP framework, we first evaluate the overall performance when predicting different numbers of Apps. Then, we test the performance of the personalized feature selection algorithm. The impact of different parameters for the KAP framework and kNN classification is also included. Note that we use top-k = 4, kNN=40%, and the minimum data coverage of personalized feature selection as 70% to be the default parameter settings throughout the experiment.

Overall Performance

First, we evaluate the performance KAP and other different methods under various numbers of prediction, k. As can be seen in Figure 2.7, when the number of prediction k increases, both the recall and nDCG values also increase. However, KAP, MRU, MFU, and SVM perform better than others. In Figure 2.7(a), when $k = 9$ (the number of predictions shown in the latest Android system), the recall of KAP could be more than 95%, while it is only about 90% for MFU, MRU, and SVM. On the other hand, the nDCG value of KAP shown in Figure 2.7(b) is always higher than that of the other methods, which means the prediction order of KAP is better.

Second, we test the accuracy of varied top-k frequency. The top-k frequency is defined as the ratio of the usage of the most frequent k Apps. For example, if a user has 5 Apps and the usage counts are 3, 1, 2, 5, and 2, the top-2 frequency is thus $\frac{5+3}{3+1+2+5+2} = \frac{8}{13}$. Figure 2.8 shows the results when top-k = 4. Intuitively, when the top-k frequency increases, the accuracy of the MFU method could be better. However, in Figure 2.8(a), even when the ratio is 0.9, the MFU method performs just slightly better than the MRU method, but worse than both KAP and SVM. In Figure 2.8(b), the prediction



(a) Recall (b) nDCG

Figure 2.8: Impact of top-k frequency.

Table 2.2: The storage consumption and accuracy under varied data coverage ρ.

Coverage(%)	30	40	50	60	70	80	90	100
Storage(KB)	28	31	34	37	43	52	82	94
Recall	0.78	0.78	0.80	0.80	0.82	0.82	0.82	0.83
nDCG	0.50	0.51	0.52	0.53	0.55	0.57	0.57	0.58

order of KAP is also better than the results of the other methods.

Impact of Personalized Feature Selection

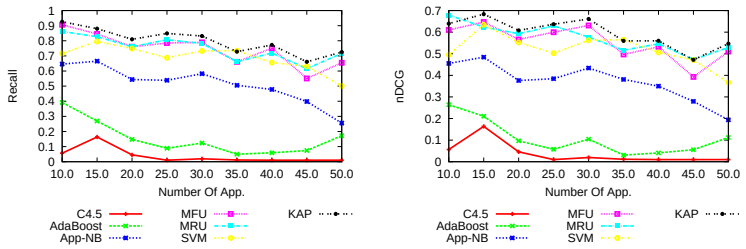
For the proposed KAP method, we evaluate the performance of the personalized feature selection to see if the proposed MDL-based selection algorithm could reduce the used storage when maintaining a good prediction accuracy. For one user, the average used storage and prediction accuracy is shown in Table 2.2 under different data coverage ρ. As can be seen in Table 2.2 the personalized feature selection could reduce 55% of training data size and only lose 1% of recall and 3% of nDCG when the data coverage is 70%. In addition, Table 2.3 compares the execution time of KAP with and without the personalized feature selection, where the training time is reduced dramatically under $\rho = 70\%$.

2.5.4 Comparison of Different Usage Behavior

Since different users have different usage behaivor, which could extremely affect the prediction accuracy. In this section, we separate users into different groups according to 1) number of installed Apps, 2) usage frequency, and 3) usage entropy. Then, we test the performance of applying different methods on different groups.

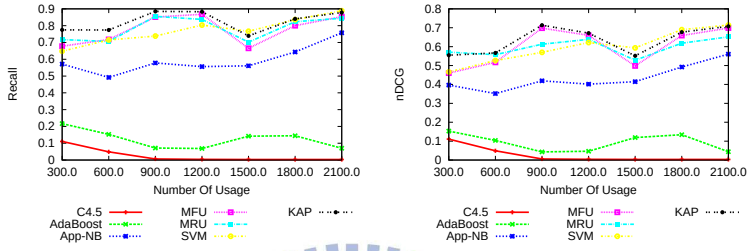
Table 2.3: The execution time of KAP with and without personalized feature selection.

Execution time (ms)	Training	Testing	Total
KAP	86	160	246
KAP without selection	185	160	345



(a) Recall (b) nDCG

Figure 2.9: Impact of the number of Apps.



(a) Recall (b) nDCG

Figure 2.10: Impact of the usage count.

Impact of the Number of Installed Apps

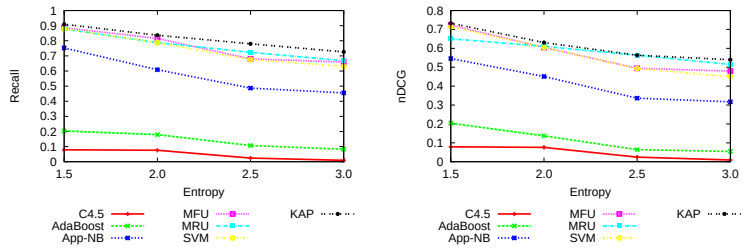
When users launch more Apps, it becomes more difficult to accurately predict Apps usage. Figure 2.9 shows the recall and ndcg results for a varying number of used Apps. As can be seen in Figure 2.9, both the recall and ndcg values decrease when the number of used Apps increases for all methods. However, the decreasing rate of the proposed KAP method is much smoother than that of the others. The recall of KAP is around 85% while that of the others is below 40% when the number of used Apps is 30.

Impact of the Usage Count

Now, we test the impact of the usage count. A higher usage count means we could have more training data to learn the classification model for App prediction. Concurrently, it provides more complicated information of users' usage behavior, and could make noisy data. Figure 2.10 shows the recall and ndcg values. The performance of KAP, Naive Bayes, and SVM goes up when the usage count increases. However, AdaBoost and Decision Tree have worse performance as the usage count goes up. The result shows that the KAP algorithm can handle more complicated and noisy data.

Impact of the Entropy of the Apps Usage

We evaluate the impact of the entropy of the Apps usage. Intuitively, as the entropy of the Apps usage becomes larger, the Apps usage is almost random, and the performance of Apps usage prediction would become worse. Figure 2.11 depicts that the proposed KAP could have around 50% accuracy when the entropy goes to 3 where the other methods only have accuracy of less than 40%.



(a) Recall (b) nDCG

Figure 2.11: Impact of the entropy of Apps.

Table 2.4: The recall and nDCG values under varied numbers of iterations.

#Iterations	1	2	3	4	5
Recall	0.67	0.79	0.79	0.80	0.81
nDCG	0.43	0.59	0.59	0.60	0.61

2.5.5 Impact of Different Parameters

Number of Iterations for Implicit Feature Extraction

First, we test the number of iterations of deriving the implicit feature for each testing case. As shown in Table 2.4, the accuracy stays almost the same after the second iteration. This indicates that the iterative refinement algorithm could converge within 2 iteration which is sufficient to estimate the implicit feature.

Minimum Probability for Identifying Usage Sessions

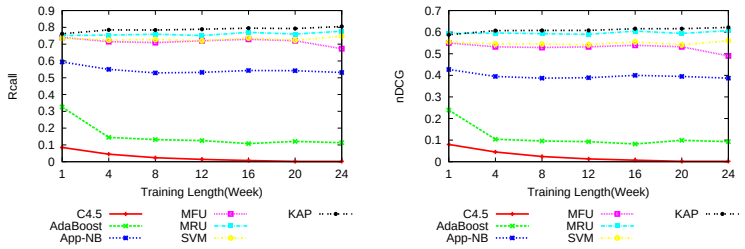
As users usage sessions could be varied according to different tasks, we only need the useful length of the usage sessions to perform accurate Apps usage prediction, instead of calculate the full usage sessions. Therefore, we conduct this experiment to evaluate the impact of the length of usage sessions. Ac can be seen in Table 2.5, the results are not affected by the minimum transition probability, min_{tp}, too much. From our collected data, the session length is around 2 when min_{tp} is 0.5, and the best case is under $min_{tp} = 0.1$, which has the session length as around 5.

Parameters for kNN Classification

There are two main parameters affecting the accuracy of kNN classification: 1) the length of the training period, and 2) the number of neighbors of kNN. Here, we fix the number of predictions to 4 Apps and compare the recall and nDCG values of KAP and the other methods. Figures 2.12(a) and 2.12(b) show the results, where the recall and nDCG values of the KAP, MFU, MRU, and SVM methods almost keep

Table 2.5: The recall and nDCG values under varied minimum probability for session identification.

min_{tp}	0.5	0.25	0.1	0.075	0.05	0.025	0.001
Recall	0.73	0.77	0.83	0.81	0.80	0.75	0.74
nDCG	0.53	0.57	0.61	0.58	0.55	0.53	0.52



(a) Recall (b) nDCG

Figure 2.12: Impact of training length.

Table 2.6: The recall and nDCG values under varied number of neighbors for kNN.

kNN(%)	20	40	60	80	100
Recall	0.74	0.79	0.80	0.80	0.81
nDCG	0.55	0.61	0.63	0.63	0.64

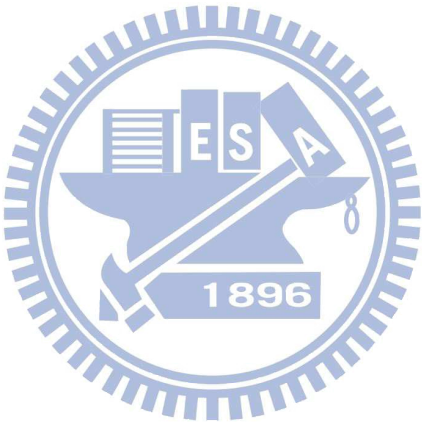
the same performance under varied training lengths. Therefore, we suggest that we just need to collect a short period as training data to predict users' Apps usage, since users' behavior is considered as stable over a short period.

Finally, we evaluate the impact of selecting different numbers of neighbors to perform kNN classification. Because the training data of different users could vary from several hundreds to thousands. we use a relative value for the number of neighbors. Table 2.6 shows the results of the recall and nDCG values for different number of neighbors. As can be seen in Table 2.6, even only select 40% of training data as the neighbors, the recall value is almost 80%. Therefore, we set the default number of neighbor as 40% throughout the experiments.

2.6 Conclusion

In this chapter, we propose an Apps usage prediction framework, KAP, which predicts Apps usage regarding both the explicit readings of mobile sensors and the implicit transition relation among Apps. For the explicit feature, we consider three different types of mobile sensors: 1) device sensors, 2) environmental sensors, and 3) personal sensors. For the implicit features, we construct an Apps Usage Graph (AUG) to model the transition probability among Apps. Then, for each training datum, we could represent the next used App as the implicit feature which describes the probability of transition from other Apps. Note that, since the next App in the testing data is unknown, we propose an iterative refinement algorithm to estimate both the probability of the App to be invoked next and its implicit feature. We claim that different usage behaviors are correlated to different types of features. Therefore, a personalized feature selection algorithm is proposed, where for each user, only the most relative features are selected. Through the feature selection, we can reduce the dimensionality of the feature space and the energy/storage consumption.

We integrate the explicit and implicit features as the feature space and the next used App as the class label to perform kNN classification. In the experimental results, our method outperforms the state-or-the-art methods and the currently used methods in most mobile devices. In addition, the proposed personalized feature selection algorithm could maintain better performance than using all features. We also evaluate the performance of KAP for different types of users, and the results show that KAP is both adaptive and flexible.



Chapter 3

Mining Temporal Profiles of Mobile Applications for Usage Prediction

3.1 Introduction

Smart phones have became an important smart device in people's daily life. We use them to communicate with friends, check emails, take pictures, and play games. Concurrently, we can install many kinds of mobile applications (abbreviated as Apps) in our smart phone and invoke them for individual purposes. However, according to our observation, the average number of Apps of each device is about 70 to 80 and that of some devices which even exceed 150. We realize that as the number of Apps in a user's smart phone increases, users will spend an increasing amount of time looking for and launching the Apps they want to use.

To deal with this problem, we have designed an AppNow widget which can dynamically predict users' App usage through mining temporal profiles from the users' previous usage behavior. For example, Figure 3.1 shows different prediction results at different times in one day. In Figure 3.1(a), the time is 9:00 a.m. and AppNow shows that the user is intending to start work by checking calender, emails, and so on. In Figure 3.1(b), the time goes to 12:30 p.m. and AppNow indicates that the user is about to communicate with friends using social network services. In Figure 3.1(c), the user is likely to play games at home when the time is 8:30 p.m. However, there are two challenges when designing the AppNow widget: 1) connecting the relation between Apps and their launched times, and 2) calculating the usage probability through comparing the App launched times and current time. First, Apps are not always launched at the same time. For example, if a user checks Facebook approximately once every one and a half hours, the usage time could be around 9:00, 10:30, 12:00, and so on. Therefore, to connect the relation between time and Apps usage, we proposed a temporal profile to summarize the usage history of each App. Second, since the launched times of an App may not exactly match the current time, we have

(a) Morning (b) Noon (c) Night

Figure 3.1: An example of the AppNow widget on a smart phone.

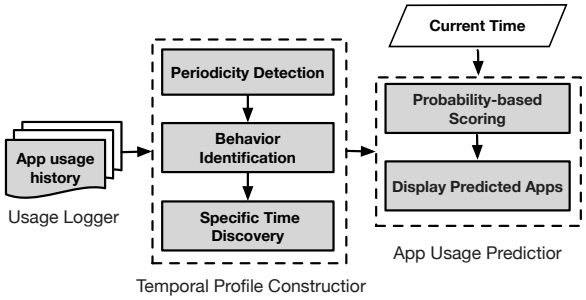


Figure 3.2: The system flow of AppNow.

to model the usage probability over the time different between the launched times and current time.

To the best of the authors' knowledge, although there are many research works focusing on smart phones [66, 32, 53, 67, 77, 57, 58, 81], there are no existing works that explore predicting usage behavior, let alone developing a widget on smart phones. On the other hand, current prediction algorithms on location, purchasing, and co-authoring [28, 30, 51, 12] do not create the relation with the aspect of time, such that they cannot be applied to solve the novel problem of predicting the App usages.

3.1.1 System Framework

The system flow of the AppNow widget is shown in Figure 3.2, where AppNow possesses three main components, a usage logger, a temporal profile constructor and an App usage predictor. The usage logger records the launched time and App ID on every App launch. The temporal profile constructor builds a temporal profile for each App. We summarize and investigate the usage history for each App into a set of (period, {specific time}) tuples which is, therefore, called a temporal profile for that App. The usage predictor calculates the probability of using each App at the current time. The AppNow widget then shows the 4 Apps with the highest probability.

3.1.2 Demonstration of the System

The AppNow widget is developed on Android based smart phones. Users do not need to set any parameters. The system automatically logs the App usage behavior and updates the temporal profiles. When the widget becomes active (i.e. shown on the screen), the predictor updates the displayed four Apps regarding current time. Users can execute the Apps by directly touching the App icon in the AppNow widget. However, we neither move nor re-organize the placement of Apps in users' smart devices, so when the AppNow widget cannot provide the correct Apps, the user can still find them in their original place.

3.2 Design and Implementation

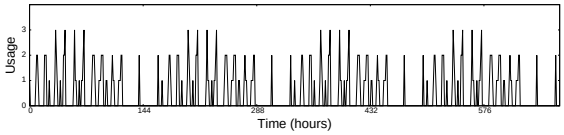
3.2.1 Mining Temporal Profiles

A temporal profile is a set of (p, T) tuples, where p is the period of usage and T is the set of usage specific times in the periodicity p. The discovery of temporal profiles consists of three steps: periodicity detection, behavior identification and specific time discovery.

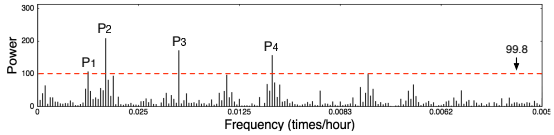
First, we detect the periodicities of Apps by the idea proposed in [69]. For each App, we adopt Discrete Fourier Transform (DFT) to find the power spectral density (PSD). The App usage history is represented as $AH(app) =< a_1, a_2, ..., a_N >$, where a_i represents the number of launches at time unit i. For example, Figure 3.3(a) plots the usage chart of one App usage history for 4 weeks, where the length of one time unit is 1 hour, and therefore, N is 672. In addition, Figure 3.3(b) depicts the periodogram after applying DFT to Figure 3.3(a). In Figure 3.3(b), the dashed line is an automatically adjusted threshold which is obtained by using the dynamic cut approach [69]. The main concept of dynamic cut is that the power of a significant frequency should be higher than the maximum power derived from a random sequence. The random sequence is generated by shuffling the original $AH(app)$ and we claim that there should be no significant frequency in the random sequence. Finally, autocorrelation is adopted to verify the periodicities by a more accurate estimation. In Figure 3.3(c), we map the frequency to period, and we can see that the mapped period P_2 corresponds to 24 hours in this case.

Second, after the periodicity detection step, for an App with its periodicity set $\{p_1, p_2, \dots\}$, we further identify multiple behaviors for an individual periodicity, p_i. Since different behaviors may share the same periodicity, we separate them in this step. Here, we perform a hierarchical clustering to identify the user's multiple behaviors. Figure 3.4 shows an example of behavior identification, where we first decompose the App-history into several pieces according to the derived periodicity, and then we utilize EDR [9] to calculate the distance of two pieces. A hierarchical clustering would separate the pieces into different groups which are viewed as the user's multiple behaviors. In Figure 3.4, we observed that the two groups belong to weekday behavior and weekend behavior respectively.

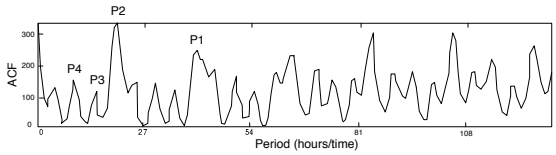
Since the exact usage time would be slightly shifted, in the third step, the specific times of each



(a) Usage history



(b) Power spectral density



(c) Discovered periods

Figure 3.3: An example of periodicity detection.

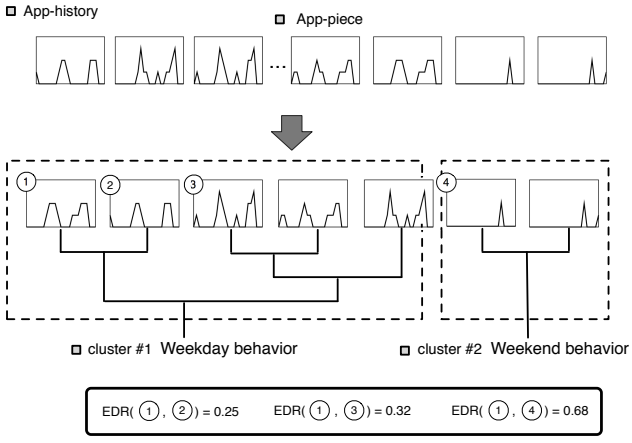
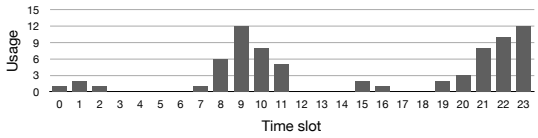
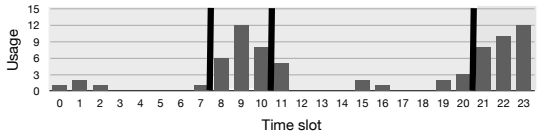


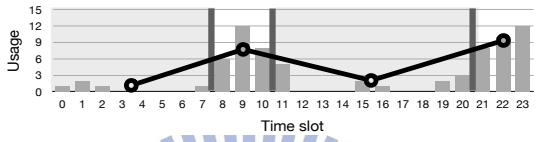
Figure 3.4: An example of behavior identification.



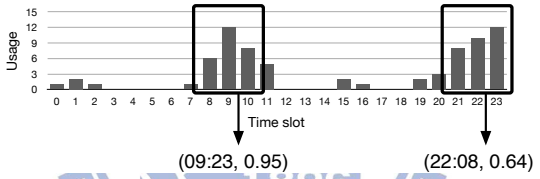
(a) Accumulated usage count



(b) Partitioning



(c) Discover local maximum



(d) Calculate mean and variance

Figure 3.5: An example of specific times discovery.

group identified in the previous step is composed by the mean and variance of previous usages. In Figure 3.5(a), we take an example by plotting the accumulated usage in 24 hours which is the periodicity P_2 in periodicity detection. We first separate the temporal space into partitions such that the variance of usage in each part could be minimized. Intuitively, the partitions could be derived by a greedy algorithm. In Figure 3.5(b), the partitions are [0,8], [8,11], [11,21], and [21,24]. Then, we calculate the usage of each part and derive their local maximums to be the usage times. As shown in Figure 3.5(c), [8,11] and [21,24] are local maximums. Finally, the specific times are the means of usage in the local maximums as depicted in Figure 3.5(d). We use a tuple of $(mean, variance)$ to represent each specific time. Eventually, $(24, \{(09:23, 0.95), (22:08, 0.64)\})$ is added to the App's temporal profile.

3.2.2 Apps Usage Prediction

By deriving the App temporal profile which consists of the usage periods and the corresponding specific times, we can predict the possible App usage by calculating the usage probability of each App. We propose a probability-based scoring model, which is based on Chebyshev's inequality from probability theory [52],

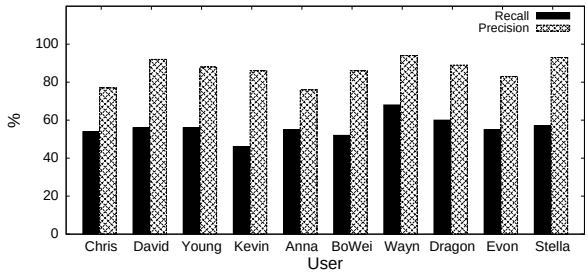


Figure 3.6: Precision and recall of specific times discovery.

to formulate the usage probability of each App. Equation 3.1 shows the Chebyshey's inequality. It shows the probability of that the time difference between the current time and the specific time is not less than λ. Therefore, we can use Equation 3.2 to calculate the score, where CTime is current time, STime is the specific time, and $Var[STime]$ is the variance of $STime$.

$$P(|x - E[x]| \geq \lambda) \leq \frac{Var[x]}{\lambda^2} \tag{3.1}$$

Finally, the probability of launching an App is the maximum score from all specific times in its temporal profile, which is formulated as $MAX(Score(CurrentTime, SpecificTime_i))$, for each specific time in the temporal profile.

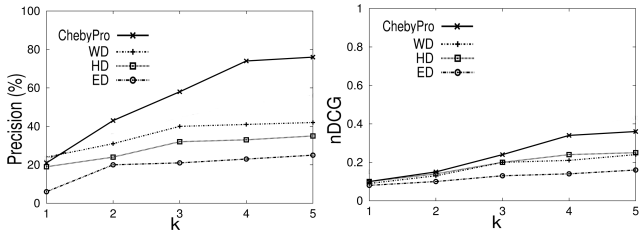
$$Score(CTime, STime) = \frac{Var[STime]}{|CTime - STime|^2} \tag{3.2}$$

3.3 Experimental Evaluation

We installed a monitoring logger in 10 smart phones to collect the usage traces from July to December 2012. The dataset was separated into three parts: July to August, September to October, and November to December. For each part, the first month is the training date while the second month is the testing data. The overall performance is, therefore, the average performances of the three parts.

Based on the collected traces, we first evaluate the correctness of the discovered temporal profiles. We check if Apps are launched at the specific times indicated in their temporal profile. Figure 3.6 shows the precision and recall results for each user. As can be seen in Figure 3.6, the average precision is 86.4% and the average recall is 55.9% when the AppNow widget shows 4 Apps.

Then, we compare the accuracy of usage prediction among AppNow and three frequency-based baseline methods, WD, HD, and ED. Figures 3.7(a) and 3.7(b) depict the precision and nDCG score with different k (the number of Apps shown in AppNow). The baseline methods select the top-k most frequently used Apps in different time intervals. The interval of WD is a whole day (24 hours), HD is half



(a) Precision (b) nDCG

Figure 3.7: The comparison of different prediction approaches.

a day (12 hours) and ED is 8 hours. As can be seen in Figure 3.7, our proposed AppNow outperforms the three frequency-based methods in terms of both precision and nDCG score.

3.4 Conclusions and Future Work

We develop an AppNow widget to predict App usage from mining the App temporal profile. The temporal profile summarizes and investigates the usage history of an App. When the AppNow widget is activated, it calculates the usage probability for each App through a proposed probability-based scoring model. We collected real usage traces from 10 smart phones for 6 months. We evaluated the accuracy of both the temporal profiles and the App prediction. The results show that AppNow outperforms three frequency-based methods. In addition to predicting App usage, AppNow can also help recommend new Apps for users, since the predicted Apps can reflect the user's semantic activity. For example, when the predicted Apps are composed of the Apps related to games, we can infer that the semantic activity of the user is gaming and can recommend other games for that user.

Chapter 4

Mining Usage Traces of Mobile Apps for Dynamic Preference Prediction

4.1 Introduction

As mobile devices become more and more popular, a tremendous amount of mobile applications (abbreviated as Apps) are designed for varied functions and purposes. Users can download and execute Apps in their mobile devices to satisfy their needs and affinities. For App providers, to understand users' preferences is quite important to recommend new Apps, and to decide their marketing strategies for selling Apps [16, 47, 23]. Although users can rate the Apps they have experienced, only a small percentage of users rate their Apps. For example, the famous App, Angry Birds, only received 4% ratings from 1.3 million of downloads [72]. Besides, users may not be willing to consistently rate the Apps when they change their preferences. On the other hand, although [63] states an Apps recommendation problem, it does not show the dynamic preferences of users. By contrast, through the dynamic preferences, we can not only recommend Apps but investigate more tasks on Apps.

In this chapter, we aim to predict users' dynamic preferences of each App and further quantize the preferences to real numbers such that we can compare the preferences among different users. As users repeatedly invoke these Apps, their preferences are dynamic over time based on what they have experienced. Here, we claim that a user's dynamic preference is related to the usage trace (i.e., series of usage counts). For example, Fig. 4.1 shows three usage traces of Calender, Browser, and Messenger, for a certain user. As can be seen in Fig. 4.1, the number of usages on "Messenger" apparently drops down after 14 days (two weeks). Therefore, we can infer that the user decline his/her preference on "Messenger" in 14 days by either implicit or explicit reasons.

Nevertheless, usage counts of Apps are not directly related to the preferences of Apps. For example, in Fig. 4.1, although the usage count of Messenger is higher than the other two Apps, the preference of

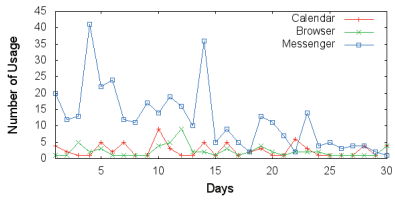


Figure 4.1: The number of usages of different mobile applications.

Messenger is not necessarily higher than the other two Apps. Probably, Messenger is a communication tool, which is designed to be used frequently. As for Calendar, users will not frequently check their calendar all the time. In our experimental results, we also show that the usage-based algorithm cannot predict anything, where its accuracy is often close to zero. To correctly predict preferences of Apps from the usage traces, we propose two methods, Mode-based Prediction (abbreviated as MBP) and Reference-based Prediction (abbreviated as RBP). Both methods utilize different strategies to avoid the impact of the inherent magnitude bias of the the usage counts. MBP adopts only the usage mode of Apps: active mode for using the App while inactive mode for not using the App. RBP refers to the previous usage counts of each App as a reference history and thus, the usage count becomes a relative value of the reference history.

Both MBP and RBP consist of two phases: the trend detection phase and the change estimation phase. The first phase determines whether the preference is decreasing or increasing. The second phase estimates the absolute value on the preference change. For MBP, we increase the preferences of those Apps which are used at current time unit, but decreases the preferences of others. Then, we propose a utility model: when a user uses more Apps at the same time unit, each App would receive less preference increment. According to the utility model, we can calculate the increment and decrement of each App. For RBP, it calculates an expected number of usage for each App at current time unit by solving an optimization problem where the expected number of usage can keep the trend of preference change staying static. If the actual number of usage is larger (smaller, respectively) than the expectation, the preference will increase (decrease, respectively). Then, RBP uses a probabilistic model to estimate the change of preferences.

The contributions of this study are:

1. We explore usage traces of Apps for dynamically predicting the perferences of Apps.

2. We analyze the characteristics of Apps, and propose two algorithms, MBP and RBP, to predict preferences of Apps.

3. In the MBP method, we derive the dynamic preferences according to only the usage mode and propose a utility model to calculate the change of users' preferences.

4. In the RBP method, by solving an optimization problem, the expected number of usage is derived

as a reference, and a probabilistic model is constructed to estimate the users' preferences.

5. We conduct a comprehensive performance evaluation. The experimental results show that the predicted dynamic preferences of both MBP and RBP can better reflect users' behavior

4.2 Related Work

To the best of our knowledge, this chapter is the first work discussing dynamic preferences prediction problem. Although there are many research works discussing the problem of predicting users' preference, they only focused on a static environment. In a static circumstance, such as renting movies and purchasing books, users generally only act on them once and the preference remain static. Therefore, they can use the existing user preferences to predict the unknown preferences through the attributes of items [18]. The attributes could be the metadata, such as artist, genre, etc., or the ratings the item already had. Although [18] focused on predicting the ratings of musics, they still treated the music ratings as static preferences. This is because their focus is on purchasing songs or CDs, not on the preference to listening to a song from a user collection at a particular moment. Only the authors in [34, 40, 20] recognized the temporal dynamics of users' preferences. Nevertheless, [34] still need to obtain at least a portion of static ratings as training data. [40, 20] only consider the evolution of users' behavior, instead of quantize their preferences. For predicting preferences of Apps, users can use Apps repeatedly; therefore, their preference changes over time, and even be impacted by new Apps [37]. Consequently, the traditional preference prediction methods cannot be adopted for the dynamic preference problem, because 1) the traditional methods all need to obtain at least a portion of static preferences as training data, and 2) the static preferences are out-of-date when we perform the prediction in a dynamic environment.

4.3 Preliminary

In this section, we first describe the symbols used in this chapter. Explicitly, we use r_{min} and r_{max} to represent the minimum and maximum value of users' preference. Thus, the preference at time unit t is a real number $r_{ij}^{(t)} \in [r_{min}, r_{max}]$, which represents the preference of user i on App j. To facilitate the presentation of this chapter, U is the set of users and I is the set of Apps. A dynamic preference matrix is used to represent the preferences of Apps at a certain time unit. Here, we divide time space into time units, and use l, an application dependent parameter, to represent the length of a time unit. The formal definition is below:

Definition 1. ***(Dynamic Preference Matrix)*** *A dynamic preference matrix at time unit t, $R^{(t)}$, is a $|U| \times |I|$ matrix, where $r_{ij}^{(t)} \in [r_{min}, r_{max}]$, for each $r_{ij}^{(t)}$.*

A usage count matrix constructed from users' traces is defined in Definition 2

Definition 2. ***(Usage Count Matrix)*** *A usage count matrix at time unit t, $C^{(t)}$, is a $|U| \times |I|$ matrix, where each element $c_{ij}^{(t)}$ represents how many times user i used App j at time unit t.*

We use a change matrix to record the preference change of each user-App pair. When the value is positive (negative, respectively), the preference is increasing (decreasing, respectively). Definition 3 shows the detail of change matrix. In this chapter, the change matrix is derived from both usage count matrix and dynamic preference matrix.

Definition 3. ***(Change Matrix)*** *A change matrix at time unit t, denoted as $\Delta^{(t)}$, is a $|U| \times |I|$ matrix, where the value of each element $\delta_{ij}^{(t)}$ is in either [0,$r_{max} - r_{ij}^{(t-1)}$] for positive value, or in [$r_{ij}^{(t-1)} - r_{min}$,0] for negative value.*

We claim that the preference of an App would not change dramatically. Even when users do not use an App for a long time, the preference of it would decay smoothly over time. Therefore, we derive users' preferences according to the previous preferences and current usage behavior as described in Definition 4.

Definition 4. ***(Dynamic Preferences Prediction Problem)*** *Let $R^{(t-1)}$ be the dynamic preference matrix at time unit $t-1$, and $C^{(t)}$ be the usage count matrix at time unit t, the dynamic preference prediction problem is 1) calculating the change matrix, $\Delta^{(t)}$, and 2) deriving $R^{(t)}$ according to Eq. 4.1.*

$$R^{(t)} = R^{(t-1)} + \Delta^{(t)} \tag{4.1}$$

For example, suppose we have two users and three Apps, and the system parameters are $r_{min} = 0$ and $r_{max} = 5$. Let $R^{(t-1)} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$ be the dynamic preference matrix derived at time unit $t-1$, and $C^{(t)} = \begin{bmatrix} 100 & 2 & 30 \\ 2 & 300 & 40 \end{bmatrix}$ be the usage count matrix. First, we calculate the change matrix according to $C^{(t)}$ and $R^{(t-1)}$, such that, in this example, the values of the change matrix are related to the usage counts and will be in the defined range to avoid the values in $R^{(t)}$ being out of range. Assume that we can obtain $\Delta^{(t)} = \begin{bmatrix} 4 & -1 & 1 \\ -2 & 2 & 1 \end{bmatrix}$. Then, the new dynamic preference could be derived as $R^{(t)} = \begin{bmatrix} 5 & 1 & 4 \\ 0 & 5 & 5 \end{bmatrix}$.

4.4 Dynamic Preference Prediction

As described in Definition 4, to obtain the dynamic preference matrix, $R^{(t)}$, we need to know the change matrix, $\Delta^{(t)}$, in advance. Here, we use δ_{ij}^{t} to represent the elements in $\Delta^{(t)}$. Empirically, we can calculate δ_{ij}^{t} by Eq. 4.2 which consists of two parts: 1) $m \in \{0, 1\}$ which indicates whether δ_{ij}^{t} is positive ($m = 0$) or negative ($m = 1$), and 2) $v_{ij}^{(t)} > 0$ which is the absolute value of δ_{ij}^{t}. Through this equation, we can

calculate the change matrix, $\Delta^{(t)}$, by finding a proper pair of m and v_{ij}^t for each $\delta_{ij}^{(t)}$.

$$\delta_{ij}^{(t)} = (-1)^m \times v_{ij}^{(t)} \tag{4.2}$$

In this chapter, we design a two-phase framework: the trend detection phase for the value of m and the change estimation phase to calculate $v_{ij}^{(t)}$. In order to smooth the preference change, the value of $v_{ij}^{(t)}$ depends on not only the current usage count, $c_{ij}^{(t)}$ but the previous preference, $r_{ij}^{(t-1)}$. In addition, when the preference is increasing (respectively, decreasing), the value of $v_{ij}^{(t)}$ is in the range of $[0, r_{max} - r_{ij}^{(t-1)}]$ (respectively, $[0, r_{ij}^{(t-1)} - r_{min}]$). Thus, we can formulate $v_{ij}^{(t)}$ as in Eq. 4.3, where $u_{ij}^{(t)}$ is a utility parameter determined by user's preference change. Explicitly, when $u_{ij}^{(t)}$ is larger (i.e. user's preference change is large), $v_{ij}^{(t)}$ would be larger.

In order to address the challenge related to the number of usages of Apps, we propose two algorithms based on different points of view. The first one is Mode-based Prediction (MBP) which takes into account of the binary usage mode of active and inactive. The second one is called Reference-based Prediction (RBP) which adopts the previous usage counts as a reference history to examine the $\Delta^{(t)}$ matrix.

$$v_{ij}^{(t)} = \begin{cases} (r_{ij}^{(t-1)} - r_{min}) \times u_{ij}^{(t)} & , m = 1 \\ (r_{max} - r_{ij}^{(t-1)}) \times u_{ij}^{(t)} & , m = 0 \end{cases} \tag{4.3}$$

4.4.1 Mode-based Prediction (MBP)

The Mode-based Prediction (MBP) ignores the magnitude of usage counts by only considering two usage mode: one is active mode for using the App and another is inactive mode for not using the App. Then, a utility model is proposed to measure the usage change of a user, and the $\Delta^{(t)}$ matrix could be estimated through this model.

Trend Detection Phase.

In this phase, we decide the value of m in Eq. 4.2. If user i executed App j at time unit t (i.e. $c_{ij}^{(t)} > 0$), we would set m as 0 (increase the preference). By contrast, if $c_{ij}^{(t)} = 0$, we set m to 1 (decrease the preference).

$$\sum_{k \in P} u_{ik}^{(t)} - \sum_{k \in N} u_{ik}^{(t)} = 0 \tag{4.4}$$

$$u_{ij}^{(t)} = \begin{cases} \frac{1}{|P|} & , c_{ij}^{(t)} > 0 \\ \frac{1}{|N|} & , c_{ij}^{(t)} = 0 \end{cases} \tag{4.5}$$

$$v_{ij}^{(t)} = \begin{cases} \frac{r_{max} - r_{ij}^{(t-1)}}{|P|} & , c_{ij}^{(t)} > 0 \\ \frac{r_{ij}^{(t-1)} - r_{min}}{|N|} & , c_{ij}^{(t)} = 0 \end{cases} \tag{4.6}$$

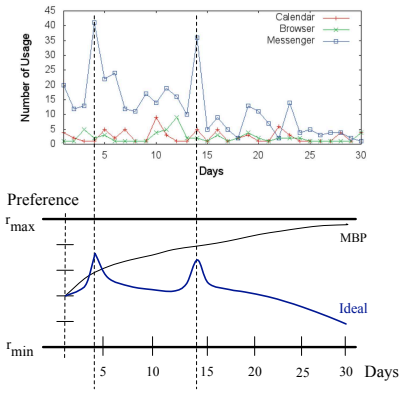


Figure 4.2: The preferences derived by MBP comparing with the Ideal preferences.

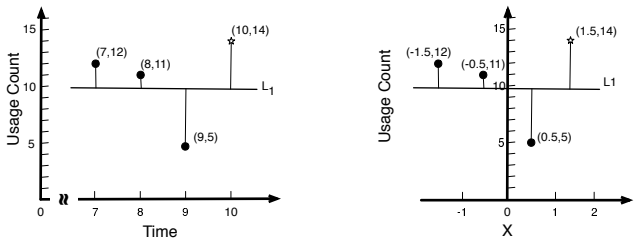
Change Estimation Phase.

The second phase is to estimate the absolute value of the preference change. In other words, we need to derive the value of $v_{ij}^{(t)}$ according to utility parameter, $u_{ij}^{(t)}$. Since we only have the information of usage mode of each App, we propose a utility model to derive the utility parameter based only on the usage mode. Intuitively, when a user spends more time on some Apps, (s)he should spend less time on others. Thus, we claim that the overall usage change among Apps should be equal to 0. Eq. 4.4 formulates the utility model, where P (respectively, N) is the set of Apps with active (respectively, inactive) mode. Suppose that the importance of each App is the same, the utility parameter is derived by Eq. 4.5. As a result, we can obtain $\delta_{ij}^{(t)}$ from $u_{ij}^{(t)}$, as shown in Eq. 4.6.

4.4.2 Reference-based Prediction (RBP)

Although MBP successfully avoids the magnitude of usage counts by adopting the usage mode and the utility model, ignoring the magnitude of the usage counts makes the estimated preferences not be able to reflect users' actual preferences. For example, in Fig. 4.2, the preference of Messenger predicted by MBP becomes higher and higher over time, since MBP increases the user's preference once the user invokes the Apps. However, we believe that the curve representing the preference of Messenger should be like the Ideal one. To obtain the ideal result, we propose a Reference-based Prediction (RBP) algorithm which compares the usage counts within an App instead of with other Apps.

RBP uses the previous usage counts of each App as a reference history, and derives a reference value from the reference history. In this chapter, the size of reference history is decided by a tunable parameter, h, which means how many historical data points are included into the reference history. The concept is that only when the actual usage count of an App is higher than the reference value, its preference is increasing. Similarly, the preference decreases only when the number of usage is less than the reference



(a) Deriving the star point from the three black points.

(b) Shifting the data points in Fig. 4.3(a).

Figure 4.3: Estimate the expected usage count (marked as a star point).

value.

$$slope = \frac{(h+1)\sum_{k=t-h}^{t} k \times c_{ij}^{(k)} - \sum_{k} k \sum_{k} c_{ij}^{(k)}}{(h+1)\sum_{k} k^2 - (\sum_{k} k)^2} = 0 \tag{4.7}$$

$$(h+1)\sum_{k=t-h}^{t} k \times c_{ij}^{(k)} - \sum_{k} k \sum_{k} c_{ij}^{(k)} = 0 \tag{4.8}$$

Trend Detection Phase.

In this phase, we decide whether the value of $\delta_{ij}^{(t)}$ is negative or positive. Here, we use the previous usage counts as the reference history and derive an expected number of usage as the reference value from the reference history. We adopt the linear regression to model the trend of reference history, and thus, the expected number of usage count should make the slope of the regression line be zero. Since the slope of a regression line represents the trend of the data points, the expected number of usage count which makes the regression line stay horizontal means that it makes the preference stay static. Then, if the actual number of usage is larger (smaller, respectively) than the expected number of usage, the preference is considered as increasing (decreasing, respectively). We use Fig. 4.3(a) to illustrate the concept of obtaining the expected number of usage by linear regression model. In Fig. 4.3(a), the three black points are the reference history (i.e. $h = 3$) and the reference value is the expected usage at time unit 10 (marked as a star point) which makes the regression line, L_1, be horizontal. Therefore, the goal of this phase is to find the value of star point by satisfying Eq. 4.7 which can be simplified into Eq. 4.8.

Since we only consider the slope of the regression line, we can shift the regression line left such that $\sum_{k=1}^{h+1} x_k = 0$, where x_k represents the shifted position in x-axis of the k-th point of reference history and thus, x_{h+1} is the shifted x-axis of the star point. As shown in Fig. 4.3(b), we can shift the regression line to the positions of x-axis as $< -1.5, -0.5, 0.5, 1.5 >$. Eq. 4.9 shows how to calculate the shifted x-axis positions. Now, we can simplify Eq. 4.8 into Eq. 4.10, where the index of time units of $c_{ij}^{(k)}$ is also shifted to $(k + t - h - 1)$ for $k = 1, 2, \ldots, h + 1$.

$$x_k = \frac{2k - (h+2)}{2} \tag{4.9}$$

$$(h+1)\sum_{k=1}^{h+1} x_k \times c_{ij}^{(k+t-h-1)} = 0 \tag{4.10}$$

Therefore, we can extract $c_{ij}^{(t)}$ from Eq. 4.10, and it is the expected number of usage $EXP(c_{ij}^{(t)})$, which could be derived from Eq. 4.11. For example, the value of the star point in Fig. 4.3(a) is $EXP(c_{ij}^{(10)}) = [(3+2)(12+11+5) - 2(1\times 12 + 2\times 11 + 3\times 5)]/3 = 42/3 = 14$.

$$\begin{aligned} EXP(c_{ij}^{(t)}) &= -\frac{\sum_{k=1}^{h} x_k \times c_{ij}^{(k+t-h-1)}}{x_{h+1}} \\ &= -\frac{\frac{2(1)-(h+2)}{2} c_{ij}^{(t-h)} + \cdots + \frac{2(h)-h-2}{2} c_{ij}^{(t-1)}}{\frac{2(h+1)-(h+2)}{2}} \\ &= \frac{(h+2)\sum_{k=1}^{h} c_{ij}^{(k+t-h-1)} - 2\sum_{k} k \times c_{ij}^{(k+t-h-1)}}{h} \end{aligned} \tag{4.11}$$

Change Estimation Phase.

As we have $EXP(c_{ij}^{(t)})$ to be the reference value, we need to formulate the utility parameter, $u_{ij}^{(t)}$, by calculating the distance between $c_{ij}^{(t)}$ and $EXP(c_{ij}^{(t)})$, denoted as $dist(EXP(c_{ij}^{(t)}), c_{ij}^{(t)})$. When $c_{ij}^{(t)}$ is far from $EXP(c_{ij}^{(t)})$, it means that the user is considered more likely to change his/her preference. Since we need a distance measure between 0 and 1, directly subtracting $EXP(c_{ij}^{(t)})$ from $c_{ij}^{(t)}$ or the other way around will not work. We devise the following distance measure. Here, $dist(EXP(c_{ij}^{(t)}), c_{ij}^{(t)})$ is estimated by evaluating how many possible cases are between $EXP(c_{ij}^{t})$ and $c_{ij}^{(t)}$. Therefore, when the preference is increasing ($m = 0$), we use $p(EXP(c_{ij}^{(t)}) \leq x \leq c_{ij}^{t})$ representing the probability of obtaining a number of usage in-between $EXP(c_{ij}^{(t)})$ and $c_{ij}^{(t)}$, where x is a random variable. On the other hand, when the preference is decreasing ($m = 1$), the distance between $EXP(c_{ij}^{(t)})$ and $c_{ij}^{(t)}$ is formulated as $p(c_{ij}^{(t)} \leq x \leq EXP(c_{ij}^{(t)}))$. In this chapter, we approximate the probability, $p(c_{ij}^{(t)})$, of using an App j by $c_{ij}^{(t)}$ times in a given time duration l (a parameter for the length of each time unit) by assuming a Poisson distribution shown in Eq. 4.12, where $\lambda = EXP(c_{ij}^{(t)})$. Now, the utility parameter, $u_{ij}^{(t)}$, could be formulated as in Eq. 4.13 and the absolute amount of preference change, $v_{ij}^{(t)}$, as in Eq. 4.14. We also list algorithm 4 to describe the flow of RBP in detail. In the first iteration, we set $r_{ij}^{(t)}$ to an initial preference, r_{init}, which is a tunable parameter. The preference will stay the same when the actual number of usage equals to the expected number of usage.

$$p(c_{ij}^{(t)}) = \frac{\lambda^{c_{ij}^{(t)}} \times e^{-\lambda}}{c_{ij}^{(t)}!} \tag{4.12}$$

$$u_{ij}^{(t)} = dist(\lambda, c_{ij}^{(t)}) = \begin{cases} p(\lambda \leq x \leq c_{ij}^{(t)}) = \sum_{k=\lambda}^{c_{ij}^{(t)}} p(k) & , c_{ij}^{(t)} > \lambda \\ p(c_{ij}^{(t)} \leq x \leq \lambda) = \sum_{k=c_{ij}^{(t)}}^{\lambda} p(k) & , c_{ij}^{(t)} < \lambda \end{cases} \tag{4.13}$$

$$v_{ij}^{t} = \begin{cases} (r_{max} - r_{ij}^{(t-1)}) \times \sum_{k=\lambda}^{c_{ij}^{(t)}} p(k) & , c_{ij}^{(t)} > \lambda \\ (r_{ij}^{(t-1)} - r_{min}) \times \sum_{k=c_{ij}^{(t)}}^{\lambda} p(k) & , c_{ij}^{(t)} < \lambda \end{cases} \tag{4.14}$$

Algorithm 4: Algorithm of Reference-based Prediction

Input: Input: $R^{(t-1)}$, $C^{(t)}$
Output: Output: $R^{(t)}$

foreach $c_{ij}^{(t)}$ **do**
  Let $r_{ij}^{(t)} \leftarrow r_{init}$
end
foreach *initialled* $r_{ij}^{(t-1)}$ **do**
  $EXP(c_{ij}^{(t)}) \leftarrow \frac{(h+2)\sum_{k=1}^{h} C - 2\sum_{k} k \times c^{(k)}}{h}$
  if $c_{ij}^{(t)} > EXP(c_{ij}^{(t)})$ **then**
    $m \leftarrow 0$ $P \leftarrow P \cup App_j$
  else
    $m \leftarrow 1$ $N \leftarrow N \cup App_j$
  end
  $\lambda \leftarrow EXP(c_{ij}^{(t)})$
  if $App_j \in P$ **then**
    $v_{ij}^{(t)} \leftarrow (r_{max} - r_{ij}^{(t-1)}) \times \sum_{k=c_{ij}^{(t)}}^{\lambda} \frac{\lambda^{c_{ij}^{(t)}} \times e^{-\lambda}}{c_{ij}^{(t)}!}$
  else
    $v_{ij}^{(t)} \leftarrow (r_{ij}^{(t-1)} - r_{min}) \times \sum_{k=\lambda}^{c_{ij}^{(t)}} \frac{\lambda^{c_{ij}^{(t)}} \times e^{-\lambda}}{c_{ij}^{(t)}!}$
  end
  $\delta_{ij}^{(t)} \leftarrow (-1)^m \times v_{ij}^{(t)}$
end
return $R^{(t)} \leftarrow R^{(t-1)} + \Delta^{(t)}$

4.5 Experimental Results

To evaluate the accuracy of the derived dynamic preferences, we examine the accuracy by testing the performance of using those derived preferences to make recommendation. We adopt the All-But-One evaluation methods [71] which, for each user, we iteratively skip one App from a user's preference list, and then make recommendation for this user. If the skipped App is recommended, we treat it as a hit. The hit ratio of user u at time unit t is calculated by Eq. 4.15, where k is the number of recommended Apps, $I(\cdot)$ is an indicator function defined in Eq. 4.16, $App_k(u,t)$ is the top-k Apps with highest preference

score for user u at time unit t, and $R_k(u,t)$ is the list of k recommended Apps for user u at time unit t. The length of one time unit, l, is 1 day for both App traces, and 7 days for the Last.fm dataset. Eventually, the overall accuracy is the average hit ratio of every user at every time unit, which is shown in Eq. 4.17.

$$HitRatio_k(u,t) = \frac{\sum_{i \in App_k(u,t)} I_{R_k(u,t)}(i)}{|App_k(u,t)|} \tag{4.15}$$

$$I_{R_k(u,t)}(i) \begin{cases} 1 & , i \in R_k(u,t) \\ 0 & , i \notin R_k(u,t) \end{cases} \tag{4.16}$$

$$Accuracy_k = OverallHitRatio_k = \frac{\sum_u \sum_t HitRatio_k(u,t)}{|U| \times |T|} \tag{4.17}$$

4.5.1 Environment

The range of users' preferences is set to [0,5]. The adopted recommendation algorithm is Collaborative Filtering (CF) provided by Apache project, Mahout, with its similarity function as Pearson correlation function.

Dataset Description.

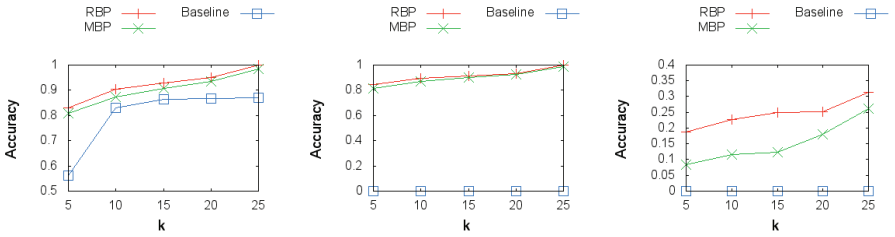
We have three real-world datasets: two are App usage traces and one is music listening log from Last.fm [7]. For the two traces of App usage, one is a smaller trace which consists of 30 users and 226 Apps, while the other one has 80 users and 650 Apps. Through the two different scales of datasets, we can ensure whether our methods are scalable or not. For the music listening dataset, we have a relatively huge amount of users in the Last.fm 1K-users dataset. The music listening dataset consists of 1000 users and 48,361 music albums which is a very sparse dataset we have to deal with. The total time duration for two App traces is half a year and for the music listening log is one and half years.

$$\frac{r_{ij}^t - r_{min}}{r_{max} - r_{min}} = \frac{c_{ij}^t - \min_k c_{ik}^t}{\max_k c_{ik}^t - \min_k c_{ik}^t} \tag{4.18}$$

$$r_{ij}^t = \frac{(c_{ij}^t - \min_k c_{ik}^t) \times (r_{max} - r_{min})}{\max_k c_{ik}^t - \min_k c_{ik}^t} + r_{min} \tag{4.19}$$

Compared Methods.

To compare the accuracy of our proposed algorithms, we adopt a usage-based method as the baseline. The usage-based method calculates the users' preferences only by the usage count. The item with largest number of usage will be assigned the preference of r_{max}, while the one with smallest number of usage will be assigned the preference of r_{min}. Besides, the preferences of other items are calculated by an interpolation method shown in Eq. 4.18.



(a) App-small dataset. (b) App-large dataset. (c) Last.fm dataset.

Figure 4.4: Accuracy evaluation with different k.

4.5.2 Performance Evaluation

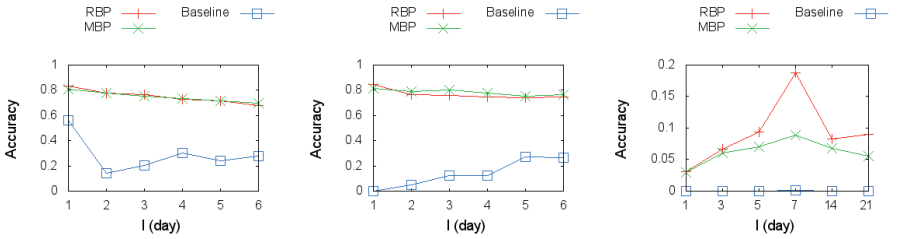
In this study, we evaluate the accuracy under various number of recommended Apps, k, and different length of a time unit, l over two proposed algorithms and one baseline method. Then, we focus on the proposed Reference-based Prediction (RBP) algorithm to see the accuracy when changing the parameter h which is used to control how many historical data are used.

Accuracy Changed by k.

Since k would affect the hit ratio, we calculate the hit ratio by different k from 5 to 25. However, although larger k could derive a better performance on hit ratio, fewer recommended items is more meaningful for users. Figs. 4.4(a) and 4.4(b) show the results of two App traces under different numbers of recommended items, k. Obviously, the accuracy increases as k grows up. Specifically, when $k = 5$, both RBP and MBP can achieve the accuracy of more than 80%. we note that in Fig. 4.4(b), the baseline remains close to zero even for $k = 25$, while in Fig. 4.4(a), the baseline achieves relatively low accuracy compared with RBP for $k = 5$. This is because the App-large dataset consists of more Apps and makes the dataset become sparser than App-small. Fig 4.4(c) depicts the results of Last.fm dataset. Since the music listening dataset is much sparser than the App traces, the performance on accuracy is not as good as the accuracy of the App traces. However, RBP is always the best method, while the baseline is close to zero. Here, for the two App traces, the length of one time unit is one day and the size of reference history, h, of RBP is set to 4 time units; for music listening dataset, the length of time unit is 1 week and the parameter h of RBP is 6 time units.

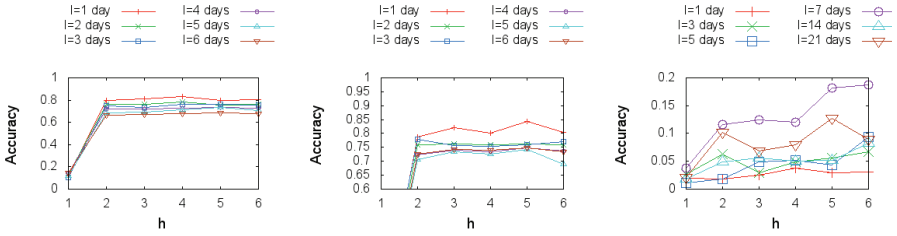
Impact of Parameter l.

Here, we evaluate the accuracy change of various length of a time unit. As can be seen in Figs. 4.5(a) and 4.5(b), both RBP and MBP slightly decrease their accuracy when the amount of training data increases. The best length of a time unit is one day which matches the human behavior. By contrast, the baseline method increases the accuracy when the number of training data becomes larger. Because the baseline method does not consider the temporal information, more training data could provide more information to overcome this drawback. However, when $d > 6$, the accuracy of baseline method also



(a) App-small dataset. (b) App-large dataset. (c) Last.fm dataset.

Figure 4.5: Accuracy evaluation with the length of a time unit varied.



(a) App-small dataset. (b) App-large dataset. (c) Last.fm dataset.

Figure 4.6: Accuracy evaluation with the size of reference history varied.

declines. On the other hand, as shown in Fig. 4.5(c), the best length of a time unit for Last.fm dataset is 7 days (one week) since music listening behavior is sparse and users may repeat the songs they listened in one week. Here, the reference history parameter, h, is set to 6 time units.

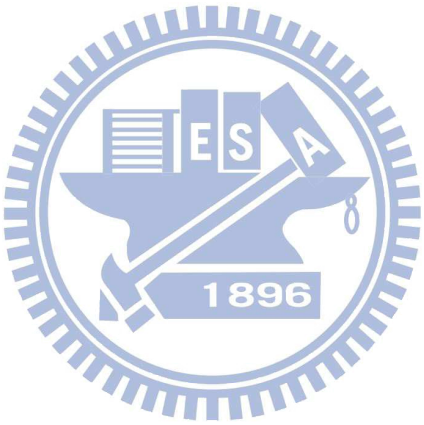
Impact of Parameter h for RBP.

Since the amount of reference history is a critical parameter for RBP algorithm, we evaluate the accuracy of recommendations under various reference histories. Figs. 4.6(a) and 4.6(b) depict the results of the App-small and App-large traces, and they reach the best accuracy on $h = 4$ and $h = 5$ respectively. Furthermore, the results of $h \geq 2$ are much better than the result of $h = 1$, because when $h = 1$, the number of reference points is too few to reflect the trend of users' usage. In addition, Fig. 4.6(c) shows the results of the Last.fm dataset, and the best accuracy falls on $h = 6$ and $l = 7$. Because the Last.fm is a sparse dataset, RBP algorithm needs more reference points and training data to achieve a better performance. Empirically, the setting of h does highly depend on different applications. In this chapter, we suggest choosing a proper h larger than 4, since the regression line constructed in the first phase of RBP is more meaningful to reflect the trend of users' usage.

4.6 Conclusion

We proposed a novel dynamic preference prediction problem which is to dynamically quantize a user's preferences on Apps they have used from their usage traces. Two effective algorithms are designed to solve this problem. One is named Mode-base Prediction (MBP) which adopts a user's binary usage

mode (active and inactive) and a proposed utility model to predict the preference value on an App. The other one is named Reference-base Prediction (RBP) which discovers a reference value by solving an optimization problem in a linear regression model and constructs a probabilistic model to check if the current behavior satisfies the reference model. RBP estimates the users' preferences by measuring the difference between actual usage and the derived reference value. In the experiments section, we evaluate the derived dynamic preferences by applying Collaborative Filtering. When the derived preferences can provide more accurate recommendation, the preferences are considered closer to users' actual affinities. As the experimental results show, the derived preferences of both MBP and RBP are effective. In addition, the RBP method can reach the accuracy of more than 80% for App traces. We suggest that the proposed dynamic preferences are valuable for many applications, such as providing recommendation of mobile applications, predicting and analysing users behavior, and make marketing decision.



Chapter 5

Mining Sequential Patterns Across Multiple Sequence Databases

5.1 Introduction

Sequential pattern mining has attracted a considerable amount of research effort recently [3][5][14] [54][55]. Given a sequence database that contains a set of sequences and a user-specified threshold (the minimum support), the main task of sequential pattern mining is to discover frequent subsequences that appear in a sufficient number of sequences. Since sequential pattern mining is able to discover temporal relationship (i.e., order of events), a significant amount of research works has elaborated on developing novel approaches to discover sequential patterns for a variety of applications [11][19][39][48][56][61].

Note that prior works only mine sequential patterns in one sequence database. This sequence database consists of sequences of events in one domain. For example, given a sequence database of purchasing in a supermarket, frequent purchasing behavior is discovered. In many real world applications, we may have events in multiple domains. Consider payment lists of credit cards, where a user uses a credit card for a variety of services, such as payments in restaurants, food, books and movies. These payments are referred to as events in different domains. For each domain, we could extract these events and build the corresponding sequence database. Then, one could utilize sequential pattern mining to discover frequent sequences. For example, in the movie domain, one sequential pattern is that users watch a series of movies related to Harry Potter. On the other hand, in the book domain, one sequential pattern is that users buy a series of novels related to Harry Potter. From these two sequential patterns of two domains, one could derive a composite sequential pattern across two domains (referred to as a *multi-domain sequential pattern*) if events in these two sequential patterns closely occur together (i.e., events occur within the same time window). For the above example, Figure 1 shows that these two sequences are sequential patterns in the movie domain and the book domain, respectively. Moreover, the corresponding time of

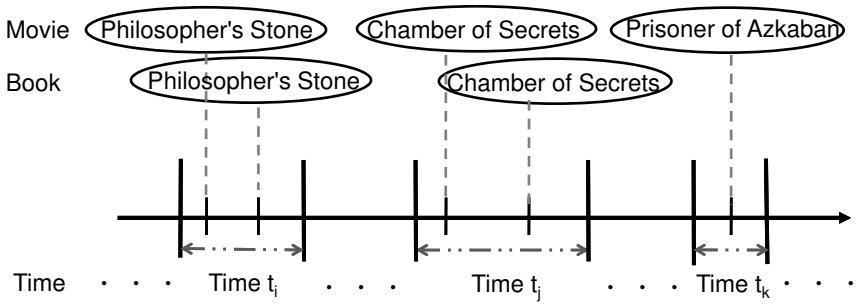


Figure 5.1: An example of multi-domain sequential pattern.

events in these two sequences are within the same time window. In this chapter, we claim that discovering sequential patterns from multiple domains will provide a unique way to reveal complex relationships across multiple domains. In Figure 1, one could infer that users are likely to buy novels of Harry Potter, which is motivated by movies. Furthermore, this multi-domain sequential pattern also implies that users who go to watch movies are likely to be triggered by books bought. To reveal more information from sequence databases across multiple domains, multi-domain sequential patterns are very useful. Depending on requirements of applications, one could decide which domains should be involved in mining multi-domain sequential patterns. In our above example, if a user wants to know the cross-relationship between book and movie domains, sequence databases of these two domains are given. Consequently, such a multi-domain sequential pattern captures the cross-relationship among multiple domains, which in turn can yield significant information and reveal more knowledge.

Given a set of sequence databases across multiple domains, we aim at mining multi-domain sequential patterns, where a multi-domain sequential pattern is a sequence of events whose occurrence time is within a pre-defined time window. With a set of sequence databases, these sequence databases could be joined into one sequence database according to time information of sequences. Then, by exploring traditional sequential pattern mining algorithms, we could obtain multi-domain sequential patterns as well. This method is referred to as algorithm *Naive* in the chapter and the details of this algorithm are presented later. However, there are three drawbacks in this algorithm: (1) integrating sequence databases of multiple domains into a single sequence database incurs a considerable cost due to the nature of joining operations, (2) the length of each sequence becomes longer and the number of items becomes huge after joining operations, and (3) sequential patterns mined should be further verified whether these sequential patterns satisfy multi-domain sequential patterns or not. Hence, the above algorithm unavoidably exhibits poor efficiency and scalability performance, which calls for the design of efficient mining algorithms for multi-domain sequential patterns.

To avoid the above poor performance issues, in this chapter, we first propose algorithm IndividualMine in which sequential patterns in each sequence database should be mined first and then the sequential patterns in each domain are integrated as candidate multi-domain sequential patterns. Clearly, algorithm

IndividualMine is able to avoid join operations among sequence databases. In Figure 1, it can be seen that in the movie domain (respectively, book domain), we could have one sequential pattern, a series of movies (respectively, books) related to Harry Potter. By checking the corresponding time of events in these two sequential patterns, we could combine these two sequential patterns into one multi-domain sequential pattern since each event in these two sequential patterns has close occurrence time. It is possible that sequential patterns from each sequence database cannot be formed as multi-domain sequential patterns since events' occurrence time is not close. Though avoiding join operations, algorithm IndividualMine is likely to suffer from mining cost since sequential patterns in each sequence database should be discovered. Consequently, we propose algorithm PropagatedMine to further reduce the mining cost in each sequence database. Algorithm PropagatedMine first performs one sequential pattern mining from one sequence database. In light of sequential patterns mined, algorithm PropagatedMine propagates time information (referred to as the time-instance set) of sequential patterns mined to other sequence databases. By utilizing time-instance sets, we are able to extract a subset of sequences from sequence databases, where the subset of sequences has the same time information. As such, only a limited number of sequences that are likely to form multi-domain sequential patterns are extracted. Furthermore, sequential patterns mined are represented as a lattice structure for further reducing the number of time-instance sets to other sequence databases. In addition, we develop some mechanisms to allow some empty sets in multi-domain sequential patterns. Performance of the proposed algorithms is comparatively analyzed and sensitivity analysis is conducted. It is shown by our simulation results that both algorithms IndividualMine and PropagatedMine perform better than algorithm Naive. By exploring propagation and lattice structures, algorithm PropagatedMine outperforms algorithms IndividualMine and Naive in terms of efficiency (i.e., the execution time).

The remainder of the chapter is organized as follows. In Section 5.2, we present existing research works of mining sequential patterns. In Section 5.3, some notations and the problem definition are given. Our proposed algorithms are described in Section 5.4. Performance study and experimental results are shown in Section 5.5. Section 5.6 concludes with this chapter.

5.2 Related Works

A significant amount of research efforts has been devoted to sequential pattern mining [8, 21, 27, 68, 70, 74, 75]. The problem of sequential pattern mining is first formulated in [3] and the authors in [3] proposed mining algorithms based on the Apriori algorithm. Algorithm GSP [65] was developed for mining sequential patterns using a breadth first search and button-up method, whereas algorithm SPADE [79] employed a depth first search and button-up method with an ID-list. The authors in [24][54][55] exploited the projection concept to reduce the amount of data for sequential pattern mining. To prevent candidate

cid	cust-grp	city	age-grp	sequence
10	business	Boston	middle	$< (bd)(c)(b)(a) >$
20	professional	Chicago	young	$< (bf)(ce)(fg) >$
30	business	Chicago	middle	$< (ah)(a)(b)(f) >$
40	education	New York	retired	$< (be)(ce) >$

Table 5.1: Multi-dimensional sequence database [56].

generation, DISC-all [13] used a novel sequence comparison strategy. A progressive concept has been explored in mining sequential patterns to capture the dynamic nature of data addition and deletion [26]. The above research works are focused on improving the performance of traditional sequential pattern mining.

Some variations and applications on sequential patterns are proposed recently. We mention in passing that the authors in [56] developed to mine multi-dimensional sequential patterns, in which sequential patterns indicate not only frequent sequences but also some attributes in the category dimensions. In [56], the sequence database consists of category attributes and sequence attributes, and Table 5.1 shows an example of a multi-dimensional sequence database. Clearly, the problem of mining multi-domain sequential patterns is very different from the problem in [56] in terms of the input and output of problem definitions. In this chapter, the input is the set of sequence databases and the output is the set of multi-domain sequential patterns that consist sequences of co-occurred events across sequence databases. However, [78] is another study that mentioned multidimensional sequence. In [78], sequence data are divided into different dimensions according to user's specification. However, there is no time information amount different dimensions. In other words, each event in different dimensions is not co-occurred. Therefore, it is quite different with our study. Furthermore, the problem in [6] is to discover events that are occurred together. In contrast, our problem is that given a set of sequence databases, we intend to discover sequences consisting of co-occurred events. Moreover, the authors in [22] proposed the problem of distributed sequential pattern mining, where each set of co-occurred events is complete and sequences are separated into different databases. Similarly, the problem in [35] is indeed a distributed sequential pattern mining problem and the authors in [35] exploited the concepts of approximate patterns and local clustering to avoid noise and a large number of local patterns. As pointed early, given a payment list of credit cards, we could divide payments into several domains according to payment services. Thus, our problem of mining multi-domain sequential patterns is not the same as distributed mining of sequential patterns.

To the best of our knowledge, previous studies have not adequately explored multi-domain sequential patterns, let alone proposing efficient algorithms for mining such sequential patterns. The contributions of this chapter are twofold: (1) exploiting novel and useful sequential patterns (i.e., multi-domain sequential patterns), and (2) devising algorithms IndividualMine and PropagatedMine to efficiently mine multi-

Domain Database D_1		
ID	Time sequences	Sequences
s_1	$< (T_1)(T_2)(T_3)(T_4) >$	$< (a)(b,c)(b,c,d)(e) >$
s_2	$< (T_5)(T_6)(T_7) >$	$< (a,b)(b,c)(c,e) >$
s_3	$< (T_{10})(T_{12})(T_{13}) >$	$< (a,e)(h)(g,j) >$
s_4	$< (T_{21})(T_{22})(T_{23})(T_{24}) >$	$< (a,b,f)(d)(b,c)(e,f) >$

Domain Database D_2		
ID	Time sequences	Sequences
l_1	$< (T_{21})(T_{22})(T_{23})(T_{24}) >$	$< (1,2,5)(7)(2,3)(4,5,6) >$
l_2	$< (T_{10})(T_{12})(T_{13}) >$	$< (1,6)(5)(9,10) >$
l_3	$< (T_5)(T_6)(T_7) >$	$< (1,3)(2,4)(8) >$
l_4	$< (T_1)(T_2)(T_3)(T_4) >$	$< (1,2)(2,3)(6)(4,5) >$

Table 5.2: Example of sequence databases in two domains.

domain sequential patterns. Our preliminary works were presented in [42] and [43]. In this chapter, more detailed complexity and theoretical analysis are conducted. Also, we develop some mechanisms in each proposed algorithm to allow multi-domain sequential patterns with some empty sets in some domains. In particular, by exploring lattice structures, algorithm PropagatedMine is able to further reduce the number of candidate multi-domain sequential patterns. Furthermore, an extensive performance study is conducted and sensitivity analysis is investigated on several parameters for each algorithm.

5.3 Preliminaries

Assume that each domain has its own set of items and a sequence database. The problem of mining multi-domain sequential patterns is that given a set of sequence databases, we aim at discovering sequential patterns that consist of co-occurred events across multiple domains. Table 5.2 shows two domains with its own sequence database, where in each sequence of sequence databases, the corresponding time sequence indicates the occurrence time of events. For example, in sequence s_1 in D_1, it can be seen that event a occurs at T_1 and both b and c occur at T_2. By joining these two sequence databases via their time sequences, we could have one Multi-Domain sequence dataBase (referred to as MDB). As such, Table 5.3 is an example of multi-domain sequence database.

To facilitate the presentation of multi-domain sequences, one sequence s_i in domain D_i is expressed by $< X_{i1}, X_{i2}, \ldots, X_{il} >$, where X_{ij} is the jth element of sequence s_i, and l is the number of elements of s_i. Therefore, a multi-domain sequence across k domains (abbreviated as *k-domain sequence*) is represented as $M = [s_1, s_2, \ldots, s_k]^T$ and is further denoted as $M = \begin{bmatrix} X_{11} & X_{12} & \ldots & X_{1l} \\ X_{21} & X_{22} & \ldots & X_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ X_{k1} & X_{k2} & \ldots & X_{kl} \end{bmatrix}$, where each column is a set of itemsets that occur within the same time window, denoted as T_i. A time sequence $TS(M)$

ID	Time sequences	Multi-domain sequences
S_1	$<(T_1)(T_2)(T_3)(T_4)>$	$\begin{bmatrix}(a) & (b,c) & (b,c,d) & (e)\\ (1,2) & (2,3) & (6) & (4,5)\end{bmatrix}$
S_2	$<(T_5)(T_6)(T_7)>$	$\begin{bmatrix}(a,b) & (b,c) & (c,e)\\ (1,3) & (2,4) & (8)\end{bmatrix}$
S_3	$<(T_{10})(T_{12})(T_{13})>$	$\begin{bmatrix}(a,e) & (h) & (g,j)\\ (1,6) & (5) & (9,10)\end{bmatrix}$
S_4	$<(T_{21})(T_{22})(T_{23})(T_{24})>$	$\begin{bmatrix}(a,b,f) & (d) & (b,c) & (e,f)\\ (1,2,5) & (7) & (2,3) & (4,5,6)\end{bmatrix}$

Table 5.3: An example of a multi-domain sequence database.

is represented as $TS(M) =< T_1, T_2, \ldots, T_l >$ to indicate the occurrence time of M. Actually, the time window, a time interval, is determined in accordance with application requirements.

With the above representation of multi-domain sequences, we further define the length and the number of elements for multi-domain sequential patterns. Since a multi-domain sequence consists of multiple sequences from various domains, the length of a multi-domain sequence across k domains can be defined as follows:

Definition 5. (Length and number of elements) *Let $M = [s_1, s_2, \ldots, s_k]^T$ be a k-domain sequence. The length of M, denoted as $|M|$, is the length of the longest sequence in multi-domain sequence M. Furthermore, the number of elements in a multi-domain sequence, expressed by $e(M)$, is the number of itemsets in the multi-domain sequence.*

For example, given a 2-domain sequence $M = \begin{bmatrix}(a) & (b,c) & (b)\\ (1) & (2) & (1,2,3)\end{bmatrix}$, the length of M is 5 due to that the longest sequence $<(1)(2)(1,2,3)>$ in M, and the number of elements is 3 (i.e., $e(M) = 3$).

Once we have the definition of the length and the number of elements for a multi-domain sequences, the containing relation among multi-domain sequences is thus defined as follows:

Definition 6. (Containing relation) *Suppose that we have two multi-domain sequences $M = \begin{bmatrix}X_{11} & X_{12} & \ldots & X_{1b}\\ X_{21} & X_{22} & \ldots & X_{2b}\\ \vdots & \vdots & \ddots & \vdots\\ X_{a1} & X_{a2} & \ldots & X_{ab}\end{bmatrix}$ and $N = \begin{bmatrix}Y_{11} & Y_{12} & \ldots & Y_{1b'}\\ Y_{21} & Y_{22} & \ldots & Y_{2b'}\\ \vdots & \vdots & \ddots & \vdots\\ Y_{a1} & Y_{a2} & \ldots & Y_{ab'}\end{bmatrix}$, where $e(M) \leqslant e(N)$. M is contained by N, denoted as $M \sqsubseteq N$, if and only if there exists an integer list $L(M,N)$, denoted as $< l_1, l_2, \ldots, l_b >$, such that $1 \leqslant l_1 < l_2 < \ldots < l_b \leqslant b'$ and $X_{ij} \subseteq Y_{il_j}$, where $1 \leq i \leq a$ and $1 \leq j \leq b$.*

For example, assume that $M = \begin{bmatrix}(a) & (b,c)\\ (2) & (6)\end{bmatrix}$ and $N = \begin{bmatrix}(a) & (b,c) & (b,c,d) & (e)\\ (1,2) & (2,3) & (6) & (4,5)\end{bmatrix}$. It

can be verified that M is contained by N since there exist integer list $L(M,N) =< 1,3 >$ such that $1 \leqslant 1 < 3 \leqslant 4$,and $(a) \subseteq (a)$, $(2) \subseteq (1,2)$, $(b,c) \subseteq (b,c,d)$ and $(6) \subseteq (6)$.

Based on the above definitions, a multi-domain sequence database is a set of multi-domain sequences. Consider an example of a multi-domain sequence database in Table 5.3, where the number of 2-domain sequences is 4. Given a multi-domain sequence database MDB, the support value of a multi-domain sequence M is the number of multi-domain sequences in MDB that contain the multi-domain sequence M.

Multi-domain Sequential Pattern Mining: Given a set of sequence databases across multiple domains, one could join these sequence databases as one multi-domain sequence database. Then, the task of mining multi-domain sequential patterns is to derive multi-domain sequences with their supports larger than a user-specified minimum support threshold δ in MDB. For example, for the multi-domain sequence database MDB in Table 5.3 and a minimum support $\delta = 3$, multi-domain sequential patterns are $\begin{bmatrix} (a) \\ (1) \end{bmatrix}$, $\begin{bmatrix} (b) \\ (2) \end{bmatrix}$, $\begin{bmatrix} (b) \\ (3) \end{bmatrix}$, $\begin{bmatrix} (c) \\ (2) \end{bmatrix}$, $\begin{bmatrix} (b,c) \\ (2) \end{bmatrix}$, $\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}$, $\begin{bmatrix} (a) & (c) \\ (1) & (2) \end{bmatrix}$, and $\begin{bmatrix} (a) & (b,c) \\ (1) & (2) \end{bmatrix}$.

Notice that joining these multiple sequence databases is costly due to the nature of join operations. It can be verified that multi-domain sequential patterns contain sequential patterns in each domain. For example, $\begin{bmatrix} (a) & (b,c) \\ (1) & (2) \end{bmatrix}$ is a multi-domain sequential pattern, where $(a)(b,c)$ (respectively, $(1)(2)$) is a sequential pattern in domain D_1 (respectively, D_2) and events in $(a)(b,c)$ and $(1)(2)$ has the same time sequences. Thus, in this chapter, we propose algorithms to discover multi-domain sequential patterns without joining.

5.4 Algorithms of Mining Multi-domain Sequential Patterns

In this section, we first describe one *Naive method* in which multiple sequence databases are joined as one sequence database, and multi-domain sequential patterns are derived by using traditional sequential pattern mining algorithms (e.g., PrefixSpan [54][55]). As pointed out early, to avoid the overheads of joining multiple sequence databases, we then propose algorithm IndividualMine in which sequential patterns in each sequence database should be mined and further merged for possible multi-domain sequential patterns. Furthermore, to further reduce the cost of mining sequential patterns in each sequence database, algorithm PropogatedMine is proposed. By propagation of sequential patterns to other sequence databases, the number of sequences in other sequence databases is reduced. In addition, the above three algorithms could be extended to discover multi-domain sequential patterns with some empty sets in some domains.

ID	Sequences
S_1	$< (a,1,2)(b,c,2,3)(b,c,d,6)(e,4,5) >$
S_2	$< (a,b,1,3)(b,c,2,4)(c,e,8) >$
S_3	$< (a,e,1,6)(h,5)(g,j,9,10) >$
S_4	$< (a,b,f,1,2,5)(d,7)(b,c,2,3)(e,f,4,5,6) >$

Table 5.4: An example of a transformed sequence database.

5.4.1 Naive Algorithm with One Multi-domain Sequence Database

As mentioned early, to mine multi-domain sequential patterns, one naive method is joining sequence databases into one multi-domain sequence database. Then, this multi-domain sequence database is transformed such that the naive algorithm could utilize existing sequential pattern mining algorithms. Consequently, in the naive algorithm, there are two steps: the joining step and the mining step. In the joining step, multiple sequence databases are first joined together by the time sequences and then the multiple sequence databases are thus transformed into a sequence database. In the mining step, one could utilize existing sequential pattern mining algorithms to derive sequential patterns. In light of sequential patterns mined, we have to separate the items from different domains and derive multi-domain sequential patterns. The detailed steps are described as follows:

Step 1: Joining Step: In the beginning, sequence databases are joined by their time sequences to form one multi-domain sequence database. For example, Table 5.3 is derived by performing the join process among two sequence databases in Table 5.2. It can be verified that s_1 in D_1 sequence database and l_4 in D_2 sequence database are joined as one sequence S_1 in Table 5.3. With the multi-domain sequence database derived, one should transform this multi-domain sequence database into one sequence database. Explicitly, in Table 5.3, for each sequence, time sequences are deleted and multi-domain sequences could be viewed as one sequence. Table 5.4 is an example of a sequence database transformed from Table 5.3. It can be seen that in sequence S_1 in Table 5.4, co-occurred events from multiple domains are viewed as one event. For example, (a, 1, 2) comes from $\begin{bmatrix} (a) \\ (1,2) \end{bmatrix}$ in sequence S_1 of Table 5.3.

Step 2: Mining Step

According to the sequence database derived in Step 1, by exploiting traditional sequential pattern mining algorithms, we could derive sequential patterns. The second column of Table 5.5 shows some examples of sequential patterns mined from the sequence database in Table 5.4 with the minimum support as 3. However, even if a sequence database is obtained, traditional sequential pattern mining algorithms are not directly able to mine multi-domain sequential patterns. This is due to that several sequential patterns mined do not contain events from all domains. Thus, each sequential pattern should be represented as multi-domain sequential patterns. Then, we could first verify whether multi-domain sequential patterns consists of events from all domains or not. For example, the third column of Table 5.5 shows multi-domain sequential patterns from the second column of Table 5.5. Since we have all events of all

Pattern ID	Sequential patterns	multi-domain sequential patterns
P_1	$< (1)(b,2)(e) >$	$\begin{matrix}(b) & (e)\\ (1) & (2)\end{matrix}$
P_2	$< (a,1)(5) >$	$\begin{matrix}(a)\\ (1) \quad (5)\end{matrix}$
P_3	$< (a,1)(c,2) >$	$\begin{matrix}(a) & (c)\\ (1) & (2)\end{matrix}$
P_4	$< (b,3) >$	$\begin{matrix}(b)\\ (3)\end{matrix}$
P_5	$< (b,c,2) >$	$\begin{matrix}(b,c)\\ (2)\end{matrix}$
P_6	$< (b,c,2)(e) >$	$\begin{matrix}(b,c) & (e)\\ (2)\end{matrix}$

Table 5.5: An example of a transformed sequence database.

domains, it is very straightforward to represent sequential patterns as multi-domain sequential patterns. It can be seen in Table 5.5, P_1, P_2 and P_6 have some empty sets and these patterns are referred to as multi-domain sequential patterns with empty sets (abbreviated as *relaxed multi-domain sequential patterns*). On the other hands, P_3, P_4 and P_5 are called *strong multi-domain sequential patterns* since all co-occurred events are from all domains required.

Algorithm Naive needs to perform join operations among multiple sequence databases. Due to join operations, the performance of algorithm Naive is not efficient. Furthermore, in order to utilize traditional sequential pattern mining algorithms, one sequence database is derived by transforming from one multi-domain sequence database joined from sequence databases. Clearly, with events from all domains, the sequence database contains long sequences, which is not efficient in mining sequential patterns. With the above two drawbacks of algorithm Naive, we develop two efficient algorithms for mining multi-domain sequential patterns without joining sequence databases.

5.4.2 Algorithm IndividualMine: Mining Patterns in Each Domain

In this section, we develop algorithm IndividualMine. Figure 5.2 shows the overview of algorithm IndividualMine, where algorithm IndividualMine consists of two phases: the mining phase and the checking phase. In the mining phase, sequential patterns in each sequence database are first mined by utilizing sequential pattern mining algorithms (e.g., PrefixSpan [54][55]). In the checking phase, sequential patterns from all domains are combined to generate candidate multi-domain sequential patterns. If a candidate multi-domain sequential pattern has its support value larger than the minimum support threshold, this candidate multi-domain sequential pattern is a multi-domain sequential pattern. The support counts of candidate multi-domain sequential patterns will be described later.

Without loss of generality, given k sequence databases, we intend to derive multi-domain sequential patterns across k domains. Furthermore, we denote the set of k sequence databases as $\{D_1, D_2, \ldots, D_k\}$, and SP_i as the set of i-domain sequential patterns across a set of i sequence databases

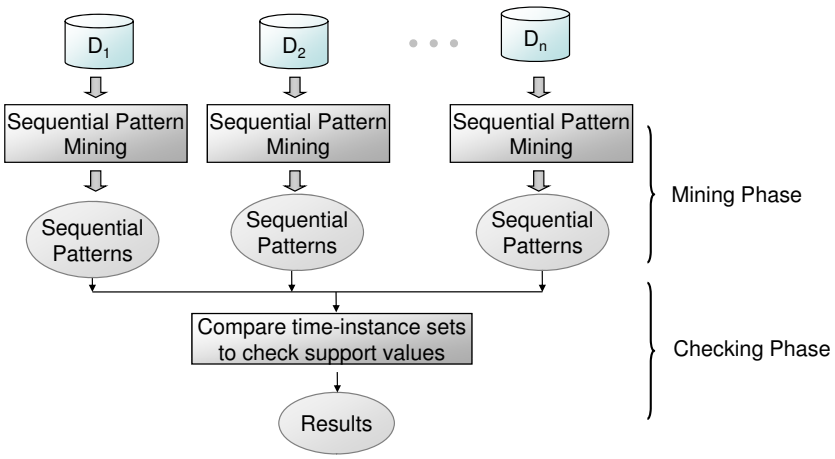


Figure 5.2: Overview of algorithm IndividualMine.

(i.e., $\{D_1, D_2, \ldots, D_i\}$). To derive k-domain sequential patterns, we should start with one sequential patterns from one domain and progressively composite sequential patterns from other domains until the number of domains is k. Hence, sequential patterns mined in D_1 is first in the set of SP_1. Then, for each pattern in SP_1, candidate 2-domain sequential patterns (across two domains $\{D_1$ and $D_2\}$) are generated by combining sequential patterns in domain D_2. For example, given a minimum support as 3, in our above example in Table 5.2, $< (a)(b) >$ is a sequential pattern and is put in the set of SP_1. Also, $< (1), (2) >$ is one sequential pattern in D_2. Consequently, we could have a candidate 2-domain sequential pattern $\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}$.

After generating candidate multi-domain sequential patterns, their support values should be determined. As can be seen in Table 5.2, each sequence is associated with its own time sequence. Thus, one could use time sequences to derive support values. Explicitly, the time-instance set of sequence M is defined as follows:

Definition 7. (Time-instance set) *Let MDB be a k-domain sequence database*[1] *and M be a k-domain sequence. The time-instance set of M is defined as $TIS(M) = \{< TS(N) : L(M,N) > | N \in MDB$ and $M \sqsubseteq N\}$.*

Based on the above definition, for a candidate multi-domain sequential pattern, we could determine its support value by evaluating the intersections in time-instance sets of each sequential pattern. For example, to determine the support of $\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}$, we should check both time-instance set of $< (a)(b) >$ and $< (1)(2) >$ in Table 5.2. It can be seen that in Table 5.2, the time-instance set of $< (a)(b) >$ is $\{< (T_1)(T_2)(T_3)(T_4) : 1, 2 >, < (T_1)(T_2)(T_3)(T_4) : 1, 3 >, < (T_5)(T_6)(T_7) : 1, 2 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 1, 3 >\}$. Moreover, we could have $TIS(< (1)(2) >)$ as $\{< (T_1)(T_2)(T_3)(T_4) : 1, 2 >, < (T_5)(T_6)(T_7) : 1, 2 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 1, 3 >\}$. Thus, the support of a candidate 2-domain sequential pattern

[1]To facilitate our presentation, one could image that MDB are virtually joined by multiple sequence databases.

$\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}$ is represented as $TIS(\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}) = \{< (T_1)(T_2)(T_3)(T_4) : 1,2 >, < (T_5)(T_6)(T_7) : 1,2 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 1,3 >\}$. Therefore, $Support(\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}) = |TIS(\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix})| = 3$. Given a minimum support threshold of 3, $\begin{bmatrix} (a) & (b) \\ (1) & (2) \end{bmatrix}$ is a 2-domain sequential pattern, since its support value is not less than the minimum support. Consequently, through the time-instance sets, support values for candidate multi-domain sequence patterns are derived.

Once we have 2-domain sequential patterns, these 2-domain sequential patterns are in the set of SP_2. Then, for each pattern in SP_2, candidate 3-domain sequential patterns and their corresponding supports will be generated by the above same procedure. Given sequential patterns in k domains, k-domain sequential patterns are derived by iteratively expanding one domain in each round until the number of rounds is k.

Algorithm: IndividualMine
Input: Sequence databases across n domains $D_1, D_2, \ldots, D_n$, and minimum support δ.
Output: Multi-domain sequential patterns across n domains.
Begin
  Let C_k be the set of candidate patterns across k domains, where $k = 1, 2, \ldots, n$.
  Apply sequential pattern mining on each domain D_i, $i = 1, 2, \ldots, n$.
  Let SP_1 be the set of sequential patterns mined in D_1.
  For each domain D_{i+1}, $i = 1, 2, \ldots, n-1$
    For each $P \in SP_i$
      For each sequential pattern Q of D_{i+1}
        If $e(Q) = e(P)$ **Then** append $\begin{bmatrix} P \\ Q \end{bmatrix}$ to C_{i+1}.
    For each candidate $c \in C_{i+1}$
      If $Support(c) \geqslant \delta$ **Then** append c to SP_{i+1}.
  Output=SP_n.
End

Without joining, algorithm IndividualMine could still discover multi-domain sequential patterns. It can be seen that in algorithm IndividualMine, each domain should individually perform sequential pattern mining algorithms, which incurs a considerable amount of mining cost. Furthermore, those sequential patterns mined from each domain are not necessarily able to become multi-domain sequential patterns. Thus, to further reduce the cost of mining sequential patterns in each domain and the number of candidate multi-domain sequential patterns, we develop algorithm PropagatedMine in which those sequences that are likely to form multi-domain sequential patterns are extracted from their sequence databases.

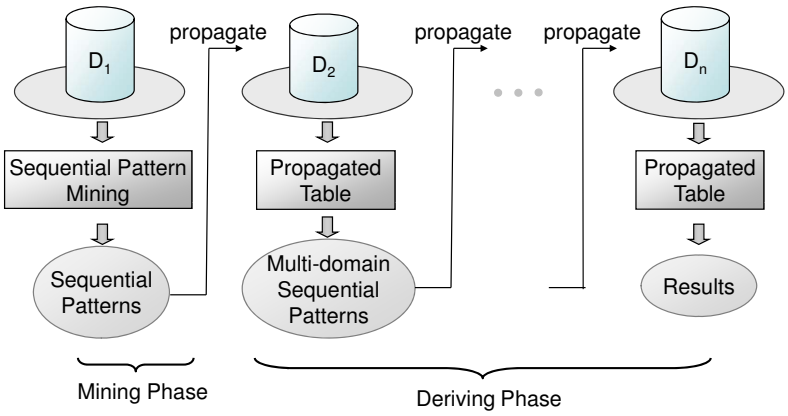


Figure 5.3: Overview of algorithm PropagatedMine.

5.4.3 Algorithm PropagatedMine: Propagating Sequential Patterns among Domains

Algorithm PropagatedMine is designed to reduce the mining cost in each sequence database. Explicitly, algorithm PropagatedMine first performs sequential pattern mining in one domain (referred to as the starting domain) and then propagates time-instance sets of the mined sequential patterns to other domains. By propagating time-instance sets, only those sequences that have the same time sequences with the time-instance sets are extracted, thereby reducing the mining space in each sequence database. Algorithm PropagatedMine iteratively propagates time-instance sets of multi-domain sequential patterns to the next domain until all domains have been mined. Figure 5.3 shows an overview of algorithm PropagatedMine, where there are two phases in algorithm PropagatedMine: the mining phase and the deriving phase.

In the mining phase, PropagatedMine utilizes existing sequential pattern mining algorithms to discover sequential patterns in a starting domain (i.e., D_1) and then propagates these patterns to other domains. Note that the mined sequential patterns in the starting domain provide a guideline to extract multi-domain sequential patterns from other domains, and hence for mining multi-domain sequential patterns in sequence databases across multiple domains, the length and the number of elements of multi-domain sequences are constrained by sequential patterns mined in the starting domain. Consequently, sequential patterns mined in the starting domain could be represented as a lattice structure to facilitate the generation of candidate multi-domain sequential patterns across other domains.

For example, assume that the starting domain is set to D_1 in Table 5.2 and that sequential patterns are then found using existing sequential pattern mining algorithms with the same minimum support 3. The mined sequential patterns are represented as a lattice structure in Figure 5.4, where each node represents a sequential pattern, the linkages of nodes (or *intradomain links*) represent containing relation, and nodes are ordered by the number of elements. In Figure 5.4, those nodes having the same number

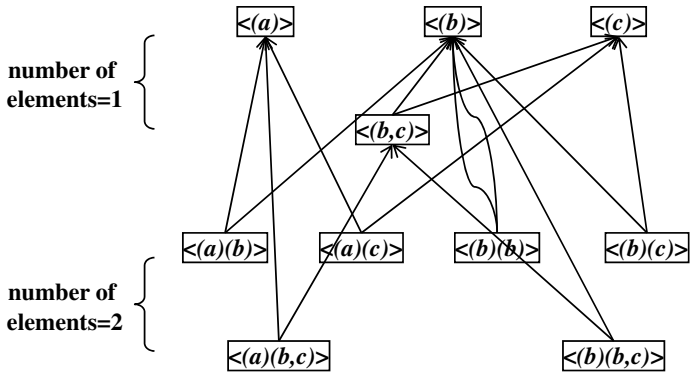


Figure 5.4: An example of lattice structures for sequential patterns in a starting domain (i.e., D_1 in Table 2).

of elements are further arranged level by level according to their sequence lengths and nodes with one element are placed level by level in increasing order of sequence length. For example, $< (b, c) >$ in Figure 5.4 is below the nodes whose sequence length is 1 (e.g., $< (b) >$). As mentioned above, the lattice structure is used as a guideline for propagating time-instance sets of sequential patterns to other domains. In the deriving phase, algorithm PropagatedMine extracts those sequences with occurrence times equal to those of the time-instance sets propagated. Thus, for each propagated time-instance set, we can build the corresponding propagated table as defined in Definition 8.

Definition 8. (Propagated table) *Let M be a k-domain sequential pattern. The propagated table of M in sequence database D_{k+1} is denoted as $D_{k+1}||_M = \{< S_i[l_1], S_i[l_2], \ldots, S_i[l_b] > \mid < TS(S_i) : l_1, l_2, \ldots, l_b > \in TIS(M)$, where $S_i \in D_{k+1}\}$ which is consisted of sequences that co-occurred with M. Furthermore, $D_{k+1}||_M$ is also a sequence database, and $\begin{bmatrix} M \\ S \end{bmatrix}$ is a $(k+1)$-domain sequential pattern if and only if S is a sequential pattern of $D_{k+1}||_M$ and $e(S) = e(M)$ with the same minimum support threshold.*

For example, in domain D_1 of Table 5.2, we have $TIS(< (a)(c) >) = \{< (T_1)(T_2)(T_3)(T_4) : 1, 2 >, < (T_1)(T_2)(T_3)(T_4) : 1, 3 >, < (T_5)(T_6)(T_7) : 1, 2 >, < (T_5)(T_6)(T_7) : 1, 3 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 1, 3 >\}$, and propagating $TIS(< (a)(c) >)$ to domain D_2 yields propagated table $D_2||_{<(a)(c)>}$. Table 5.6 is the propagated table $D_2||_{<(a)(c)>}$, where each sequence is very likely to form multi-domain sequential patterns with $< (a)(c) >$ mined from domain D_1. From propagated tables, one could mine sequential patterns having the same number of elements as the propagated sequential pattern and these sequential patterns could be formed as multi-domain sequential patterns. Consider the above example, where the minimum support is set to 3. We can easily find that $< (1)(2) >$ is the sequential pattern of $D_2||_{<(a)(c)>}$ and thus $\begin{bmatrix} (a) & (c) \\ (1) & (2) \end{bmatrix}$ is a 2-domain sequential pattern by compositing $< (a)(c) >$ and $< (1)(2) >$.

Time sequences	Sequences
$< (T_1)(T_2)(T_3)(T_4) >$	$(1,2)(2,3)$
$< (T_1)(T_2)(T_3)(T_4) >$	$(1,2)(6)$
$< (T_5)(T_6)(T_7) >$	$(1,3)(2,4)$
$< (T_5)(T_6)(T_7) >$	$(1,3)(8)$
$< (T_{21})(T_{22})(T_{23})(T_{24}) >$	$(1,2,5)(2,3)$

Table 5.6: Example of propagated table $D_2||_{<(a)(c)>}$.

Note that even though PropagatedMine successfully prevents mining sequential patterns in each domain, however, the cost of some redundant mining of propagated tables can be further reduced. For example, some patterns mined in propagated tables $D_2||_{<(a)>}$ and $D_2||_{<(c)>}$ are the same as patterns mined in propagated table $D_2||_{<(a)(c)>}$. This is due to that the time-instance set of $< (a)(c) >$ is contained in both time-instance sets of $< (a) >$ and $< (c) >$. Consequently, sequences in propagated table $D_2||_{<(a)(c)>}$ also include some sequences in propagated table $D_2||_{<(a)>}$ and $D_2||_{<(c)>}$. Therefore, only sequential patterns with their length being one should be propagated to other domains. In other words, only time-instance sets of the top-level nodes (referred to as *atomic patterns*) in lattice structures are propagated. After obtained, propagated tables are viewed as transaction databases. Consequently, given a propagated table, by utilizing frequent itemset algorithms in [1][2][80][25], we could generate the corresponding multi-domain sequential patterns. We now analyze some important properties of the propagated table. With these properties of propagated tables, the lattice structure in the starting domain is used to determine multi-domain sequential patterns whose length is larger than one. The details of generating multi-domain sequential patterns are described later.

Property of the propagated table of atomic patterns: Suppose that P is a k-domain sequential pattern (i.e., $P \in SP_k$) with $|P| = 1$. $\begin{bmatrix} P \\ \beta \end{bmatrix}$ is a multi-domain sequential pattern across $(k+1)$-domain sequence databases (i.e., $D_1, D_2, \ldots,$ and D_{k+1}) with a minimum support of δ if and only if β is a frequent itemset in propagated table $D_{k+1}||_P$ with the same minimum support δ.

Property of antimonotone with multiple domains: If M is a k-domain sequential pattern (i.e., across $D_1, D_2, \ldots,$ and D_k), k-domain sequences contained by M are also k-domain sequential patterns.

Based on the antimonotone property, algorithm PropagatedMine generates candidate multi-domain sequential patterns in a level-by-level manner. However, in the propagated domain, sequential patterns are also generated level by level according to the number of sequence elements. The detailed steps for deriving multi-domain sequential patterns are described below.

Step 1: Derive atomic patterns across $(k+1)$ domains

Let SP_k be the set of multi-domain sequential patterns across k domains. When deriving atomic patterns across $(k+1)$ domains, the corresponding frequent itemsets can be derived from the propagated tables of each atomic pattern in SP_k. Through the property of propagated table of atomic patterns, those

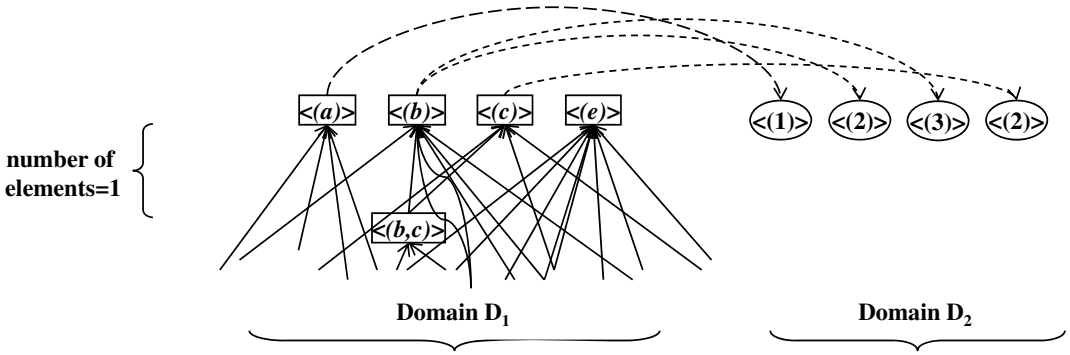


Figure 5.5: Example of generating atomic patterns in domain D_2.

frequent items mined from propagated tables are merged with atomic patterns in SP_k to derive atomic patterns across $(k+1)$ domains. Consider the sequence databases across two domains in Table 5.2 as an example, where sequential patterns of domain D_1 are represented as a lattice structure. We could derive atomic patterns in domain D_2 and thus generate their corresponding multi-domain sequential patterns by propagating the time-instance sets of atomic patterns in domain D_1 (i.e., the top-level nodes) to domain D_2. Specifically, in Figure 5.5, for each atomic pattern in D_1, there are *interdomain links* representing that these two patterns are able to form multi-domain sequential patterns. Consequently, we have $\begin{bmatrix} (a) \\ (1) \end{bmatrix}$, $\begin{bmatrix} (b) \\ (2) \end{bmatrix}$, $\begin{bmatrix} (b) \\ (3) \end{bmatrix}$, and $\begin{bmatrix} (c) \\ (2) \end{bmatrix}$ in the above example, and they are obviously also atomic patterns.

Step 2: Derive $(k+1)$-domain sequential patterns with one element

This step involves deriving $(k+1)$-domain sequential patterns with one element. Assume that k-domain sequential pattern P across k-domain sequence databases (i.e., $D_1, D_2, \ldots,$ and D_k) and that there is only one element in P (i.e., $e(P) = 1$). The intradomain links in the lattice structure for domain k can be followed to find two multi-domain sequential patterns (e.g., X and Y, which are the components of P). The corresponding multi-domain sequential patterns in domain $k+1$ are found by traversing interdomain links of X and Y. According to the antimonotone property, if there exists any corresponding sequential patterns of X or Y in domain $k+1$, they must have been discovered due to $X \sqsubseteq P$ and $Y \sqsubseteq P$. Hence, the corresponding sequential patterns of P in domain $k+1$ are generated from the union of all the multi-domain sequential patterns found in domain $k+1$. For example, let $P =< (b,c) >$ be a sequential pattern with $e(P) = 1$ in D_1 of Table 5.2. The components of P (i.e., $< (b) >$ and $< (c) >$) can be found from the intradomain links. Following interdomain links of $< (b) >$ and $< (c) >$ in Figure 5.6, yields the multi-domain sequential patterns in domain D_2 (i.e., $\begin{bmatrix} (b) \\ (2) \end{bmatrix}$ and $\begin{bmatrix} (b) \\ (3) \end{bmatrix}$ for $< (b) >$, and $\begin{bmatrix} (c) \\ (2) \end{bmatrix}$ for $< (c) >$). Consequently, two candidates are generated by union operation: $\begin{bmatrix} (b) \\ (2) \end{bmatrix} \cup \begin{bmatrix} (c) \\ (2) \end{bmatrix} =$

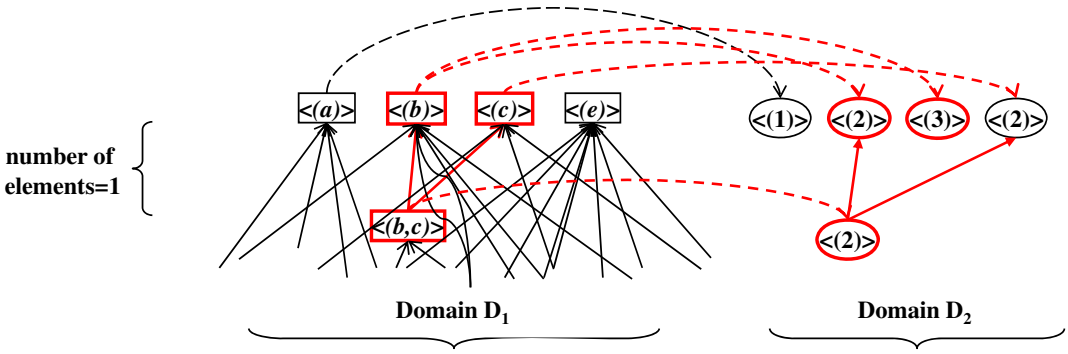


Figure 5.6: An Example of generating sequential patterns with one element in domain D_2.

$\begin{bmatrix} (b,c) \\ (2) \end{bmatrix}$ and $\begin{bmatrix} (b) \\ (3) \end{bmatrix} \cup \begin{bmatrix} (c) \\ (2) \end{bmatrix} = \begin{bmatrix} (b,c) \\ (2,3) \end{bmatrix}$.

Once the candidate multi-domain sequential patterns are obtained, support values of these patterns are examined by checking their time-instance sets (i.e., $Support(\begin{bmatrix} (\alpha) \\ (\beta) \end{bmatrix}) = |TIS(\begin{bmatrix} (\alpha) \\ (\beta) \end{bmatrix})| = |TIS(< (\alpha) >) \cap TIS(< (\beta) >)|$). Given a minimum support of 3, since the support values of $\begin{bmatrix} (b,c) \\ (2) \end{bmatrix}$ and $\begin{bmatrix} (b,c) \\ (2,3) \end{bmatrix}$ are 3 and 2, respectively, only $\begin{bmatrix} (b,c) \\ (2) \end{bmatrix}$ is frequent. Thus, the lattice structure in domain D_2 contains node $< (2) >$, and interdomain links are built between lattice structures in domains D_1 and D_2.

Step 3: Derive $(k+1)$-domain sequential patterns with more than one element

After generating atomic patterns and the $(k+1)$-domain sequential patterns with one element in step1 and step 2 respectively, algorithm PropagatedMine can further generate remaining $(k+1)$-domain sequential patterns in a level-by-level manner by referring to the lattice structure in the last domain propagated (i.e., domain D_k). In this step, PropagatedMine starts deriving from those patterns with two elements due to the antimonotone property. The frequent patterns in the upper levels are found from the intradomain links in the lattice structure of D_k, and the corresponding upper level patterns in the lattice structure of domain D_{k+1} are identified from their interdomain links. Now, the interdomain links of upper level patterns must been established due to the antimonotone property. Before deriving $(k+1)$-domain sequential patterns, it should be determined whether or not to merge the sequential patterns identified in the lattice structure based on their time order. This leads to Definition 9.

Definition 9. (Concatenate operation of TIS) *Let M and N be two multi-domain sequences, where $TIS(M) = \{< TS_1 : l_{11}, l_{12}, \ldots, l_{1e(M)} >, < TS_2 : l_{21}, l_{22}, \ldots, l_{2e(M)} >, \ldots, < TS_m : l_{m1}, l_{m2}, \ldots, l_{me(M)} >\}$, $TIS(N) = \{< TT_1 : k_{11}, k_{12}, \ldots, k_{1e(N)} >, < TT_2 : k_{21}, k_{22}, \ldots, k_{2e(N)} >, \ldots, < TT_n : k_{n1}, k_{n2}, \ldots, k_{ne(N)} >\}$, and TS_i is the time sequence for $i = 1, 2, \ldots, m$ while TT_j is also time sequence for $j = 1, 2, \ldots, n$. The*

Algorithm: PropagatedMine
Input: Sequence databases across n domains $D_1, D_2, \ldots, D_n$, and minimum support δ.
Output: Multi-domain sequential patterns across n domains.
Begin
Apply sequential pattern mining on D_1.
Let SP_1 be the set of sequential patterns mined in D_1.
For each domain D_i, $i = 2, 3, \ldots, n$
For each $P \in SP_{i-1}$
//Step 1
If $|P| = 1$ **Then Begin**
Construct propagation table $D_i||_P$.
Find frequent items in $D_i||_P$ with minimum support δ.
Let FI be the set of frequent items in $D_i||_P$.
For each $Q \in FI$
Append $\begin{bmatrix} P \\ Q \end{bmatrix}$ to SP_i.
Let $TIS(\begin{bmatrix} P \\ Q \end{bmatrix}) = TIS(P) \cap TIS(Q)$.
End
//Step 2
If $e(P) = 1$ **Then Begin**
Let X and Y be two patterns pointed to by intradomain links of P.
For each pattern α pointed to by interdomain links of X
For each pattern β pointed to by interdomain links of Y
If $Support(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}) \geqslant \delta$ **Then Begin**
Construct interdomain links from P to $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$.
Construct intradomain links from $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ to α and β.
Append $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ to SP_i.
End
//Step 3
If $e(P) > 1$ **Then Begin**
Let X and Y be two patterns pointed to by intradomain links of P.
For each pattern α pointed to by interdomain links of X
For each pattern β pointed to by interdomain links of Y
If $Support([(\alpha)(\beta)]) \geqslant \delta$ **Then Begin**
Construct interdomain links from P to $[(\alpha)(\beta)]$.
Construct intradomain links from $[(\alpha)(\beta)]$ to α and β.
Append $[(\alpha)(\beta)]$ to SP_i.
End
Output=SP_n.
End

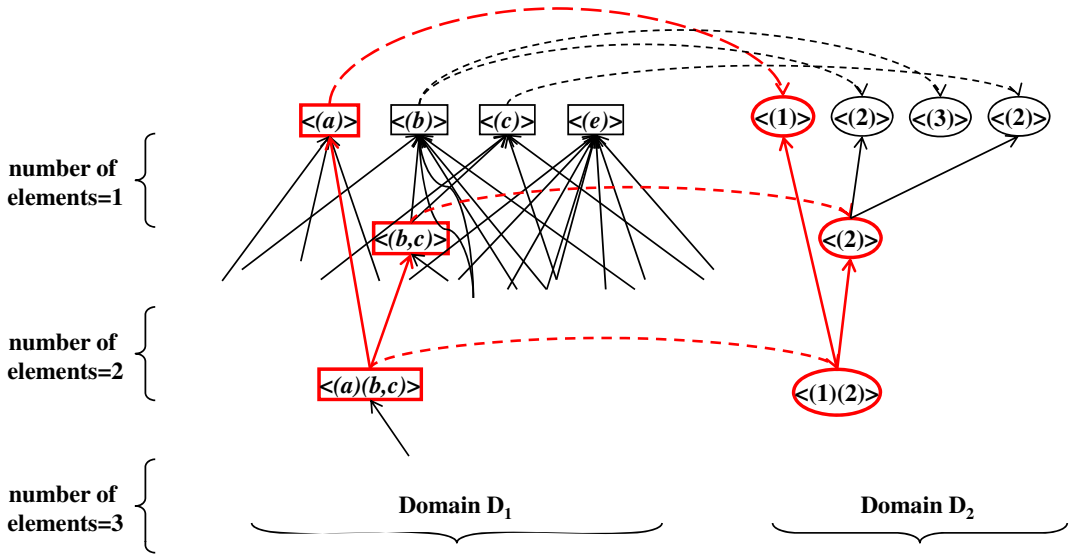


Figure 5.7: Example of generating sequential patterns with more than one element in domain D_2.

concatenation of $TIS(M)$ *and* $TIS(N)$ *is denoted as* $TIS(M) \cap_< TIS(N) = \{< TS_i : l_{i1}, l_{i2}, \ldots, l_{ie(M)}, k_{j1}, k_{j2}, \ldots, k_{je}$ $\}$, *such that* $TS_i = TT_j$ *and* $l_{ie(M)} < k_{j1}$. *In other words,* $TIS(M) \cap_< TIS(N)$ *is the time-instance set of the multi-domain sequence* $[M, N]$, $TIS([M, N])$.

For example, given $M = \begin{bmatrix} (a) \\ (1) \end{bmatrix}$, $N = \begin{bmatrix} (b,c) \\ (2) \end{bmatrix}$, and the sequence database across two domains in Table 5.2, where $TIS(M) = \{< (T_1)(T_2)(T_3)(T_4) : 1 >, < (T_5)(T_6)(T_7) : 1 >, < (T_{10})(T_{12})(T_{13}) : 1 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 1 >\}$, and $TIS(N) = \{< (T_1)(T_2)(T_3)(T_4) : 2 >, < (T_5)(T_6)(T_7) : 2 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 3 >\}$. It can be verified that $TIS(\begin{bmatrix} (a) & (b,c) \\ (1) & (2) \end{bmatrix}) = TIS(M) \cap_< TIS(N) = \{< (T_1)(T_2)(T_3)(T_4) : 1, 2 >, < (T_5)(T_6)(T_7) : 1, 2 >, < (T_{21})(T_{22})(T_{23})(T_{24}) : 1, 3 >\}$.

Assume that pattern $P \in SP_k$ and $e(P) > 1$. Similar to Step 2, we can obtain the components of P, X and Y, by traversing intradomain links among lattice structures across k domains, and the multi-domain sequential patterns pointed to by their interdomain links can be determined. In light of Definition 9, a concatenate operation is considered rather than generating their union as in Step 2. For example, assume pattern $P =< (a)(b,c) >$ in Figure 5.7. The intradomain and interdomain links yield $\begin{bmatrix} (a) \\ (1) \end{bmatrix}$ and $\begin{bmatrix} (b,c) \\ (2) \end{bmatrix}$. Therefore, candidate multi-domain sequential pattern $\begin{bmatrix} (a) & (b,c) \\ (1) & (2) \end{bmatrix}$ is generated, as its support value, $Support(\begin{bmatrix} (a) & (b,c) \\ (1) & (2) \end{bmatrix}) = |TIS(\begin{bmatrix} (a) \\ (1) \end{bmatrix}) \cap_< TIS(\begin{bmatrix} (b,c) \\ (2) \end{bmatrix})| = 3$.

The above steps allow multi-domain sequential patterns across $(k+1)$-domain sequence databases to be derived from k-domain sequential patterns. Algorithm PropagatedMine iteratively repeats the above three steps until all sequence databases are propagated.

Theorem 1. *Algorithm PropagatedMine is able to mine all multi-domain sequential patterns via lattice structures.*

Proof. Mining frequent itemsets in propagated tables reveals multi-domain atomic patterns across other sequence databases. To prove the correctness of Steps 2 and 3, first let P be a k-domain sequential pattern and P' be a $(k+1)$-domain sequential pattern derived from P, where $e(P') = e(P) = 1$ and $|P'| \geqslant |P| > 1$. In other words, $P' = \begin{bmatrix} P \\ Z \end{bmatrix}$, where Z is a frequent itemset in the propagated table $D_{k+1}||_{<(P)>}$. Assume that X and Y are parts of P, and $X \cup Y = P$. Hence, in the lattice structure, we have intradomain links from P to X and Y. In addition, there are interdomain links from X and Y to Z', where Z' is the power set of Z and $Z' \neq \emptyset$. Due to the antimonotone property, all multi-domain sequences contained by P' must also be frequent. In other words, both $\begin{bmatrix} X \\ Z' \end{bmatrix}$ and $\begin{bmatrix} Y \\ Z' \end{bmatrix}$ are frequent. Therefore, the lattice structures can be used to derive all pairs of P and P' while $e(P') = e(P) = 1$. Similarly, when $e(P') = e(P) > 1$, X and Y are parts of P and $TIS(X) \cap_< TIS(Y) = TIS(P)$. Moreover, assume that Z is a frequent itemset in propagated table $D_{k+1}||_{<(P)>}$. Clearly, interdomain links exist from X and Y to Z' in domain D_{k+1}, where Z' is the power set of Z and $Z' \neq \emptyset$. The antimonotone property means that all multi-domain sequences contained by P' must also be frequent. This results in both $[(X, Z')]$ and $[(Y, Z')]$ being frequent. This proof indicates that algorithm PropagatedMine is able to mine all multi-domain sequential patterns. □

5.4.4 Mining Relaxed Multi-domain Sequential Patterns

The above three algorithms are utilized in mining strong multi-domain sequential patterns, where all co-occurred events are from all domains required. Strong multi-domain sequential patterns are very restricted since users may have their minds on analyzing the behavior across domains interested by users. In this chapter, we further develop some mechanisms for mining relaxed multi-domain sequential patterns in which in some time slots, some empty sets are allowed. Note that both the naive algorithm and algorithm IndividualMine could be extended for mining relaxed multi-domain sequential patterns. However, due to the feature of propagation, algorithm PropagatedMine is not able to discover relaxed patterns. In the following, we will discuss how to mine relaxed multi-domaon sequential patterns.

Naive algorithm:

As pointed out early, given a set of sequence databases, algorithm Naive will join these sequence databases into one multi-domain sequence database. With the proper transformed of multi-domain sequence databases, one could generate a sequence database whose events are from all domains. Thus, existing sequential pattern mining algorithms could be utilized to discover sequential patterns. Note that sequential patterns mined are then represented as the form of multi-domain sequential patterns. Hence,

those multi-domain sequential patterns that have some empty sets are directly viewed as relaxed patterns.

Algorithm IndividualMine:

Algorithm IndividualMine performs sequential pattern mining algorithms in each sequence database. After generating all sequential patterns in all domains, in the checking phase, algorithm IndividualMine will check and composite candidate multi-domain sequential patterns with the same number of elements. In order to mine relaxed patterns, all possible compositions of multi-domain sequential patterns from sequential patterns of each domain should be enumerated. For example, assume that one i-domain sequential pattern $P =< P_1, P_2, \ldots, P_l >$, is selected SP_i and $Q =< q_1, q_2, \ldots, q_r >$ is a sequential pattern of domain D_{i+1}. Candidate $(i+1)$-domain sequential patterns generated from P and Q are $\begin{bmatrix} P_1 & P_2 & \ldots & P_l & & & \\ & & & & q_1 & q_2 & \ldots & q_r \end{bmatrix}$, …, $\begin{bmatrix} P_1 & P_2 & \ldots & & P_l & & \\ & & & q_1 & q_2 & \ldots & q_r \end{bmatrix}$ and so on. Note that the number of candidate patterns is denoted as $f(r,l)$ which is formulated as follows:

$$f(r,l) = \begin{cases} 1, & \text{if } r = 0 \\ f(l, r-1) + 2\sum_{i=0}^{r-1} f(i, l-1), & otherwise. \end{cases} \tag{5.1}$$

Obviously, it could be very large when r and l increase. As expected, we could have a large number of candidate multi-domain sequential patterns, degrading the performance of algorithm IndividualMine.

Algorithm PropagatedMine:

By exploring propagation and lattice structures, algorithm PropagatedMine is able to reduce the mining cost. However, algorithm PropagatedMine cannot mine relaxed patterns since propagation needs to obtain time-instance sets of sequential patterns. Empty sets mean that events don't occur and thus there are no any available time information for the empty sets. Thus, it is impossible to derive time-instance sets of empty sets. Consequently, for mining relaxed patterns, algorithm Naive and algorithm IndividualMine should be used.

5.5 Performance Evaluation

To evaluate the performance of our proposed algorithms, we implement a simulation model and conduct extensive experiments. In Section 5.5.1, the simulation model and synthetic datasets are described. Section 5.5.2 is devoted to experimental results.

5.5.1 Simulation Model

We modify the well-known data generator in [3] to generate datasets that include multiple domains. The data generator is broadly used in many studies for evaluating the performance of their proposed methods [36]. The detailed generation process could be referred to [36]. Some parameters are summarized

in Table 5.7 . Explicitly, M denotes the number of domains, D is the number of sequences, C is the average number of elements in a sequence, T is the average number of events in an element and I is the total number of distinct events. The modeling of these parameters are almost the same in [3]. For example, dataset M5D10kC10T5I100 represents that there are 5 domains , each of which contains 10k of sequences, where the average number of elements in a sequence is 10, the average number of items in an element is 5, and the total number of distinct items is 100. For the traditional sequential pattern mining, we use algorithm PrefixSpan which is obtained from the IlliMine project (http://illimine.cs.uiuc.edu/). Algorithm PrefixSpan is used in algorithm Naive and the mining phases of both algorithms IndividualMine and PropagatedMine. Our programs are executed in the platform with the hardware as an Intel 2.4-GHz XEON CPU and 3.5 GB of RAM, and the software as FreeBSD 5.0 and GCC 3.2. We use three performance metrics: the execution time, memory consumption and the number of mined patterns to compare the proposed algorithms.

5.5.2 Experimental Results

Several experiments were conducted to evaluate the performance and memory consumption of the three algorithms. Sensitivity analysis on some important parameters, such as the minimum support, the number of sequences, and the number of domains, is conducted.

Impact of the Minimum Support Threshold

We first investigated the performances of three algorithms with the minimum support varied. For the dataset M2D2kC3T4I200, Figure 5.8 shows the execution time and the memory consumption of three algorithms. It can be seen in Figure 5.8 that the execution time of algorithm IndivudualMine and PropagatedMine is reduced as the minimum support increases. This is due to that with a larger minimum support, the number of sequential patterns in sequence databases is smaller. Furthermore, algorithm PropagatedMine significantly outperforms the other two algorithms in terms of execution time, which demonstrates the advantage of exploring propagation and lattice structures in mining multi-domain sequential patterns. On the other hand, when the minimum support was smaller than 1.5%, algorithm IndividualMine was worse than algorithm Naive. The reason is that with a smaller minimum support, a larger number of sequential patterns are mined in each domain. Thus, algorithm IndividualMine needs

Parameter	Description
M	number of domains
D	number of sequences
C	average number of elements within a sequence
T	average number of items within an element
I	total number of different items

Table 5.7: Parameters used for the data generator.

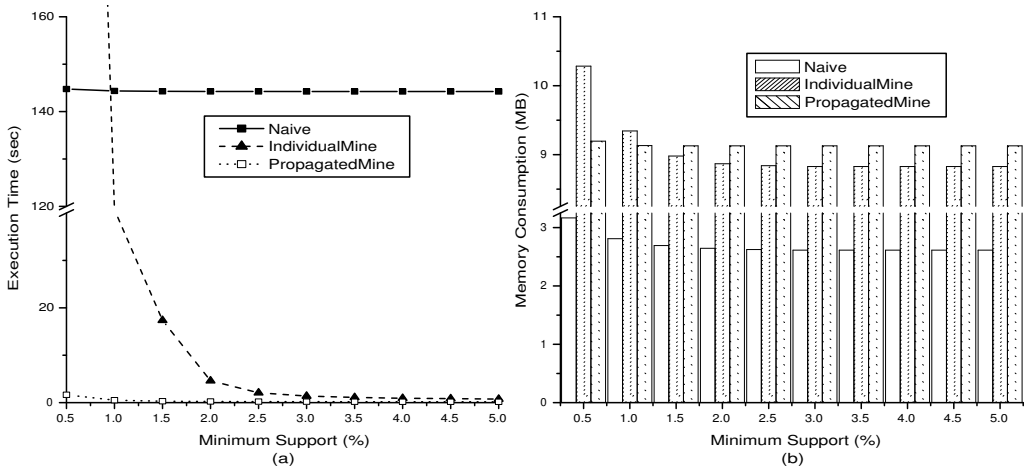


Figure 5.8: Execution times of the three algorithms with various minimum support thresholds.

Number of domains	2	3	4	5
Naive	5.3	206.7	2513.9	21769.7
IndividualMine	126.3	163.9	180.2	181.1
PropagatedMine	0.4	0.6	0.7	0.7

Table 5.8: Execution times of algorithms Naive, IndividualMine, and PropagatedMine with the number of domains varied on D1kC2T3I100.

more time to composite candidate multi-domain sequential patterns and determine their supports. In Naive algorithm, joining operations among sequence databases are costly, which dominates the execution time. As for the memory consumption, algorithm Naive use less memory than algorithms IndividualMine and PropagatedMine. This is due to that both algorithms IndividualMine and PropagatedMine use more memory spaces for storing sequential patterns mined. Algorithm PropagatedMine also needs to store lattice structures, which incurs more memory space than algorithm IndividualMine. On the other hand, algorithm IndividualMine does not need any more memory space for storing sequential patterns. Though algorithm PropagatedMine needs more memory spaces, algorithm PropagatedMine is able to quickly derive multi-domain sequential patterns, which strikes a compromise between memory space and the execution time.

Impact of the Number of Domains

We next examine the impact of domains on the performance of three proposed algorithms. The experiments were conducted on D1kC2T3I100 (referred to as a smaller dataset) and D1kC3T4I200 (referred

Number of domains	2	3	4	5
Naive	57.1	3065.3	53164.9	379118.5
IndividualMine	1052.1	1192.9	1213.9	1214.4
PropagatedMine	2.1	2.4	2.5	2.5

Table 5.9: Execution times of algorithms Naive, IndividualMine, and PropagatedMine with the number of domains varied on D1kC2T4I200.

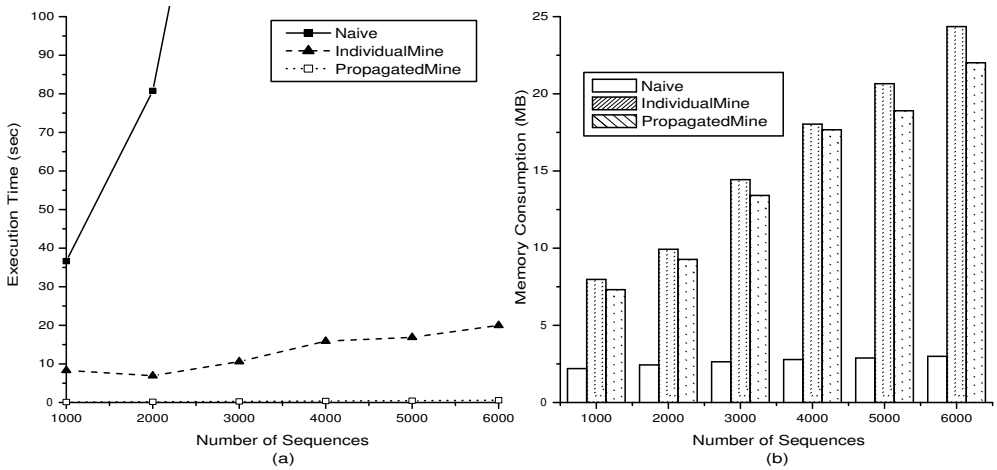


Figure 5.9: Performance of Naive, IndividualMine, and PropagatedMine with the number of sequences varied.

to as a larger dataset). With the minimum support as 0.3%, the execution time with its unit as seconds for these proposed algorithms is shown in Table 5.8 and Table 5.9. From both tables, it can be seen that all three algorithms have a larger execution time when the number of domains increases. In particular, the execution of algorithm Naive drastically increases the execution time. Both algorithms IndividualMine and PropagatedMine have smaller execution time than algorithm Naive. Furthermore, algorithm PropagatedMine outperforms other algorithms in terms of the execution time, showing the advantage of utilizing propagation to reduce the mining cost. In addition, given a larger dataset with more number of events and larger sequence lengths, the execution time of algorithm Naive is worse. On the other hands, algorithm PropagatedMine incurs a smaller execution time than algorithms Naive and IndividualMine, showing the good scalability of algorithm PropagatedMine.

Impact of the Number of Sequences

Experiments with the number of sequences varied are examined, where the number of sequences is from 1000 to 6000 and other parameters are M2C3T3I200. With a given minimum support was 1%, Figure 5.9 shows the execution time of all algorithms. As can be seen in Figure 5.9, the execution of all three algorithms increases as the number of sequences increases. Notice that the execution time of algorithm Naive is significantly increasing when the number of sequences is lager than 2000. Thus, to compare algorithms IndividualMine and PropagatedMine, we only put the execution time of algorithms IndividualMine and PropagatedMine. By exploring lattice structures, PropagatedMine should mine only atomic patterns, from which other patterns are derived accordingly. As a result, the execution time of PropagatedMine slightly increases with the number of sequences. Note that the execution time of algorithm PropagatedMine is very smaller compared with algorithms IndividualMine and Naive. However, both algorithms IndividualMine and PropagatedMine need more memory space for storing sequential patterns mined. Thus, it can be seen in Figure 5.9 that both algorithms IndividualMine and PropagatedMine have

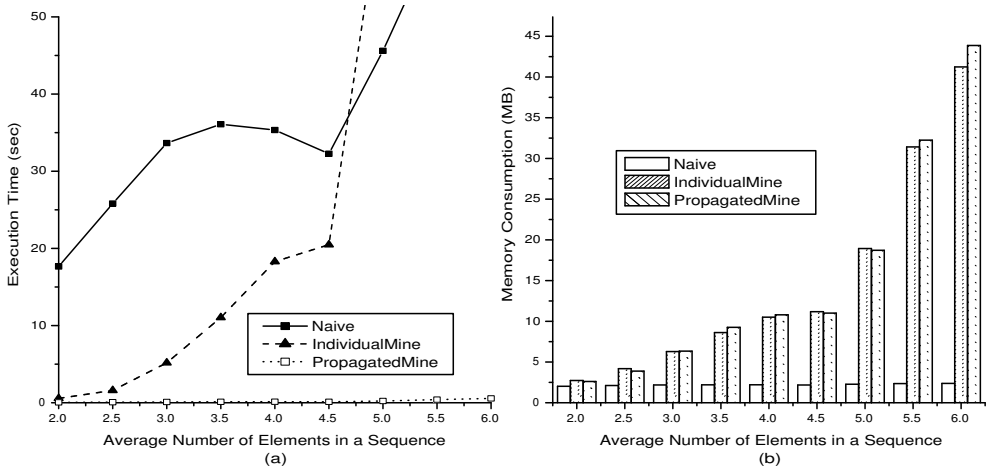


Figure 5.10: Performance of Naive, IndividualMine, and PropagatedMine with the average number of elements within a sequence varied.

a larger memory consumption than algorithm Naive. This also agrees that algorithm Naive is bounded by execution time, and algorithms IndividualMine and PropagatedMine are bounded by memory spaces.

Impact of the Average Number of Elements within a Sequence

In this section, we investigate the performance of Naive, IndividualMine, and PropagatedMine with the average number of elements within a sequence varied. Without loss of generality, the minimum support threshold is set to 1% and the other parameters in the dataset are M2D1kT3I200. Figure 5.10 shows experimental results of Naive, IndividualMine, and PropagatedMine. Clearly, the execution time of mining multi-domain sequential patterns increases with the average number of elements within a sequence. Note that algorithm IndividualMine even performs worse than algorithm Naive when the average number of elements in a sequence is larger than 4.7. The reason is that IndividualMine mines a large number of sequential patterns in each domain and spends more costs to composite candidate multi-domain sequential patterns. The above observation is also proved in Figure 5.11, where algorithm IndividualMine generates a larger number of sequential patterns propagated than algorithm PropagatedMine. Note that, the number of patterns propagated in algorithm IndividualMine is the number of patterns discovered in the starting domain. Figure 5.10 (b) also indicates that though algorithm PropagatedMine has a smaller execution time, algorithm PropagatedMine needs more memory spaces to store lattice structure.

Impact of the Average Number of Items within an Itemset

The average number of items within an itemset generally impacts on the performance of sequential pattern mining. Thus, we investigate the effect of varying the average number of items within an itemset. The minimum support was set to 1% and we used the dataset M2D1kC3I200. The execution time and memory consumption with the average number of items in an itemset varied are shown in Figure 5.12. As can be seen that in Figure 5.12, PropagatedMine performs the best in terms of the execution time. When the

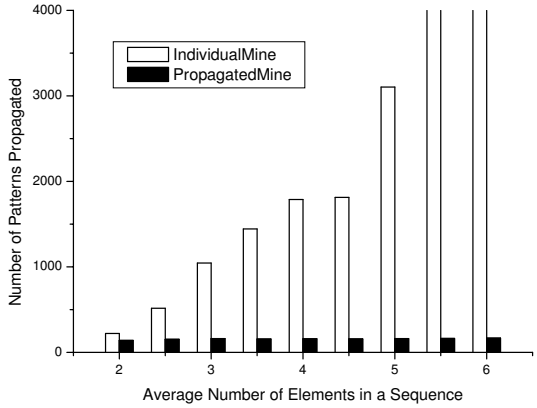


Figure 5.11: Number of patterns propagated in IndividualMine and PropagatedMine with the average number of elements within a sequence varied.

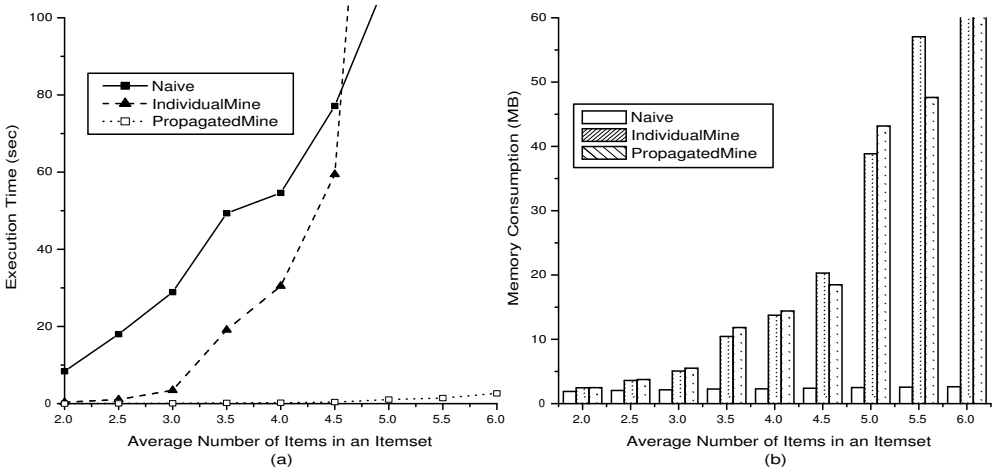


Figure 5.12: Performance of Naive, IndividualMine, and PropagatedMine with the average number of items within an itemset varied.

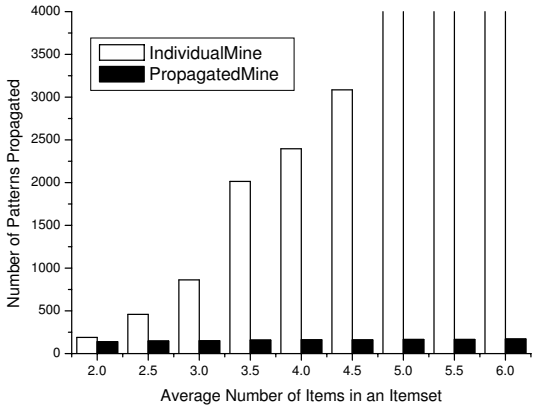


Figure 5.13: Number of patterns propagated in IndividualMine and PropagatedMine with the average number of items within an itemset varied.

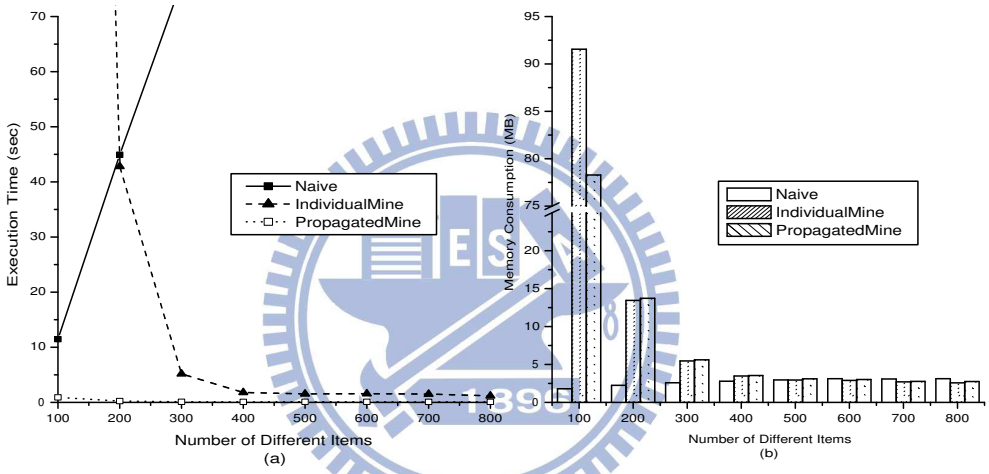


Figure 5.14: Performance of Naive, IndividualMine, and PropagatedMine with the number of different items varied.

average number of items in an itemset is smaller, the execution time of IndividualMine is smaller than that of Naive. However, if there is a large number of items within an itemset, IndividualMine performs worse than Native since algorithm IndividualMine has a larger number of patterns mined, which incurs a considerable cost in the checking phase. Figure 5.13 demonstrates that PropagatedMine is better than IndividualMine because sequential patterns mined in the starting domain are much smaller than that of algorithm IndividualMine. In algorithm PropagatedMine, only atomic patterns are mined and thus the number of patterns mined in the starting domain is equal to the number of atomic patterns. Consequently, by exploring lattice structures, algorithm PropagatedMine outperforms the other algorithms in terms of the execution time.

Impact of the Number of Items

We next investigate the impact of the total number of items, where a minimum support is set to 1% and other parameters are set as M2D1kC3T4. Figure 5.14 shows the execution times and memory consumption of Naive, IndividualMine, and PropagatedMine. It can be seen in Figure 5.14 that both

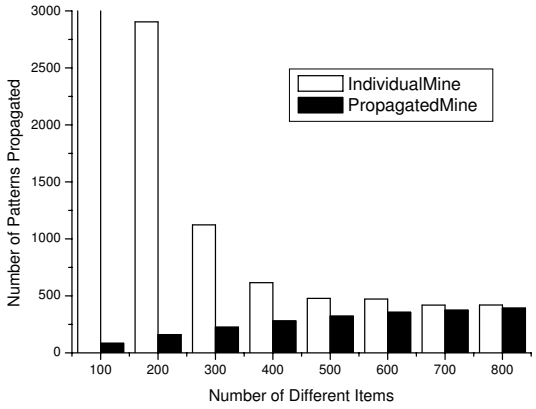


Figure 5.15: Number of patterns propagated in IndividualMine and PropagatedMine with the number of different items varied.

IndividualMine and PropagatedMine have a smaller execution time than Naive as the number of items increases. When the number of items is larger, the probability of being frequent for each item is smaller with the same setting in D1kC3T4. Figure 5.15 depicts the number of patterns with the number of items varied. As can be seen in Figure 5.15, PropagatedMine has a smaller number of patterns derived, which demonstrates the advantage of using lattice structures for discovering multi-domain sequential patterns.

Impact of the Propagation Order for PropagatedMine

Since algorithm PropagatedMine explores propagation on mining multi-domain sequential patterns, we now get insight into the impact of propagation orders on performance of algorithm PropagatedMine. As pointed out early, algorithm PropagatedMine first selects a starting domain and then performs sequential pattern mining. Based on the mining results, a lattice structure is built. Clearly, one should judiciously determine the starting domain in algorithm PropagatedMine. Intuitively, selecting a domain with a smaller number of sequential patterns is good to reduce the size of lattice structures, thereby improving the performance of algorithm PropagatedMine. In this experiment, we conduct experiments on different propagation orders.. Figure 5.16 shows the execution time of algorithm PropagatedMine with various propagation orders, where the value in the x-axle is the propagation order used. For example, 12435 indicates that the algorithm PropagatedMine starts with D_1, and then propagates to D_2, D_4, D_3 and D_5. As can be seen in Figure 5.16, selecting domain D_1 as a starting domain is better since algorithm PropagatedMine has a smaller execution time and memory consumption. This implies that sequential patterns in D_1 has the minimal number of sequential patterns. Table 5.10 depicts the number of sequential patterns in each domain and the number of sequential patterns in D_1 is the smallest among other domains. Furthermore, in Figure 5.16, propagation order 12435 incurs the smallest execution time of algorithm PropagatedMine. This observation gives a guideline in which a good propagation order is determined as an ascending order of the number of sequential patterns in sequence databases. Note that there are many ways (e.g., sampling) to approximate the number of sequential patterns in each domain. Thus, according

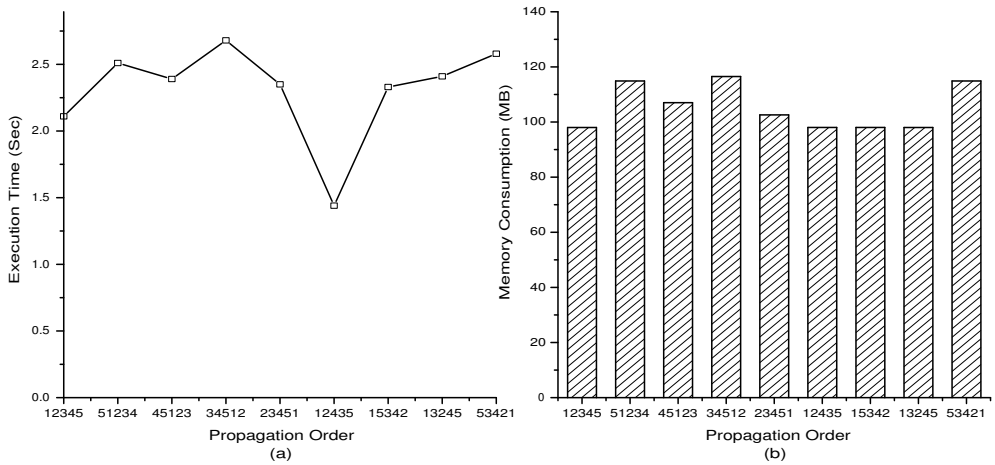


Figure 5.16: Performance of PropagatedMine with varied propagation order.

Domains	D_1	D_2	D_3	D_4	D_5
Number of Sequential patterns	24982	25507	28204	27654	28560

Table 5.10: Number of sequential patterns mined in each domain.

to the guideline above, one could determine a good propagation order for algorithm PropagatedMine.

5.6 Conclusions

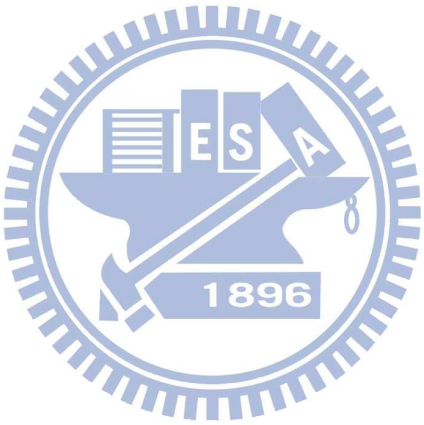
This chapter addresses a novel mining task: the multi-domain sequential pattern mining problem. Multi-domain sequential patterns are of practical interest and use since they clearly reflect the relations of domains hidden in user's behavior. We designed algorithm Naive as a baseline algorithm and two efficient algorithms, IndividualMine and PropagatedMine, to solve this problem. Specifically, in algorithm IndividualMine, each domain individually performs sequential pattern mining and then candidate multi-domain sequential patterns are generated by combining all mined sequential patterns in each domain. Finally, by checking the time-instance sets of candidate multi-domain sequential patterns, the multi-domain sequential patterns are discovered without scanning databases. In order to reduce the mining cost of discovering sequential patterns in each domain, algorithm PropagatedMine first mines sequential patterns in a starting domain. Propagated tables are then constructed to discover the candidate multi-domain sequential patterns. Note that by using propagated tables, only sequential patterns that are likely to form multi-domain sequential patterns are extracted. Algorithm PropagatedMine further explores lattice structures to reduce the number of patterns propagated. A comprehensive experimental study is conducted and experimental results show that both algorithms IndividualMine and PropagatedMine are able to quickly mine multi-domain sequential patterns compared with algorithm Naive. By exploring propagation and lattice structures, algorithm PropagatedMine outperforms other algorithms in terms of execution times.

Chapter 6

Conclusion

In this dissertation, we develop a series of research works for Apps usage behavior mining and explore patterns mined from multiple categories of Apps. We select useful features from all sensor readings and Apps usage relations to perform Apps usage prediction. Then, two algorithms are proposed to estimate users dynamic preferences of Apps. Finally, the multi-domain sequential patterns are discovered to formulate the Apps usage pattern in the category level. In the first work, we focus on collecting all sensor readings and Apps usage transitions, and performing kNN classification to predict Apps usage. Two main type of features are proposed. The explicit feature consists of 1) device sensors, 2) environmental sensors, and 3) personalized sensors. The implicit feature models the Apps usage transitions. Two implicit feature discovery algorithms are proposed to explore the implicit features for training and test purposes. Then, a personalized feature selection algorithm is proposed to measure which features are useful for different users usage behavior. In the second work, we implement an AppNow widget on Android based smartphones. We further reduce the used features into only the temporal information, and build a temporal profile for each App. As the observation of Apps usage behavior, we realized that the Apps usage could have a specific usage period. We adopt Fourier transform to discover the usage periods for each App. The temporal profile is thus modelled by the discovered usage periods. The AppNow widget will predict the Apps usage by comparing the temporal profile of every App and current time to see which Apps have higher probability to be launched. In the third work, we propose a novel dynamic preference prediction problem which is to quantize and predict users preference according to their Apps usage counts. Two algorithms are proposed. The mode-based prediction (MBP) considers the usage status of each App to calculate its preference. The reference-based prediction (RBP) calculate a reference point for each App. The preference of an App is thus estimated by comparing the reference point and the real usage counts. In the forth work, a novel data mining task: mining multi-domain sequential patterns is proposed. The multi-domain sequential pattern represents the usage transition of Apps in the category level. Two algorithms are proposed to solve this problem. The individualMine algorithm discovers sequential patterns in each

domain and combines those sequential patterns into one single domain. The propagatedMine algorithm only performs sequential pattern mining in one starting domain and propagates the discovered sequential patterns to the next domains. We design several operations when propagating patterns.



Bibliography

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM International Conference on Management of Data (SIGMOD)*, pages 207–216, 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns. In *Proceedings of the 1995 IEEE International Conference on Data Engineering (ICDE)*, pages 3–14, 1995.

[4] Driss Choujaa andC Naranker Dulay. Predicting Human Behaviour from Selected Mobile Phone Data Points. In *Proc. of UbiComp*, pages 105–108, 2010.

[5] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential Pattern Mining Using A Bitmap Representation. In *Proceedings of the 2002 ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 429–435, 2002.

[6] Mete Celik, Shashi Shekhar, James P. Rogers, James A. Shine, and Jin Soung Yoo. Mixed-Drove Spatio-Temporal Co-occurence Pattern Mining: A Summary of Results. In *Proceedings of the 2006 IEEE International Conference on Data Mining (ICDM)*, pages 119–128, 2006.

[7] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[8] Gong Chen, Xindong Wu, and Xingquan Zhu. Sequential Pattern Mining in Multiple Streams. In *Proceedings of the 2005 IEEE International Conference on Data Mining (ICDM)*, pages 585–588, 2005.

[9] Lei Chen, M. Tamer zsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proc. of SIGMOD*, pages 491–502, 2005.

[10] Shuo Chen, Joshua L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 714–722, 2012.

[11] Hong Cheng, Xifeng Yan, and Jiawei Han. IncSpan: Incremental Mining of Sequential Patterns in Large Database. In *Proceedings of the 2004 ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 527–532, 2004.

[12] Ding-An Chiang, Yi-Hsin Wang, and Shao-Ping Chen. Analysis on repeat-buying patterns. *Knowl.-Based Syst.*, 23(8):757–768, 2010.

[13] Ding-Ying Chiu, Yi-Hung Wu, and Arbee L. P. Chen. An Efficient Algorithm for Mining Frequent Sequences by A New Strategy without Support Counting. In *Proceedings of the 2004 IEEE International Conference on Data Engineering (ICDE)*, pages 375–386, 2004.

[14] Chung-Wen Cho, Yi-Hung Wu, and Arbee L. P. Chen. Effective Database Transformation and Efficient Support Computation for Mining Sequential Patterns. In *Proceedings of the 2005 International Conference Database Systems for Advanced Applications (DASFAA)*, pages 163–174, 2005.

[15] Trinh Minh Tri Do, Jan Blom, and Daniel Gatica-Perez. Smartphone usage in the wild: a large-scale analysis of applications and context. In *Proceedings of the 13th International Conference on Multimodal Interfaces, ICMI 2011, Alicante, Spain, November 14-18, 2011*, pages 353–360, 2011.

[16] Yuxiao Dong, Qing Ke, Jun Rao, Bai Wang, and Bin Wu. Random walk based resource allocation: Predicting and recommending links in cross-operator mobile communication networks. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on, Vancouver, BC, Canada, December 11, 2011*, pages 358–365, 2011.

[17] Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V. Chawla, Jinghai Rao, and Huanhuan Cao. Link prediction and recommendation across heterogeneous social networks. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 181–190, 2012.

[18] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The yahoo! music dataset and kdd-cup'11. In *KDD-Cup Workshop*, 2011.

[19] Themis P. Exarchos, Markos G. Tsipouras, Costas Papaloukas, and Dimitrios I. Fotiadis. A Two-Stage Methodology for Sequence Classification Based on Sequential Pattern Mining and Optimization. *Data and Knowledge Engineering*, 66(3):467–487, 2008.

[20] Hongliang Fei, Ruoyi Jiang, Yuhao Yang, Bo Luo, and Jun Huan. Content based social behavior prediction: a multi-task learning approach. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 995–1000, 2011.

[21] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. In *Proceedings of the 1999 International Conference on Very Large Data Bases (VLDB)*, pages 223–234, 1999.

[22] Valerie Guralnik and George Karypis. Parallel Tree-Projection-Based Sequence Mining Algorithms. *Parallel Computing*, 30(4):443–472, 2004.

[23] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. Social media recommendation based on people and tags. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 194–201, 2010.

[24] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. In *Proceedings of the 2000 ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 355–359, 2000.

[25] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM International Conference on Management of Data (SIGMOD)*, pages 1–12, 2000.

[26] Jen-Wei Huang, Chi-Yao Tseng, Jian-Chih Ou, and Ming-Syan Chen. Pisa: Progressive Mining of Sequential Patterns. In *Proceedings of the ACM 2006 International Conference on Information and Knowledge Management (CIKM)*, pages 850–851, 2006.

[27] Kuo-Yu Huang, Chia-Hui Chang, Jiun-Hung Tung, and Cheng-Tao Ho. COBRA: Closed Sequential Pattern Mining Using Bi-phase Reduction Approach. In *Proceedings of the 2006 International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 280–291, 2006.

[28] Shi-Ming Huang, Chih-Fong Tsai, David C. Yen, and Yin-Lin Cheng. A hybrid financial analysis model for business failure prediction. *Expert Syst. Appl.*, 35(3):1034–1040, 2008.

[29] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[30] Hoyoung Jeung, Qing Liu, Heng Tao Shen, and Xiaofang Zhou. A hybrid prediction model for moving objects. In *Proc. of ICDE*, pages 70–79, 2008.

[31] Daisuke Kamisaka, Shigeki Muramatsu, Hiroyuki Yokoyama, and Takeshi Iwamoto. Operation prediction for context-aware user interfaces of mobile phones. In *2009 Ninth Annual International Symposium on Applications and the Internet*, pages 16–22, 2009.

[32] Eiman Kanjo, Jean Bacon, David Roberts, and Peter Landshoff. MobSens: Making Smart Phones Smarter. *IEEE Pervasive Computing*, 8(4):50–57, 2009.

[33] Jacob Kogan. Feature selection over distributed data streams through convex optimization. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012*, pages 475–484, 2012.

[34] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 447–456, 2009.

[35] Hye-Chung Kum, Joong Hyuk Chang, and Wei Wang. Sequential Pattern Mining in Multi-Databases via Multiple Alignment. *Data Mining and Knowledge Discovery*, 12(2-3):151–180, 2006.

[36] Hye-Chung Kum, Joong Hyuk Chang, and Wei Wang. Benchmarking the Effectiveness of Sequential Pattern Mining Methods. *Data and Knowledge Engineering*, 60(1):30–50, 2007.

[37] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. Temporal diversity in recommender systems. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 210–217, 2010.

[38] Po-Ruey Lei, Tsu-Jou Shen, Wen-Chih Peng, and Ing-Jiunn Su. Exploring spatial-temporal trajectory model for location prediction. In *12th IEEE International Conference on Mobile Data Management, MDM 2011, Luleå, Sweden, June 6-9, 2011, Volume 1*, pages 58–67, 2011.

[39] Neal Lesh, Mohammed Javeed Zaki, and Mitsunori Ogihara. Mining Features for Sequence Classification. In *Proceedings of the 1999 ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 342–346, 1999.

[40] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S. Glance, and Matthew Hurst. Patterns of cascading behavior in large blog graphs. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*, 2007.

[41] Zhung-Xun Liao, Po-Ruey Lei, Tsu-Jou Shen, Shou-Chung Li, and Wen-Chih Peng. Mining temporal profiles of mobile applications for usage prediction. In *12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, December 10, 2012*, pages 890–893, 2012.

[42] Zhung-Xun Liao and Wen-Chih Peng. Exploring Lattice Structures in Mining Multi-domain Sequential Patterns. In *Proceedings of the 2007 International Conference on Scalable Information Systems (InfoScale)*, pages 334–339, 2007.

[43] Zhung-Xun Liao, Wen-Chih Peng, and Xing-Yuan Hu. Mining Multi-domain Sequential Patterns. In *Workshop on Software Engineering, Databases, and Knowledge Discovery, International Computer Symposium (ICS)*, pages 334–339, 2006.

[44] Zhung-Xun Liao, Wen-Chih Peng, and Philip S. Yu. Mining usage traces of mobile applications for dynamic preference prediction. In *17th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2013, Gold Coast, Australia, April 13-17, 2013*, 2013.

[45] David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pages 556–559, 2003.

[46] Eric Hsueh-Chan Lu, Wang-Chien Lee, and Vincent Shin-Mu Tseng. A framework for personal mobile commerce pattern mining and prediction. *IEEE Trans. Knowl. Data Eng.*, 24(5):769–782, 2012.

[47] Dimitrios Lymberopoulos, Peixiang Zhao, Arnd Christian König, Klaus Berberich, and Jie Liu. Location-aware click prediction in mobile local search. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 413–422, 2011.

[48] Florent Masseglia, Pascal Poncelet, and Maguelonne Teisseire. Incremental Mining of Sequential Patterns in Large Databases. *Data and Knowledge Engineering*, 46(1):97–121, 2003.

[49] M. Matsumoto, R. Kiyohara, H. Fukui, and M. Numao. Proposition of the context-aware interface for cellular phone operations. In *5th International Conference on Networked Sensing Systems, June 17-19, 2008*, pages 233–233, 2008.

[50] Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 637–646, 2009.

[51] Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proc. of KDD*, pages 637–646, 2009.

[52] Bernard Ostle and Linda Catron Malone. Statistics in research: basic concepts and techniques for research workers, 1988.

[53] Nick Pears, Daniel Jackson, and Patrick Olivier. Smart Phone Interaction with Registered Displays. *IEEE Pervasive Computing*, 8(2):14–21, 2009.

[54] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In *Proceedings of the 2001 IEEE International Conference on Data Engineering (ICDE)*, pages 215–224, 2001.

[55] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[56] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. Multi-Dimensional Sequential Pattern Mining. In *Proceedings of the 2001 ACM International Conference on Information and Knowledge Management (CIKM)*, pages 81–88, 2001.

[57] Bodhi Priyantha, Dimitrios Lymberopoulos, and Jie Liu. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. *IEEE Pervasive Computing*, 10(2):12–15, 2011.

[58] Daniele Quercia, Giusy Di Lorenzo, Francesco Calabrese, and Carlo Ratti. Mobile phones and outdoor advertising: Measurable advertising. *IEEE Pervasive Computing*, 10(2):28–36, 2011.

[59] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[60] J. Rissanen. Hypothesis selection and testing by the mdl principle. *The Computer Journal*, 42:260–269, 1999.

[61] Pierre-Yves Rolland. FlExPat: Flexible Extraction of Sequential Patterns. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)*, pages 481–488, 2001.

[62] Salvatore Scellato, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Andrew T. Campbell. Nextplace: A spatio-temporal prediction framework for pervasive systems. In *Pervasive Computing - 9th International Conference, Pervasive 2011, San Francisco, CA, USA, June 12-15, 2011. Proceedings*, pages 152–169, 2011.

[63] Kent Shi and Kamal Ali. Getjar mobile application recommendations with very sparse datasets. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 204–212, 2012.

[64] Choonsung Shin, Jin-Hyuk Hong, and Anind K. Dey. Understanding and prediction of mobile application usage for smart phones. In *The 2012 ACM Conference on Ubiquitous Computing, Ubicomp '12, Pittsburgh, PA, USA, September 5-8, 2012*, pages 173–182, 2012.

[65] Ramakrishnan Srikant and Rakesh Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 1996 International Conference on Extending Database Technology (EDBT)*, pages 3–17, 1996.

[66] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proc. of SenSys*, pages 85–98, 2010.

[67] Alessandra Toninelli, Rebecca Montanari, Ora Lassila, and Deepali Khushraj. What's on Users' Minds? Toward a Usable Smart Phone Security Model. *IEEE Pervasive Computing*, 8(2):32–39, 2009.

[68] Petre Tzvetkov, Xifeng Yan, and Jiawei Han. TSP: Mining Top-K Closed Sequential Patterns. *Knowledge Information System*, 7(4):438–457, 2005.

[69] Michail Vlachos, Philip Yu, and Vittorio Castelli. On periodicity detection and structural periodic similarity. In *Proc. of SDM*, 2005.

[70] Jianyong Wang and Jiawei Han. BIDE: Efficient Mining of Frequent Closed Sequences. In *Proceedings of the 2004 IEEE International Conference on Data Engineering (ICDE)*, pages 79–90, 2004.

[71] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 723–732, 2010.

[72] Bo Yan and Guanling Chen. Appjoy: personalized mobile application discovery. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), Bethesda, MD, USA, June 28 - July 01, 2011*, pages 113–126, 2011.

[73] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *The 10th International Conference on Mobile Systems, Applications, and Services, MobiSys'12, Ambleside, United Kingdom - June 25 - 29, 2012*, pages 113–126, 2012.

[74] Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan: Mining Closed Sequential Patterns in Large Databases. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM)*, 2003.

[75] Jiong Yang, Wei Wang, Philip S. Yu, and Jiawei Han. Mining Long Sequential Patterns in A Noisy Environment. In *Proceedings of the 2002 ACM International Conference on Management of Data (SIGMOD)*, pages 406–417, 2002.

[76] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation: a contest between satisfaction and temptation. In *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pages 395–404, 2013.

[77] Chuang-Wen You, Chih-Chiang Wei, Yi-Ling Chen, Hao-Hua Chu, and Ming-Syan Chen. Using mobile phones to monitor shopping time at physical stores. *IEEE Pervasive Computing*, 10(2):37–43, 2011.

[78] Chung-Ching Yu and Yen-Liang Chen. Mining Sequential Patterns from Multidimensional Sequence Data. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):136–140, 2005.

[79] Mohammed Javeed Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[80] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New Algorithms for Fast Discovery of Association Rules. In *Proceedings of the 1997 ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 283–286, 1997.

[81] Chun Zhu and Weihua Sheng. Motion- and location-based online human daily activity recognition. *Pervasive and Mobile Computing*, 7(2):256–269, 2011.