Research Article

Efficiently and Effectively Mining Time-Constrained Sequential Patterns of Smartphone Application Usage

Kuo-Wei Hsu

Department of Computer Science, National Chengchi University, No. 64, Sec. 2, Zhi Nan Rd., Wen Shan District, Taipei 11605, Taiwan

Correspondence should be addressed to Kuo-Wei Hsu; kwhsu@nccu.edu.tw

Received 9 August 2016; Revised 14 November 2016; Accepted 1 December 2016; Published 16 January 2017

Academic Editor: Sergio Mascetti

Copyright © 2017 Kuo-Wei Hsu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Today, we have the freedom to install and use all kinds of applications on smartphones, thanks to the development of mobile communication and computing technologies. Undoubtedly, the system and application developers are eager to know how we use the applications on our smartphones in our daily life and so are the researchers. In this paper, we present our work on developing a pattern mining algorithm and applying it to smartphone application usage log collected from tens of smartphone users for several years. Our goal is to mine the sequential patterns each of which presents a series of application uses and satisfies a constraint on the maximum time interval between two application uses. However, we cannot mine such patterns by general algorithms and will miss some patterns by using the widely used implementation of the advanced algorithm specifically designed for time-constrained sequential pattern mining. We not only present an algorithm that can efficiently and effectively mine the patterns in which we are interested but also discuss and visualize the mined patterns. Our work could potentially support the related studies.

1. Introduction

During the past two decades, there has been a substantial development in communication infrastructure and mobile technologies. The development, along with that in the improved design of hardware and software of mobile devices, brings us mobile phones that are no longer restricted to voice communication but provide extended functionality. These are smartphones. As more features are introduced and more applications are developed, smartphones have touched almost all aspects of our life.

The growth of the market and industry for smartphone systems and applications is so rapid and strong that there is an urgent need for studies that will help the system and application developers better understand how users use their smartphones and applications. The system and application developers are interested in having patterns of smartphone application usage that can help them build better user experiences. For example, analyzing application usage patterns can be helpful to the memory, power, or network management of smartphones. The more and the better the developers know about the users, the higher the chance that they can identify potential improvements on user interfaces of applications or

identify promising integrations of functions of applications. Suppose that we have a pattern indicating that a user would probably use an application in three minutes after using another application. This pattern could possibly be used by the system to rearrange the icon of the application that would be used later to save the user's searching time. This pattern could possibly be used by the system to prelaunch in the background the application that would be used later to save the user's waiting time. Additionally, the researchers feel the need for studies that will help them gain a better understanding of users' behaviors, such as how users use their smartphones and/or applications installed and running on their smartphones. In [1], Zhou et al. stated the utility of web usage log mining; from our perspective, it can be generalized as follows: to provide intelligent services to users, it is usually necessary to model users' behaviors and a promising approach is to mine log. In [2], Mabroukeh and Ezeife also stated the utility; likewise, from our perspective, it can be generalized to the following: usage mining is an important application concerned with finding patterns by extracting knowledge from log.

In this paper, we present our work on developing an algorithm to mine time-constrained sequential patterns and

applying it to the data reflects the daily application uses done by smartphone users. Mining smartphone application usage log is close to mining web usage log [3], but the existing algorithms can hardly be directly used due to the fundamental differences between using smartphone applications and browsing web pages. We consider the context of using smartphones and define a usage session, or simply a session, as a period of time during which a user uses applications uninterruptedly on his or her smartphone. The patterns which we are interested in are not general sequential patterns but those with a constraint on the maximum time interval between two application uses in a session. Time is important in behavior analysis, but it is not considered in the general sequential pattern mining algorithms.

If the time constraint is not considered, a large number of frequent yet uninteresting patterns will be mined. For example, if we do not consider the time constraint, we would have a pattern indicating that users often use a contacts application and then a camera application. This pattern suggests a relatively high correlation between these two applications. However, the correlation seems to be overestimated, and a more common interpretation is that users sometimes use contacts in the morning and use camera in the afternoon. The large time gap that separates these two application uses makes such a pattern less interesting. Moreover, if the time constraint is not considered, the only way to have a larger number of interesting patterns is to lower the minimum support, but this usually results in a large number of patterns that are not frequent in practice. For example, if we set the minimum support to a low value, we would possibly have a pattern containing an instant messaging application that is only popular in a relatively small group of users.

Nevertheless, if we first use the time constraint to filter the data and then use a general sequential pattern mining algorithm, such as PrefixSpan [4, 5], we will underestimate the supports given by some sessions to some patterns; we will miss some patterns, especially long ones, if we use the widely used implementation of the advanced algorithm specifically designed to mine the time-constrained sequential patterns [6], provided by SPMF (Sequential Pattern Mining Framework) that features the richest set of types of implementation of the pattern mining algorithms among all the available tools for pattern mining [7].

To overcome the shortcoming of the existing methods and tools, we develop an algorithm that can efficiently and effectively mine the time-constrained sequential patterns. We present its technical details in this paper. What is more, we discuss as well as visualize the mined patterns. Although the visualization is straightforward, we have not found its use in related papers. The user's usage patterns can potentially help enhance user interfaces of smartphones and/or help build better user experiences, so this paper could make practical contributions to the related industry. This paper could contribute to the related studies, such as those on discovering the user's habits or intentions of using some applications.

In the following sections, we will first discuss related work, next introduce the data and algorithm that we use, next discuss the results that we have obtained, and finally conclude this paper.

2. Related Work

From the algorithm perspective, those related to this paper are mainly in sequential pattern mining. From the application perspective, the related papers include those in mobile data mining and web usage log mining.

2.1. Sequential Pattern Mining. Given a database of sequences, each of which consists of transactions, the sequential pattern mining problem concerns how to efficiently generate patterns, each of which indicates that a sequence of purchased items can be found in sequences done by a relatively large number of customers [8]. For example, a sequential pattern indicates that if customers purchase an item in their current transactions then they would possibly purchase another item in their next transactions. Sequential pattern mining can be used to mine users' navigation patterns from web usage log, which is similar to but different from smartphone application usage log that we intend to mine, as we shall see later.

The sequential pattern mining algorithms based on pattern-growth, such as PrefixSpan [4, 5], focus their search for frequent patterns on a smaller portion of the given sequence database. PrefixSpan recursively projects the databases of the frequent prefixes that are generated based on the suffixes, and it adds items to a pattern one at a time. In the context of using smartphone applications, sequential pattern mining can help us answer the following question: after using a camera application, what application would possibly be used by a user next? In this paper, we are more interested in answering the following question: after using a camera application, what application would possibly be used by a user *in three minutes*? General sequential pattern mining cannot answer such a question but can time-constrained sequential pattern mining, which is what we are concerned with in this paper.

In [9], Hirate and Yamana discussed sequential pattern mining with time intervals, and they proposed an algorithm similar to PrefixSpan but having two types of projections: one is to scan the transaction database given as the input and calculate supports of items, while the other is to scan a projected database and calculate time intervals for pairs of items. PrefixSpan can be generalized to allow checking constraints when growing patterns [10]. As we shall see later, we do not focus on projection and we check the time constraint in support calculation, which allows us to mine more and longer patterns.

2.2. Mobile Data Mining. Eagle and Pentland compiled a data set that captures how participants use certain functions on their mobile phones [11]. The data set was collected from 100 mobile phones that are preinstalled with logging programs. The fields in the data set include call log, used function, and phone status. In [12], Akoush and Sameh used the data set to study the problem of mobile user movement prediction. In [13], Wang et al. worked on clustering instant messages in the data set. In [14], Farrahi and Gatica-Perez worked on the discovery of users' daily routines from the data set. The location data of the data set can be used to mine patterns of

daily changes of users' locations, such as [15, 16]. This paper is not about users' moving behaviors or daily routines, and it is not limited to instant messages.

Some papers are regarding malware detection: an example is [17], in which the analysis of process state transitions and patterns of users' operations was used to differentiate the operations performed by users from those performed by programs infected by malware; another example is [18], in which a crowdsourcing system was used to collect the traces of applications' behaviors, and then the traces were used to identify applications containing malware. Furthermore, some researchers analyzed application usage patterns for power or network management of smartphones. In [19], Falaki et al. investigated the relationship between user activities and power consumption, and they indicated that by knowing how a user interacts with his or her smartphone, user experiences can be improved and energy drain can be predicted more accurately. In [20], Kang et al. explored the relationship between usage patterns and power consumption by analyzing the usage data collected from several smartphone users for two months, and they showed that all users have their own usage patterns. In [21], Xu et al. used anonymized network measurements to study when, where, and how applications are used by smartphone users. Although this paper is not about malware detection and power management, they could potentially be the further applications of this paper.

Laurila et al. presented a study in which data was collected from smartphones for one and half years [22], and the data is used in Nokia Mobile Data Challenge. The data was used in papers on user behavior inference [23], demographics prediction [24], and location prediction [25]. In [26], Wu et al. analyzed the data in order to identify smartphone users' mobility patterns and support the opportunistic data collection through smartphones. However, these papers did not really present and discuss the patterns that indicate how users use applications on smartphones; from the papers that we refer to earlier, such patterns have a wide range of potential applications.

There were researchers working on mobile phone usage prediction for various purposes. In [27], Huang et al. proposed an algorithm to predict the next application that a user would possibly use for preloading applications. In [28], Yan et al. proposed to use the context of users' using applications to speed up the application launch process. In [29], Dai et al. proposed an algorithm using feature subspace to predict users' phone call behaviors. In [30], Lu et al. adopted an algorithm using the so-called "*physical location moving path*" and "*virtual application usage path*" to discover the "*mobile application sequential patterns*." However, what were built in these papers can be viewed as black boxes, because they make predictions but provide no explanation for how the predictions are made. What is built in this paper can be viewed as a glass-box, because it mines and presents patterns that indicate how users use applications on smartphones. Therefore, this paper would potentially be more valuable to the developers and researchers working on systems and applications.

2.3. Web Usage Log Mining. Sequential pattern mining has been used to support web usage log mining and analysis, such as [31, 32]. Smartphone application usage log mining is similar to but different from web usage log mining, because using a smartphone is different from browsing a web for several reasons. First, users have a high degree of freedom when browsing web pages, while users have a higher degree of freedom in installing, uninstalling, and using applications on their smartphones. Second, a webpage cannot but an application can be as an interface between its user and the environment, and examples of such applications are recorder and camera applications. Third, hyperlinks are designed to guide users to web pages, while users can arbitrarily use and switch between applications installed and running on smartphones without being restricted by the hyperlink-like predefined paths. Fourth, a user's browsing path almost always starts from the first page (such as the login page) and often stops at some page (such as the logout page), while a user can start using his or her smartphone from one of the installed launchers, most of which are highly customizable, or from one of the running applications, and the user can stop at any application. Fifth, a session physically exists in and is managed by a server and a client when a user is browsing a web, but there is no such a session in the system or application(s) when a user is using a smartphone.

According to these differences, smartphone application usage patterns would be more complex and longer than web usage patterns. They are more complex, because there are more possible combinations of items and itemsets; they are longer, because there could be repeated items and itemsets. Due to these differences, web usage log mining algorithms cannot be directly applied to the data that we use. Moreover, web usage log mining algorithms do not handle itemsets because, as described in [2], "*ordered sequences of events in the sequence database are composed of single items and not sets of items, with the assumption that a web user can physically access only one web page at any given point of time.*" In this paper, we view a series of application uses that are switched within a very short period of time (i.e., 1 second) as an itemset. Doing this allows us to mine more patterns that are more interesting and can help us gain more insights into users' behaviors, but doing this requires us to use an algorithm different from a general web log mining algorithm.

3. Materials and Methods

3.1. Data. The raw database used in this paper was built upon the log data that was collected by using the platform presented by Chen et al. in the paper [33]. The platform is developed to collect data for applications that use the log service in the Android system. The log data was generated by 80 users during the timeframe from September 2010 to March 2015. There are more than 2.5 million log records, each of which represents an application use, and there are more than 3 thousand applications. Following the convention used in papers on sequential pattern mining, we abstract an application use as an item and a set of (approximately) indivisible items as an itemset. First of all, we need to transform the database of log records into a database of

sessions, which are called sequences in sequential pattern mining and are similar to sessions in web usage log mining. Log records that belong to a user are sorted according to their timestamps. For a user, if the difference in time between two contiguous log records is not larger than 10 minutes, the two log records belong to the same session; in a session, if the difference in time between two contiguous log records is not larger than 1 second, these two log records belong to the same itemset. Accordingly, log records are clustered and transformed into sessions. As a result of the transformation, the database used in this paper consists of more than 250 thousand sessions.

We use the 10-minute threshold to separate sessions by referring to the paper [34], where Chen et al. presented a case study about whether a user will instantly share photos on Facebook after taking them with the camera application, and they defined instant sharing as the operation of a user uploading photos to Facebook within 10 minutes after those photos are taken. In this paper, if two contiguous log records that belong to a user are away from each other for more than 10 minutes, the application uses corresponding to them are considered less relevant and are placed in two sessions. For example, it seems unlikely that a user often uses a smartphone uninterruptedly from the morning to the afternoon. The use of the 1-second threshold to separate itemsets is based on the following assumption: a user may use an application as a shortcut to quickly launch or switch to another application, and a user may accidentally launch or switch to an application and then quickly switch back to the application that he or she wants to use at first.

The database used in this paper is larger and more complex than those used in some other papers. For example, Eagle and Pentland collected data from 100 mobile phone users for an academic year [11]; Lu et al. analyzed less than 3.5 thousand sequences, conceptually equivalent to sessions, collected from 30 students during the timeframe from June to October 2013 [30]. Furthermore, in [35, 36], the authors applied pattern mining to a database built upon the log data generated by 25 users during the timeframe from September 2010 and March 2011; the authors discussed the disadvantage of applying association rule mining to smartphone application usage data, and they further proposed to modify PrefixSpan such that it considers the time constraint in its projection function. However, the method proposed in [35, 36] would miss some patterns, especially long ones, in which we are interested; for example, our algorithm can return the pattern in which the official Facebook application is used for 20 times, while the method proposed in [35, 36] cannot return such a long pattern (and neither can SPMF). Given the popularity of Facebook, the pattern should be returned by an effective mining algorithm.

3.2. Definitions. In this subsection, we give notations, which are summarized in “Notations,” and definitions of variable and operation types used in our algorithm.

For an item a , $a.ts$ is its timestamp, or $a.ts$ is the timestamp of the application use corresponding to a .

Definition 1. An itemset is a list of items and corresponds to a list of application uses that happen almost simultaneously, and it is denoted by I , where $I = (ts: a_1, \dots, a_i, \dots, a_n)$, ts is the timestamp, and a is an item, $a_i < a_j$ in the lexicographic order for $1 \leq i < j \leq n$, and $|a_j.ts - a_i.ts| \leq 1$ second for $1 \leq i, j \leq n$.

For an itemset I , $I(i)$ is the i th item or a_i , $I.ts$ is its timestamp equal to $I(1).ts$, and $I.length$ is its length equal to the number of items in it. The timestamp will not be noted when not needed.

Definition 2. An itemset I_1 containing an itemset I_2 , where $1 \leq I_2.length \leq I_1.length$, is denoted by $I_2 \sqsubseteq I_1$ and is defined as follows: if $I_2.length = 1$, $I_2(1) = I_1(i)$ for some i , $1 \leq i \leq I_1.length$ or if $I_2.length > 1$, $I_2(i) = I_1(j)$ and $I_2(i+1) = I_1(k)$ for each i , $1 \leq i \leq I_2.length - 1$, and some j and k , $i \leq j < k \leq I_1.length$.

Definition 3. An itemset I_1 having a prefix itemset I_2 , where $1 \leq I_2.length \leq I_1.length$, is denoted by $I_2 \sqsubseteq_p I_1$ and is defined as follows: $I_2(i) = I_1(i)$ for each i , $1 \leq i \leq I_2.length$.

Definition 4. An itemset I_1 having a suffix itemset I_2 , where $1 \leq I_2.length \leq I_1.length$, is denoted by $I_2 \sqsubseteq_s I_1$ and is defined as follows: $I_2(i) = I_1(i + I_1.length - I_2.length)$ for each i , $1 \leq i \leq I_2.length$.

A dependent itemset is defined upon the concept of prefix and suffix. When an itemset is divided into a nonempty prefix and a nonempty suffix, the suffix is called a dependent itemset because it cannot be used alone but needs to be combined with an itemset used as a prefix. A session, or more precisely a usage session, is a period of time during which a user uses applications uninterruptedly on his or her smartphone. We use the term session instead of sequence to emphasize that it is regarding a series of application uses done by a user, which is similar to a series of visited or browsed web pages.

Definition 5. A session is denoted by S , where $S = \langle ts: I_1, \dots, I_p, \dots, I_m \rangle$, ts is the timestamp, and I is an itemset, $I_i.ts < I_j.ts$ for $1 \leq i < j \leq m$ and $I_{i+1}.ts - I_i.ts \leq 10$ minutes for $1 \leq i \leq m - 1$.

For a session S , $S(i)$ is the i th itemset or I_i , $S.ts$ is its timestamp equal to $S(1).ts$, and $S.length$ is its length equal to the number of itemsets in it. The timestamp will not be noted when not needed.

Definition 6. A session S_1 containing a session S_2 , where $1 \leq S_2.length \leq S_1.length$, is denoted by $S_2 \sqsubseteq S_1$ and is defined as follows: if $S_2.length = 1$ and $S_2(1) \sqsubseteq S_1(i)$ for some i , $1 \leq i \leq S_1.length$ or if $S_2.length > 1$, $S_2(i) \sqsubseteq S_1(j)$ and $S_2(i+1) \sqsubseteq S_1(k)$ for each i , $1 \leq i \leq S_2.length - 1$, and some j and k , $i \leq j < k \leq S_1.length$.

A session database containing m users and n sessions is denoted by $D = \{(S_i, uid_j) \mid 1 \leq i \leq n, 1 \leq j \leq m\}$, where the i th record corresponds to the i th session S_i made by the user having the j th identification number uid_j .

Definition 7. A session S_1 having a prefix session S_2 , where $1 \leq S_2.length \leq S_1.length$, is denoted by $S_2 \sqsubseteq_p S_1$ and is defined as follows: if $S_2.length = 1$ and $S_2(1) \sqsubseteq_p S_1(1)$ or

if $S_2.length > 1$, $S_2(i) = S_1(i)$ for each i , $1 \leq i \leq S_2.length - 1$, and $S_2(S_2.length) \sqsubseteq_p S_1(S_2.length)$.

Definition 8. A session S_1 having a suffix session S_2 , where $1 \leq S_2.length \leq S_1.length$, is denoted by $S_2 \sqsubseteq_s S_1$ and is defined as follows: if $S_2.length = 1$ and $S_2(1) \sqsubseteq_s S_1(1)$ or if $S_2.length > 1$, $S_2(1) \sqsubseteq_s S_1(i + S_1.length - S_2.length)$ and $S_2(i + 1) = S_1(i + 1 + S_1.length - S_2.length)$ for each i , $1 \leq i \leq S_2.length - 1$.

Projection is a core operation in a sequential pattern mining algorithm based on pattern-growth [5]. It is also a core operation in our algorithm. Given a session S as an input and S_p as a prefix, projection is to output the corresponding suffix S_s ; S_s is a projected session for S with respect to S_p . Given a session database D and a session S_p as a prefix, database projection is to output the corresponding suffix for every session in D and generate a session database denoted by $D_{|S_p}$. A session is a dependent session if its first itemset is a dependent itemset, and it can be used only when it is appended to a session that is not dependent. A session used as a prefix to project other sessions cannot be dependent, while a suffix generated by projection can possibly be a dependent session. A projected database can contain dependent sessions.

Definition 9. A session S_1 can be generated by appending a session S_2 to a session S_3 , where $S_2.length = m$ and $S_3.length = n$, and the itemsets in S_1 are determined as follows: if $S_2(1)$ is not a dependent itemset, then $S_1(i) = S_3(i)$ for $1 \leq i \leq n$ and $S_1(n + j) = S_2(j)$ for $1 \leq j \leq m$, or if $S_2(1)$ is a dependent itemset, then $S_1(i) = S_3(i)$ for $1 \leq i \leq n - 1$, $S_1(n - 1 + j) = S_2(j)$ for $2 \leq j \leq m$, $S_3(n)$ is a prefix of $S_1(n)$, $S_2(1)$ is a suffix of $S_1(n)$, and $S_1(n).length = S_3(n).length + S_2(1).length$.

In a rough sense, a pattern is a session without the timestamp. A pattern is denoted by P , where $P: I_1 \rightarrow \dots \rightarrow I_m$, I is an itemset, and the arrow symbol ($\rightarrow$) represents the transition from one itemset to another; here, a transition means that an application use is followed by another application use, and it indicates that a user switches from one application to another. We ignore the parentheses of a single-item itemset. For a pattern P , $P(i)$ is the i th itemset or I_i , and $P.length$ is its length equal to the number of itemsets in it. Definitions that can be applied to sessions can also be applied to patterns, when the timestamps are not used. For example, the operation used to append one pattern to another is basically the same as that used to append one session to another.

It is essential to define the relationship that a session supports a pattern with respect to the time constraint. The time constraint, denoted by δ , is the maximum difference in time between two itemsets in a session to which two contiguous itemsets in a pattern supported by the session can be mapped.

Definition 10. A session S supporting a pattern P , where $1 \leq P.length \leq S.length$, with respect to the time constraint δ is defined as follows: if $P.length = 1$ and $P(1) \sqsubseteq S(l)$ for some l , $1 \leq l \leq S.length$ or if $P.length > 1$, $P(i) \sqsubseteq S(j)$, $P(i + 1) \sqsubseteq S(k)$, and $S(k).ts - S(j).ts \leq \delta$ for each i , $1 \leq i \leq P.length - 1$ and some j and k , $i \leq j < k \leq S.length$.

Clearly, to determine if a session supports a single-item pattern is to determine if the item in the pattern appears in the session. The time constraint that we consider in this paper is not the same as the time gap constraint defined in the paper [10] (where j and k in Definition 10 are required to satisfy $k = j + 1$).

Definition 11. The support of a pattern P for the given database D and the time constraint δ is denoted by $\text{sup}(P, \delta)$ and is calculated by (1), where $\mathbb{I}$ is the indicator function:

$$\text{sup}(P, \delta) = \frac{\sum_{i=1}^n \mathbb{I}(S_i \in D \text{ supports } P \text{ w.r.t. } \delta)}{|D|}. \quad (1)$$

A pattern is frequent when the proportion of sessions in the database that support it is larger than or equal to θ , which is the threshold specifying the minimum support.

Definition 12. For the given session database D with respect to the time constraint δ and the minimum support θ , a pattern P is a frequent pattern if (2) holds:

$$\text{sup}(P, \delta) \geq \theta. \quad (2)$$

3.3. Algorithm. Our algorithm is based on pattern-growth [5], and its main function is shown in Algorithm 1. It is a recursive algorithm and uses two important functions: one is to find valid suffixes that will be used to extend the given prefix, and it is shown in Algorithm 1; the other is to project a database that will be given to the next invocation of the main function, and it is shown in Algorithm 1.

Initially, the given database specified by the first parameter of the main function shown in Algorithm 1 is the original database, and it is a projected database afterward. The second parameter is null in the first run of the main function, and it is the session used for projection in a subsequent run. The third parameter is the minimum support, and it holds the prespecified value first and an adjusted value then. The fourth parameter is the time constraint, which is used as a constant in mining.

To begin with, the main function finds items that appear frequently in the given database and can be used as suffixes satisfying the given time constraint. The items are transformed into single-item patterns. It iteratively considers each of the transformed patterns. It treats a considered pattern as a dependent pattern first and an independent pattern later, and it appends the pattern to the given prefix in order to extend the prefix and generate, or *grow*, a new pattern. The generated pattern will be returned as a result if its support is larger than or equal to the given minimum support. After that, the main function transforms the considered pattern into a session. It uses the session to project the given database. It makes a recursive invocation with the projected database, the transformed session used as the prefix, the adjusted minimum support, and the time constraint. In an invocation, it extends the prefix and grows a pattern by making the prefix one more item longer, and the item can be the last itemset or be part of the last itemset. The main function adjusts the minimum support because it uses a projected database as an

Part 1: The main function

Function: Mining

Input: A session database D , a prefix session S_p , the minimum support θ , the time constraint δ

Output: A set of patterns R , which initially is empty

Steps:

- (1) Suffixes = Preparation (D, S_p, θ, δ)
- (2) For each pattern P in Suffixes, do:
  - (3) Treat P as a dependent pattern and append it to S_p to generate $P^{(d)}$
  - (4) If $\text{sup}(P^{(d)}, \delta) \geq \theta$, then do
    - (5) $R = R \cup \{P^{(d)}\}$
  - (6) Treat P as an independent pattern and append it to S_p to generate $P^{(i)}$
  - (7) If $\text{sup}(P^{(i)}, \delta) \geq \theta$, then do
    - (8) $R = R \cup \{P^{(i)}\}$
  - (9) S = the session transformed from P
  - (10) $D_{|S} = \text{Projection}(D, S_p, S)$
  - (11) $R = R \cup \text{Mining}(D_{|S}, S, |D| \times \theta / |D_{|S}|, \delta)$

Part 2: The function used to prepare suffixes for growing patterns

Function: Preparation

Input: A session database D , a prefix session S_p , the minimum support θ , the time constraint δ

Output: A set of patterns R , which initially is empty

Steps:

- (1) $M^{(i)}$ = a map used to store independent single-item itemsets
- (2) $M^{(d)}$ = a map used to store dependent single-item itemsets
- (3) For each session S in D , do
  - (4) For each index i from the first independent to last itemset in S , do
    - (5) If $S(i).ts - S(1).ts > \delta$, then do
      - (6) Escape from the inner loop
    - (7) For each item a in $S(i)$
      - (8) Increase the count in $M^{(i)}$ for appearance of a
    - (9) If S is dependent, then do
      - (10) For each item a in $S(1)$
        - (11) Increase the count in $M^{(d)}$ for appearance of a
    - (12) For each index i from the first independent to last itemset in S , do
      - (13) If $S_p(S_p.length)$ in $S(i)$, then do
        - (14) If $S(i).ts - S(i-1).ts \leq \delta$, then do
          - (15) For each item a in $S(i)$ but not in $S_p(S_p.length)$, do
            - (16) Increase the count in $M^{(d)}$ for appearance of a
          - (17) Escape from the inner loop
    - (18) For each itemset I in $M^{(i)}$, do
      - (19) If the count $\geq |D| \times \theta$
        - (20) $P^{(i)} = I$ transformed into a pattern
        - (21) $R = R \cup \{P^{(i)}\}$
    - (22) For each itemset I in $M^{(d)}$ do
      - (23) If the count $\geq |D| \times \theta$
        - (24) $P^{(d)} = I$ transformed into a pattern
        - (25) $R = R \cup \{P^{(d)}\}$

Part 3: The function used to project a database

Function: Projection

Input: A session database D , a prefix session S_p , a base session B

Output: A set of sessions R , which is initially empty, used to build a database

Steps:

- (1) For each session S in D , do
  - (2) $S^{(n)} = S, B^{(n)} = B$
  - (3) $S^{(t)}$ = the session transformed from $S_p(S_p.length)$
  - (4) If S is dependent, then do
    - (5) $S^{(n)}$ = the session generated by appending S to $S^{(t)}$
  - (6) If B is dependent, then do
    - (7) $B^{(n)}$ = the session generated by appending B to $S^{(t)}$
  - (8) If ($S^{(n)}$ is appended but $B^{(n)}$ is not)
    - or ($S^{(n)}$ and $B^{(n)}$ are appended and $B^{(n)} = S^{(t)}$), then do

- (9) If $S^{(n)}.length > 1$, then do
 (10) $S^{(n)}$ = the sub-session generated by removing $S^{(n)}(0)$
 (11) $S^{(p)}$ = the session generated by using $B^{(n)}$ to project $S^{(n)}$
 (12) $R = R \cup \{S^{(p)}\}$

ALGORITHM 1: The algorithm proposed to mine time-constrained sequential patterns.

input. Suppose that the original database consists of 1,000 sessions ($|D| = 1,000$) and the minimum support is set to 0.05 ($\theta = 0.05$). So, a frequent pattern needs to be supported by at least $1,000 \times 0.05$ or 50 sessions. If the given database in a run of the main function (or the database input to the main function) is a projected database consisting of 100 sessions ($|D_S| = 100$), then the minimum support needs to be $1,000 \times 0.05/100$ or 0.5 in order to use the same rule (or code) to check whether a pattern is frequent.

Algorithm 1 also shows the function used to prepare suffixes for growing patterns. The first parameter of the preparation function shown in Algorithm 1 is the original database or a projected database; the second is a prefix session used in calculating support count for a dependent itemset (which cannot be used alone); the third is the specified or an adjusted minimum support; the fourth is the time constraint directly passed from the main function. A suffix prepared for growing patterns is a pattern containing an item transformed into an itemset that can be independent or dependent. Every item in an independent itemset contributes to the support count of the independent single-item itemset corresponding to it. For example, in an independent itemset (a_1, a_2) , a_1 contributes to the support count of the independent itemset (a_1) and so does a_2 to that of (a_2) . Similarly, every item in a dependent itemset contributes to the support count of the dependent single-item itemset to which it corresponds. In a dependent itemset (a_1, a_2) , for example, a_1 and a_2 contribute to the support count of the dependent itemsets (a_1) and (a_2) , respectively. If the last itemset of the given prefix appears in an itemset in an independent session, the items in the itemset that follow all the items in the last itemset of the prefix contribute to support counts. Furthermore, when checking if the items in the itemsets in a session are valid suffixes with respect to the given time constraint, for a session having only one itemset, the function compares the timestamp of the itemset to that of the preceding itemset in the original session; for a session having more than one itemset, the function compares the timestamp of each of the itemsets except the first one to the timestamp of its first itemset. We possibly need to access every itemset in a session to check if the itemsets in the session satisfy the time constraint. Therefore, we need to access the original database.

The function used to project a database is also shown in Algorithm 1, and it projects sessions in the given database one after another. The first parameter of the projection function shown in Algorithm 1 is the original database that will be projected or a projected database that will be further projected; the second is a prefix session used in projecting dependent sessions in the given database; the third is a base session, which could be independent or dependent, found by the

function shown in Algorithm 1 to be frequent (locally) in the given database. Let S be a session in the database that is going to be projected, while S can be from the original database or a projected database. Let B be a session that is going to be used as the prefix to project S . If S is dependent, the function retrieves its original form by appending it to the prefix that was used to generate it through projection. Afterward, a new session is created. If B is dependent, the function appends it to the prefix that was used to generate S . Afterward, a new base is created. If S is dependent but B is not and if there is more than one itemset in the new session, the function ignores the first itemset of the new session and then uses the new base to project the sub-session of the new session. If S and B are dependent and the new base is equal to the last itemset of the prefix that was used to generate S and if there is more than one itemset in the new session, to avoid doing the same projection again, the function considers the sub-session starting from the second itemset of the new session and then projects the sub-session with the new base.

The following is an example that demonstrates how projection works. We project $S_1 = \langle (a_1, a_2, a_3) (a_1, a_3, a_4) (a_1, a_4) \rangle$ with $B_1 = \langle (a_1) \rangle$ and have a dependent session $S_2 = \langle (a_2, a_3) (a_1, a_3, a_4) (a_1, a_4) \rangle$. When we use B_1 to project S_2 , we have a dependent session $S_3 = \langle (a_3, a_4) (a_1, a_4) \rangle$. When we use $B_2 = \langle (a_3) \rangle$ to project S_2 , we have a dependent session $S_4 = \langle (a_4) (a_1, a_4) \rangle$.

3.4. Example. In this subsection, we use the session database given in Table 1 as an example. We apply our algorithm to the database with the minimum support set to $1/3$, which is the same as the minimum support count set to 2, and with the time constraint set to 10.

In S_4 , the difference in time between the third itemset and the second itemset is 15, which is larger than the time constraint, 10. If we preprocess it by removing the third itemset (in order to have a new session that satisfies the time constraint) and input the preprocessed session with others to a general sequential pattern mining algorithm, we will not have a pattern starting with a_4 ; for instance, we will not have the pattern $\langle (a_4) \rangle$. The reason is that, after the preprocessing, a_4 only appears in S_1 and hence its support count is 1, smaller than the minimum support count, 2. In [37], the authors discuss preprocessing time constraints (i.e., using the time constraint to remove or filter itemsets in sessions) to efficiently mine generalized sequential patterns. As we can see from the example, preprocessing will result in missing patterns and therefore cannot be used.

We illustrate the running of our algorithm in Figure 1. In the figure, a block or a node is a session database (which could be the original or a projected database), a line with an

TABLE 1: Package names of some popular applications.

Package	Application	Note
<i>com.android.htcdialer</i>	<i>HTC Dialer</i>	An application built in HTC smartphones
<i>com.android.phone</i>	<i>Phone</i>	A native application built in Android
<i>com.android.htccontacts</i>	<i>HTC Contacts</i>	An application built in HTC smartphones
<i>com.android.mms</i>	<i>MMS</i>	A native application built in Android for multimedia messaging service
<i>jp.naver.line.android</i>	<i>Line</i>	A popular instant messaging (IM) application
<i>com.htc.launcher</i>	<i>HTC Sense (Launcher)</i>	A multifunction launcher built in HTC smartphones
<i>com.android.camera</i>	<i>Camera</i>	A native application built in Android
<i>com.htc.album</i>	<i>HTC Album</i>	An application built in HTC smartphones
<i>com.facebook.katana</i>	<i>Facebook</i>	The official application used to access one of the most popular social networking services (SNS)
<i>com.android.browser</i>	<i>Web Browser</i>	A native application built in Android
<i>com.whatsapp</i>	<i>WhatsApp</i>	A popular instant messaging (IM) application
<i>com.google.android.apps.maps</i>	<i>Maps</i>	A native application built in Android
<i>com.google.android.talk</i>	<i>Google Talk (Hangouts)</i>	A native instant messaging (IM) application built in Android
<i>com.htc.android.mail</i>	<i>HTC Mail</i>	An application built in HTC smartphones
<i>com.android.vending</i>	<i>Google Play</i>	A native application built in Android to access the application store
<i>com.htc.android.worldclock</i>	<i>HTC Clock</i>	An application built in HTC smartphones
<i>com.android.settings</i>	<i>Settings</i>	A native application built in Android
<i>com.htc.socialnetwork.plurk</i>	<i>Plurk</i>	A popular microblogging application
<i>tw.anddev.aplurk</i>	<i>Plurk</i>	A popular microblogging application
<i>com.instagram.android</i>	<i>Instagram</i>	An application that provides photo- and video-sharing services and social networking services (SNS)
<i>com.android.launcher</i>	<i>Native Launcher</i>	A native application built in Android
<i>com.google.android.youtube</i>	<i>Youtube</i>	The official application built in Android and used to access one of the most popular video-sharing services

arrow is related to a projection, the number in square brackets associated with the base session used for a projection is the support count, and the star symbol (*) means a dependent itemset. On one hand, a node is an input to the function shown in Algorithm 1, and a set of lines at the same level is the output of the function while each line is also an input (because our algorithm runs recursively). For the first run of the function shown in Algorithm 1, the first parameter (D) is the original database represented by the root node or Node 1 in Figure 1; the second parameter (S_p) is null; the output is the set of patterns associated with the lines from Node 1 to Nodes 2, 3, 4, and 5. For the second run of the function shown in Algorithm 1, D is the projected database containing 5 sessions and represented by the Node 2 in Figure 1; S_p is $\langle\langle a_1 \rangle\rangle$ associated with the line from Node 1 to Node 2. On the other hand, a node is an input of the function shown in Algorithm 1 and also the output, and a line is an input. For the first run of the function, the first parameter (D) is the original database; the second parameter (S_p) is null; the third parameter (B) is $\langle\langle a_1 \rangle\rangle$ associated with the line from Node 1 to Node 2 in Figure 1; the output is the projected database represented by Node 2; for the second run, D is the output of the last run; S_p is $\langle\langle a_1 \rangle\rangle$ associated with the line from Node 1 to Node 2; B is $\langle\langle a_3 \rangle\rangle$ associated with the line from Node 2

to Node 6; the output is the projected database containing 2 sessions and represented by Node 6.

When the given database is the original database and the given prefix is null, the itemset (a_1) appears in 6 sessions and hence its support count is 6; each of the itemsets (a_2) and (a_3) appears in 4 sessions and hence has the support count of 4; the itemset (a_4) appears in 2 sessions and hence its support count is 2. Please recall that we consider timestamps when calculating supports but not projecting databases. After the original database is projected with the prefix $\langle\langle a_1 \rangle\rangle$, there are 5 sessions: $\langle(100: a_2, a_3) (105: a_1, a_3, a_4) (110: a_1, a_4)\rangle$, $\langle(100: a_2) (115: a_3)\rangle$, $\langle(100: a_2)\rangle$, $\langle(105: a_2, a_3) (120: a_2, a_3, a_4)\rangle$, and $\langle(100: a_3)\rangle$, where all but the fourth are dependent. The itemset (a_3) appears in 2 sessions and has the support count of 2. The independent itemset (a_2) only appears in the fourth session, while the dependent itemset (a_2) appears in first 3 sessions and hence has the support count of 3. The dependent itemset (a_3) has the support count of 2, because it appears in the first and fifth session. The item transformed into the prefix, a_1 , does not appear in any itemset in the fourth session, which is independent. Therefore, the items in the fourth session make no contributions to support counts of the corresponding dependent single-item itemsets. As an example, (a_3) in the fourth session does not contribute to

TABLE 3: Patterns mined from the example session database.

$P_1: a_1$	Support count = 6
$P_2: a_1 \rightarrow a_3$	Support count = 2
$P_3: (a_1, a_2)$	Support count = 3
$P_4: (a_1, a_3)$	Support count = 2
$P_5: a_2$	Support count = 4
$P_6: (a_2, a_3)$	Support count = 2
$P_7: a_3$	Support count = 4
$P_8: (a_3, a_4)$	Support count = 2
$P_9: a_4$	Support count = 2

In Section 4.1, we report and compare runtime performance. In Section 4.2, we present and discuss some max patterns, each of which is a pattern that may contain many others but is not contained by any other. Sections 4.1 and 4.2 are concerned with efficiency and effectiveness of our algorithm, respectively. Yan et al. state that “*the major challenge of frequent-pattern mining is not at the efficiency but at the interpretability*” [38], which declares the importance of Sections 4.2 and 4.3. In Section 4.3, we present sequential pattern graphs that can be used in the visual analysis and further the interpretability of the mining results.

4.1. Runtime Performance. We run experiments on a general PC having a quad-core CPU running at 2.3 GHz and 8 GB RAM. We implement our algorithm in Java, and experiments are run in Java runtime environment where the maximum heap size is set to 4 GB.

To see how the minimum support affects runtime performance, we set the time constraint to 180 seconds, and we

FIGURE 2: Elapsed time in seconds for applying our algorithm with varying minimum supports to the database when the time constraint is set to 180 seconds.

FIGURE 3: Number of patterns mined by our algorithm and SPMF with varying minimum supports when the time constraint is set to 180 seconds.

consider the minimum supports from 0.01 to 0.1 with a step of 0.01. Figure 2 presents the elapsed time in seconds for applying our algorithm and SPMF with varying minimum supports to the database. Figure 3 presents the number of patterns returned by our algorithm and SPMF with varying minimum supports. Please notice that the vertical axis in Figure 3 is in log scale.

As the minimum support increases for each of the two algorithms, the elapsed time decreases rapidly and so does the number of mined patterns. The distribution of applications is very skew, because some applications are used much more frequently than others. If we order applications by how frequently they are used, the top 2% applications constitute 90% of log records. When the minimum support is as low as 0.01, SPMF is faster (even though there is no significant difference in the elapsed time between our algorithm and SPMF). When the minimum support is larger, our algorithm is faster (and the difference is significant). The relative speed-up of our algorithm over SPMF increases as the minimum support increases. Moreover, SPMF only returns a smaller number of patterns, compared to the number of patterns returned by our algorithm; the number is much smaller when the minimum support is as low as 0.01. As the minimum support increases,

FIGURE 4: Elapsed time in seconds for applying our algorithm with varying time constraints to the database when the minimum support is set to 0.005.

FIGURE 5: Number of patterns mined by our algorithm (based on pattern-growth) and SPMF with varying time constraints when the minimum support is set to 0.005.

the ratio of the number of patterns returned by SPMF to that returned by our algorithm increases.

To see how the time constraint affects runtime performance, we set the minimum support to 0.005 (which will give us large numbers of patterns), and we consider various time constraints starting from 30 to 180 seconds with a step of 30 seconds. Figure 4 presents the elapsed time in seconds for applying our algorithm and SPMF with varying time constraints to the database. When we increase the time constraint from 30 to 180 seconds, we obtain a significant increase in the elapsed time for our algorithm and SPMF. Figure 5 presents the number of patterns returned by our algorithm and SPMF with varying time constraints. Please notice that the vertical axis in Figure 5 is in log scale.

Our algorithm and SPMF need more time to mine patterns when the time constraint is larger. When the time constraint is larger, there is a larger difference in the elapsed time between our algorithm and SPMF. When the minimum support is set to a small value and when the time constraint is set to a large value, SPMF is faster but only returns a relatively small number of patterns. For example, when the minimum support is set to 0.005 and the time constraint is set to 180

seconds, our algorithm uses 355.5 seconds to finish mining 194,219 patterns and SPMF uses 76.5 seconds to finish mining 977 patterns. If we take the number of patterns into account, our algorithm is more efficient than SPMF; our algorithm and SPMF work in the rates of 546.3 and 12.8 patterns mined in a second, respectively.

4.2. Max Patterns. If a pattern is not contained by or is not part of any other patterns, it is a max pattern. Max patterns can be viewed as summarization of the mined patterns. When the minimum support is set to 0.01 and the time constraint is set to 180 seconds, there are 5,400 mined patterns and they can be summarized by 1,478 max patterns. Under the same setting, SPMF returns 231 patterns, among which 54 are max patterns. Among these 54 patterns, 18 are also max patterns returned by our algorithm, while others are not because they are contained by longer patterns returned by our algorithm (but not returned by SPMF). We report 15 (about 1%) max patterns in Table 4. Among the 15 patterns, 4 are also returned by SPMF: P_1 through P_4 .

Today, most smartphones have built in GPS (Global Positioning System), and some users use smartphones as digital maps or navigation devices. When users use the map application, they may keep using it for several minutes. For example, a user may look at the map displayed on the screen of his or her smartphone when walking and finding the way or the place in which he or she is interested. According to P_1 in Table 4, the support of *Maps* is 0.02. It is a frequently used application, but we do not find longer patterns containing it and having a sufficiently large support. This may contradict the common perception that users switch between applications (such as contact, mail, and web browser) for information about addresses or directions when using the map application. This may imply that users do not use the map application as a compliment to location based services.

P_2 is regarding the use of *Google Talk*, which is an instant messaging application that can be used for voice communication. It is a max pattern, which implies that when users use this application they are not eager to switch to other frequently used applications in 180 seconds. P_2 indicates an exclusive use of *Google Talk*, while P_{15} indicates repeated uses of *Line*, another instant messaging application. The difference is possibly due to the difference in designs of their user interfaces and the difference in features that each emphasizes (or markets).

Emailing is an important function of smartphones. Smartphones are equipped with smaller screens, compared to desktops, laptops, and tablets; reading and writing long mails on a small screen are not a pleasant experience for most people. We can see from P_3 that the support of *HTC Mail* is 0.01. *HTC Mail* itself is used frequently, but we do not find longer patterns containing it and having a support that is larger than or equal to the minimum support. This implies that the use of *HTC Mail* is usually not triggered by others and is usually not going to trigger other application uses. Working on emails on smartphones is exclusive. Users may spend less than 30 seconds quickly checking emails and making short replies to emails or users may spend more than 180 seconds carefully reading and/or writing emails.

Most of the users who provide us data use smartphones made by HTC. The company develops a launcher application, named *HTC Sense*, and preinstalls it as the default launcher on the smartphones under the HTC brand. *HTC Sense* provides a user interface modified from the original user interface, additional features, additional functions, an integration of several applications developed by the company, and an interface that can aggregate information from social networking services (SNS) or some other services.

P_4 is regarding the typical function of mobile phones. It indicates that a user may open *HTC Sense* and then in 1 second switch to *MMS*, which is used to send and receive short messages. Such a pattern appears in 1% of sessions. It shows an approach to the quick launch of *MMS*. P_4 is a max pattern, which implies that quickly launching *MMS* through *HTC Sense* is exclusive and is not used right after or before another frequently used application.

P_5 indicates that users use *Google Play* to access the online application store for Android, and then they use it again in 180 seconds. There are possibly several infrequently used applications between the two uses of *Google Play*. It is a max pattern, which means that *Google Play* is neither following nor followed by the use of another frequently used application. This implies that when users are browsing or shopping at the online application store, they are not eager to leave it for another frequently used application.

P_6 and P_7 are regarding typical functions of mobile phones. P_6 can be interpreted as follows: in 1% of sessions, users use *MMS* to browse or send short messages; then in 180 seconds they use *Phone* to make or receive phone calls; afterward, in 180 seconds they open *HTC Sense* (by pressing Home button, for example) after finishing phone calls. Similarly, an interpretation of P_7 is as follows: in 1% of sessions, users use *Phone* to make or receive phone calls, then in 180 seconds they open *MMS* to send or check short messages, and then in 180 seconds they switch to *HTC Sense* after finishing phone calls.

P_8 indicates that users open *HTC Dialer* through *HTC Sense*, and then in 180 seconds they use *Phone*. The difference in time between the use of *HTC Dialer* and the use of *Phone* is probably much smaller than 180 seconds. After using *Phone*, users may use other infrequently used applications, and then in 180 seconds they use *HTC Sense* again. They use *Phone* to make or receive phone calls, which is not longer than 180 seconds.

For P_9 , a possible scenario is that a user opens *HTC Contacts* and searches the contact list for a person's contact number(s); next, the user uses *Phone* to make a phone call; next, the user switches to some applications that are not among the most frequently used applications; next, the user uses *Phone* to make a phone call again; finally, the user presses Home button to open *HTC Sense*. Another possible scenario for P_9 is that a user opens *HTC Contacts* to have a person's contact number(s); afterward, he or she uses some infrequently used applications and then uses *Phone* to receive a phone call; afterward, he or she switches to some infrequently used applications and then switches back to *Phone* because of an incoming phone call; finally, after finishing the phone call, he or she presses Home button and switches to *HTC Sense*.

TABLE 4: Max patterns mined when the minimum support is set to 0.01 and the time constraint is set to 180 seconds.

	Max Pattern	Support
P_1	<i>Maps</i>	0.02
P_2	<i>Google Talk</i>	0.02
P_3	<i>HTC Mail</i>	0.01
P_4	<i>(HTC Sense, MMS)</i>	0.01
P_5	<i>Google Play → Google Play</i>	0.01
P_6	<i>MMS → Phone → HTC Sense</i>	0.01
P_7	<i>Phone → MMS → HTC Sense</i>	0.01
P_8	<i>(HTC Sense, HTC Dialer) → Phone → HTC Sense</i>	0.01
P_9	<i>HTC Contacts → Phone → Phone → HTC Sense</i>	0.01
P_{10}	<i>HTC Sense → HTC Clock → HTC Sense</i>	0.02
P_{11}	<i>HTC Sense → (HTC Sense, Settings) → Settings → HTC Sense</i>	0.01
P_{12}	<i>Settings → Web Browser → Web Browser → HTC Sense → Settings → HTC Sense</i>	0.02
P_{13}	<i>(HTC Sense, Web Browser) → Web Browser → Web Browser → Web Browser → HTC Sense → Web Browser → Web Browser</i>	0.01
P_{14}	<i>(HTC Sense, Facebook) → Facebook → Facebook → HTC Sense</i>	0.01
P_{15}	<i>(HTC Sense, Line) → Line → Line → Line → HTC Sense</i>	0.01

For P_{10} , a possible scenario is that, a user presses Home button and opens *HTC Sense*, checks the time and/or sets (or resets) the alarm, and then presses Home button and opens *HTC Sense* again. Because the scenario is common, the support of P_{10} is relatively high.

P_{11} shows a quick switch between *HTC Sense* and *Settings*. For P_{11} , a possible scenario is that a user uses *HTC Sense* to open the (system-level) setting panel and then switches back to *HTC Sense* again. Between the first two uses of *HTC Sense*, there are possibly the uses of some infrequently used applications; between the two uses, the difference in time is larger than 1 second and smaller than 180 seconds. Similarly, between the two uses of *Settings*, some infrequently used applications are possibly used; the difference in time between the two uses of *Settings* is larger than 1 second and smaller than 180 seconds.

P_{12} can be interpreted as follows: in 2% of sessions, users use *Settings* to change settings of the system and then in 180 seconds they start using *Web Browser*; afterward, users use *Web Browser* again in 180 seconds, while they may use other infrequently used applications between the two uses of *Web Browser*; afterward, users open *HTC Sense* in 180 seconds after their last use of *Web Browser*; finally, users use *Settings* again in 180 seconds after their last use of *HTC Sense*, and then they use *HTC Sense* again in 180 seconds. Furthermore, a possible scenario for P_{12} is that a user uses *Settings* to turn on Wi-Fi (or to enable data transmission), opens *Web Browser* to browse WWW, and then uses *Settings* to turn off Wi-Fi (or to disable data transmission) after he or she finishes browsing.

Below is a possible scenario for P_{13} : a user presses Home button to launch *HTC Sense* and then switches to *Web Browser* in 1 second. Afterward, the user repeatedly uses *Web Browser*. Between two uses of *Web Browser*, the user may use other infrequently used applications, while he or she may press Home button to switch to *HTC Sense* and later switch

back to *Web Browser* again. This is a common pattern, while our algorithm discovers it but SPMF does not.

P_{14} and P_{15} are common patterns. P_{14} and P_{15} show how much users are addicted to *Facebook* and *Line*, respectively. A possible scenario for P_{14} is as follows: a user presses Home button to launch *HTC Sense* and then switches to *Facebook* in 1 second. Following that, the user repeatedly uses *Facebook*. The user may use other infrequently used applications between two uses of *Facebook*. The user switches back to *HTC Sense* by pressing Home button after finishing the use of *Facebook*. A possible scenario for P_{15} is as follows: a user presses Home button to launch *HTC Sense* and then opens *Line* in 1 second. Following that, he or she repeatedly uses *Line*. Between two uses of *Line*, he or she may use other infrequently used applications. After finishing the use of *Line*, he or she opens *HTC Sense* by pressing Home button. Our algorithm discovers these two common patterns, which show how much users are addicted to online social activities, but SPMF does not.

For the max patterns reported in Table 4, when the minimum support is set to 0.01 and the time constraint is set to 30, 60, 90, 120, 150, or 180 seconds, the corresponding supports are reported in Table 5. The first column in Table 5 is for patterns reported in Table 4. Please note that, P_1 , P_2 , P_3 , and P_4 are not included in Table 5 because they are single-itemset patterns and thus the time constraint has no effect. The values in the last column in Table 5 are directly from the values in the last column in Table 4. If the support of a pattern under a certain combination of settings is lower than the minimum support, there will be a null value (an empty cell) in the corresponding position in Table 5.

If the time constraint is small, a pattern implies that a user switches from an application to another in a short interval. If the time constraint is large, a pattern implies that a user makes a switch in a short or long interval.

TABLE 5: Max patterns and their supports obtained by setting the minimum support to 0.01 and setting the time constraint to 30, 60, 90, 120, 150, or 180 seconds.

	Time constraint in seconds					
	30	60	90	120	150	180
P_5	0.01	0.01	0.01	0.01	0.01	0.01
P_6			0.01	0.01	0.01	0.01
P_7				0.01	0.01	0.01
P_8		0.01	0.01	0.01	0.01	0.01
P_9				0.01	0.01	0.01
P_{10}	0.02	0.02	0.02	0.02	0.02	0.02
P_{11}				0.01	0.01	0.01
P_{12}					0.01	0.02
P_{13}						0.01
P_{14}						0.01
P_{15}				0.01	0.01	0.01

Furthermore, it is beneficial to observe how the support of a pattern changes as the time constraint changes. The support of a pattern increases monotonically as the time constraint increases. When the time constraint increases, if the support of a pattern increases insignificantly or does not increase at all, the pattern implies that a user often spends less time on some involved application(s) before switching; if the support increases significantly, the pattern implies that a user usually needs more time to operate or finish some involved application(s) before switching.

According to Table 5, some patterns appear even when the time constraints are small, while some patterns appear only when the time constraints are large. This shows that patterns of application uses are sensitive to the time constraints. P_5 or P_{10} appears when the time constraint is 30 seconds, and there is almost no increase in its support as the time constraint is increased to 180 seconds. This means that P_5 and P_{10} indicate short-interval application uses, which could result from or in frequently switching between applications. P_{13} or P_{14} only appears when the time constraint is 180 seconds, meaning that P_{13} and P_{14} indicate long-interval application uses, which could be due to the fact that some applications simply need more time to operate or finish. P_6 and P_7 appear when the time constraints are 90 and 120, respectively; this implies that, it is relatively difficult (or uncommon) for a user to finish the use of *Phone* or *MMS* when both are used in the same session. P_8 appears when the time constraint is 60 seconds but P_9 does not appear until the time constraint is increased to 120 seconds. Here are two possibilities. First, *HTC Dialer* needs less time to operate than does *HTC Contacts* (since the latter comes with a more complex user interface). Second, *Phone* needs more time to finish (since it needs time to make or receive a phone call). P_{11} and P_{15} indicate repeated uses of *Settings* and *Line*, respectively, and either does not appear when the time constraint is smaller than 120 seconds. For *Settings*, its user interface might be too complex for users to have its use done easily or shortly; for example, in order to search and set (or reset) a Wi-Fi connection, a user may need to make several clicks on the screen, wait for the responses,

and use a small keyboard to enter a complex password. For *Line*, its use might be of much fun so that users spend more time on it before switching to other applications and then keep switching back to it. P_{12} indicates repeated uses of *Web Browser*, and it does not appear until the time constraint is increased to 150 seconds; when the time constraint is increased from 150 to 180 seconds, its support is increased from 0.01 to 0.02. This suggests that browsing a web page might take time no less than 2.5 minutes and more often take more time.

4.3. Sequential Pattern Graphs. In this subsection, we focus more on itemsets or items than on patterns. We use sequential pattern graphs to visualize the relationships between itemsets or items in a group of patterns. We propose two types of graphs. One is defined upon itemset and the other is defined upon item; the former can show us the quickly switching relationships between items (as an itemset contains items that appear almost simultaneously), while the latter can show us the transitions between items. We use JUNG (Java Universal Network/Graph Framework) [39], a widely used tool for network/graph visualization, to create the graphs. To avoid redundant computation, we create the graphs by using max patterns.

4.3.1. Graph Defined upon Itemset. A sequential pattern graph is a directed graph, and it can be described as follows. A node is an itemset; an edge is a transition indicating that an itemset is used and then another is used; and the direction of an edge is the direction of the transition.

Figure 6 presents a sequential pattern graph defined upon itemset for the patterns mined by our algorithm when the time constraint is set to 180 seconds and the minimum support is set to 0.005. This combination of settings will give us the largest number of patterns. In the figure, there are 75 nodes and 228 edges; 20% of nodes are nonsingle-item itemsets, each of which means that a user opens an application and in 1 second switches to another. In the figure, green nodes represent applications built in Android system, while red nodes represent third-party applications. Nodes having no edges are isolated nodes, and they represent applications that are not frequently used before or after other frequently used applications with the time constraint (180 seconds); in the figure, slightly more than 30% of nodes are isolated nodes, most of which represent third-party applications. As an example, an isolated node represents *HTC Sense* and *Phone*, and a possible scenario is that a user opens the launcher and then quickly switches to *Phone* to receive a phone call. As another example, an isolated node represents *HTC Sense* and *WhatsApp*, and a possible scenario is that a user launches *WhatsApp* through the launcher to make and receive messages.

The node representing *HTC Sense* has the highest in-degree and out-degree. The node having the second highest in-degree is the one representing *Facebook*. Two nodes have the third highest in-degree, and they represent *Web Browser* and *Line*; these two nodes have the same out-degree. The nodes *Phone* and *Facebook* have the second and third highest out-degrees, respectively. Users often open launchers or use

FIGURE 7: Sequential pattern graph defined upon item for patterns mined by setting time constraint to 180 seconds and the minimum support to 0.005.

the transition. We create edges for items in an itemset. For example, we create two edges for the itemset (a_1, a_2) : one is from a_1 to a_2 , and the other is from a_2 to a_1 .

Figure 7 presents a sequential pattern graph defined upon item for the patterns mined by our algorithm when the time constraint and the minimum support are set to 0.005 and 180 seconds, respectively. There are 58 nodes and 183 edges in the figure, where again green nodes represent applications built in Android system and red nodes represent third-party applications. Isolated nodes are those having no edges, and they represent applications that are not frequently used before or after other frequently used applications with the time

constraint (180 seconds); in the figure, slightly more than 30% of nodes are isolated nodes, and most of them represent third-party applications. For example, *Youtube* is represented by an isolated node, meaning that its use is exclusive. This is because watching a video is exclusive and also because most videos are longer than 180 seconds. Compared to Figure 6, Figure 7 is simpler and easier to be analyzed.

The node representing *HTC Sense* has the highest in-degree and out-degree. The node having the second highest in-degree is *Facebook*, and the node having the third highest in-degree is *Line*. Two nodes have the second highest out-degree, namely, *Facebook* and *Phone*. Two nodes have the

third highest out-degree, namely, *Line* and *Web Browser*. Users often go back to launchers or use some other applications after having phone calls, and therefore *Phone* has a high out-degree.

If the in-degree of a node is larger than its out-degree, a possible scenario is that users stop using other frequently used applications after opening or using the application represented by the node. Among these nodes, the one having the largest difference between in-degree and out-degree is *HTC Sense*, and the one having the second largest difference is *Native Launcher*. If the out-degree of a node is larger than its in-degree, a possible scenario is that users start using other frequently used applications after opening or using the application represented by the node. Among these nodes, two have the largest difference between out-degree and in-degree, namely, *Phone* and *MMS*. *HTC Dialer* has the second largest difference. Each of the nodes representing applications for *Plurk*, *WhatsApp*, and *Instagram* has the out-degree larger than the in-degree. This again tells us that social activities are an essential function of smartphones.

5. Conclusion

In this paper, we present our work on mining patterns from the data regarding application uses collected from various smartphone users for a long period of time. The smartphone industry and the application industry are booming, and therefore related studies become essential. The system and application developers are eager to know how users use applications on smartphones and so are the researchers. Using the pattern-growth approach, we develop an algorithm that can mine sequential patterns each of which satisfies a constraint on the maximum time interval between two application uses. Mining such patterns is not as trivial as it seems to be. If we apply a general sequential pattern mining algorithm to the data filtered by using the time constraint, we will not have all the patterns in which we are interested. If we use the widely used tool implementing the algorithm dedicated to mine sequential patterns satisfying the time constraint, we will soon realize that some patterns, especially long ones, are not returned and missed. We present the technical details of our algorithm and more importantly, we present, discuss, and visualize the mined patterns to gain a better understanding of how applications are used by smartphone users. Our contributions include an efficient and effective algorithm to mine all the time-constrained sequential patterns hidden in the data, which contains rich information, and a method for pattern visualization that is simple but has not been used in related papers. Finally, our mining results and findings could potentially assist in the development of applications, because they can hardly be found in the literature; additionally, our mining results and findings could potentially assist in research work related to smartphone or application uses, because they can initiate more and further studies.

Notations

a: An item, an application use
I: An itemset, a list of lexicographically ordered items

S: A session, a list of itemsets ordered by timestamps
D: A session database
P: A pattern
 $\sqsubseteq$: The containing relationship
 δ : The time constraint, the maximum time interval allowed
 θ : The minimum support required.

Competing Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The author would like to express his sincere gratitude to the anonymous reviewers for their constructive comments on the manuscript of this paper. The work presented in this paper was supported in part by the National Science Council of Taiwan under Grant no. NSC 102-2221-E-004-013 and the Ministry of Science and Technology of Taiwan under Grant no. MOST 103-2221-E-004-015. This paper was also supported in part by the X-Mind Research Group at the National Chengchi University, Taipei, Taiwan, sponsored by “Aim for the Top University Plan” of the university and the Ministry of Education of Taiwan. Their support is gratefully acknowledged.

References

- [1] B. Zhou, S. C. Hui, and K. Chang, “An intelligent recommender system using sequential web access patterns,” in *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, vol. 2, pp. 393–398, December 2004.
- [2] N. R. Mabroukeh and C. I. Ezeife, “A taxonomy of sequential pattern mining algorithms,” *ACM Computing Surveys*, vol. 43, no. 1, article no. 3, 2010.
- [3] J. Srivastava, R. Cooley, M. Deshpande, and P. Tan, “Web usage mining: discovery and applications of usage patterns from Web data,” *ACM SIGKDD Explorations Newsletter*, vol. 1, no. 2, pp. 12–23, 2000.
- [4] J. Pei, J. Han, B. Mortazavi-Asl et al., “PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth,” in *Proceedings of the 17th International Conference on Data Engineering*, pp. 215–224, April 2001.
- [5] J. Pei, J. Han, B. Mortazavi-Asl et al., “Mining sequential patterns by pattern-growth: the PrefixSpan approach,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424–1440, 2004.
- [6] P. Fournier-Viger, R. Nkambou, and E. M. Nguifo, “A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems,” in *Proceedings of the 7th Mexican International Conference on Artificial Intelligence*, pp. 765–778, Atizapán de Zaragoza, Mexico, October 2008.
- [7] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, “SPMF: a java open-source pattern mining library,” *Journal of Machine Learning Research*, vol. 15, article A28, pp. 3389–3393, 2014.
- [8] R. Agrawal and R. Srikant, “Mining sequential patterns,” in *Proceedings of the IEEE 11th International Conference on Data Engineering*, pp. 3–14, March 1995.

- [9] Y. Hirate and H. Yamana, "Sequential pattern mining with time intervals," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '06)*, pp. 775–779, Singapore, April 2006.
- [10] J. Pei, J. Han, and W. Wang, "Constraint-based sequential pattern mining: the pattern-growth methods," *Journal of Intelligent Information Systems*, vol. 28, no. 2, pp. 133–160, 2007.
- [11] N. Eagle and A. Pentland, "Reality mining: sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255–268, 2006.
- [12] S. Akoush and A. Sameh, "Mobile user movement prediction using Bayesian learning for neural networks," in *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC '07)*, pp. 191–196, Honolulu, Hawaii, USA, August 2007.
- [13] L. Wang, Y. Jia, and W. Han, "Instant message clustering based on extended vector space model," in *Advances in Computation and Intelligence: Second International Symposium, ISICA 2007 Wuhan, China, September 21–23, 2007 Proceedings*, Lecture Notes in Computer Science, pp. 435–443, Springer, Berlin, Germany, 2007.
- [14] K. Farrahi and D. Gatica-Perez, "What did you do today? Discovering daily routines from large-scale mobile data," in *Proceedings of the 16th ACM International Conference on Multimedia (MM '08)*, pp. 849–852, ACM, October 2008.
- [15] K. Farrahi and D. Gatica-Perez, "A probabilistic approach to mining mobile phone data sequences," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 223–238, 2014.
- [16] T. Bhattacharya, L. Kulik, and J. Bailey, "Automatically recognizing places of interest from unreliable GPS data using spatio-temporal density estimation and line intersections," *Pervasive and Mobile Computing*, vol. 19, pp. 86–107, 2014.
- [17] L. Xie, X. Zhang, J.-P. Seifert, and S. Zhu, "PBMDs: a behavior-based malware detection system for cellphone devices," in *Proceedings of the 3rd ACM Conference on Wireless Network Security (WiSec '10)*, pp. 37–48, Hoboken, NJ, USA, March 2010.
- [18] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11)*, pp. 15–25, Chicago, Ill, USA, October 2011.
- [19] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '10)*, pp. 179–194, ACM, San Francisco, Calif, USA, June 2010.
- [20] J.-M. Kang, S.-S. Seo, and J. W.-K. Hong, "Usage pattern analysis of smartphones," in *Proceedings of the 13th Asia-Pacific Network Operations and Management Symposium*, pp. 1–8, Taipei, Taiwan, September 2011.
- [21] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC '11)*, pp. 329–344, ACM, Berlin, Germany, November 2011.
- [22] J. K. Laurila, D. Gatica-Perez, I. Aad et al., "From big smartphone data to worldwide research: the mobile data challenge," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 752–771, 2013.
- [23] Z. Wang, M. A. Nascimento, and M. H. MacGregor, "On the analysis of periodic mobility behavior," in *Proceedings of the Mobile Data Challenge Workshop in Conjunction with International Conference on Pervasive Computing*, 2012.
- [24] P. Montoliu, A. Martinez-Uso, J. Martinez-Sotoca, and J. McInerney, "Semantic place prediction by combining smart binary classifiers," in *Proceedings of the Mobile Data Challenge Workshop, in Conjunction with International Conference on Pervasive Computing*, 2012.
- [25] H. Gao, J. Tang, and H. Liu, "Mobile location prediction in spatio-temporal context," in *Proceedings of the Mobile Data Challenge Workshop, in conjunction with International Conference on Pervasive Computing*, Newcastle, UK, June 2012.
- [26] X. Wu, K. N. Brown, and C. J. Sreenan, "Analysis of smartphone user mobility traces for opportunistic data collection in wireless sensor networks," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 881–891, 2013.
- [27] K. Huang, C. Zhang, X. Ma, and G. Chen, "Predicting mobile application usage using contextual information," in *Proceedings of the ACM Conference on Ubiquitous Computing (UbiComp '12)*, pp. 1059–1065, Pittsburgh, Pa, USA, September 2012.
- [28] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 113–126, Lake District, UK, June 2012.
- [29] P. Dai, W. Yang, and S.-S. Ho, "Predicting mobile call behavior via subspace methods," in *Proceedings of the 6th International Conference on Social Computing, Behavioral-Cultural Modeling & Prediction (SBP '13)*, pp. 466–475, Washington, DC, USA, April 2013.
- [30] E. H.-C. Lu, Y.-W. Lin, and J.-B. Ciou, "Mining mobile application sequential patterns for usage prediction," in *Proceedings of the IEEE International Conference on Granular Computing (GrC '14)*, pp. 185–190, IEEE, Noboribetsu, Japan, October 2014.
- [31] A. Nanopoulos and Y. Manolopoulos, "Finding generalized path patterns for web log data mining," in *Current Issues in Databases and Information Systems*, vol. 1884 of *Lecture Notes in Computer Science*, pp. 215–228, Springer, Berlin, Germany, 2000.
- [32] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '00)*, pp. 396–407, Kyoto, Japan, April 2000.
- [33] P.-M. Chen, C.-H. Chen, W.-H. Liao, and T.-Y. Li, "A service platform for logging and analyzing mobile user behaviors," in *Proceedings of the International Conference on E-Learning and Games, Edutainment*, pp. 78–85, Taipei, Taiwan, September 2011.
- [34] P.-M. Chen, H.-Y. Wu, C.-Y. Hsu, W.-H. Liao, and T.-Y. Li, "Logging and analyzing mobile user behaviors," in *Proceedings of the International Symposium on Cyber Behavior (CB '12)*, Taipei, Taiwan, February 2012.
- [35] W.-R. Tseng and K.-W. Hsu, "Mining sequential application usage patterns with time constraint of smartphone users," in *Proceedings of the International Conference on Cyber Behavior*, Taipei, Taiwan, June 2014.
- [36] W.-R. Tseng and K.-W. Hsu, "Smartphone app usage log mining," *International Journal of Computer and Electrical Engineering*, vol. 6, no. 2, pp. 151–156, 2014.
- [37] F. Masseglia, P. Poncelet, and M. Teisseire, "Pre-processing time constraints for efficiently mining generalized sequential patterns," in *Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME '04)*, pp. 87–95, Normandie, France, July 2004.

- [38] X. Yan, H. Cheng, J. Han, and D. Xin, “Summarizing itemset patterns: a profile-based approach,” in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 314–323, USA, August 2005.
- [39] The JUNG Development Team, “JUNG: The Java Universal Network/Graph Framework,” 2006, <http://jung.sourceforge.net>.