Using Timed Sequential Patterns in the Transportation Industry

Somayah Karsoum, Le Gruenwald and Clark Barrus
School of Computer Science
University of Oklahoma
Norman, OK USA
{somayah.karsoum,ggruenwald, clark.barrus}@ou.edu

Eleazar Leal
Computer Science Department
University of Minnesota Duluth
Duluth, MN USA
eleal@d.umn.edu

Abstract— **Data mining techniques can be used in several significant types of research in various domains to understand the flows of data, discover hidden knowledge, and improve quality of life. Sequential pattern mining, which is one data mining method, has emerged in the transportation domain to recognize the dynamic behavior of vehicles, trains, people...etc. The idea of sequential pattern mining is to find the frequent subsequences in a database of sequences. Although multiple sequential pattern mining techniques can mine sequential patterns, timestamps are only used to order sequences and time between sequence events is ignored. This information is important in real applications, such as traffic recommendation system and transportation safety. Though knowing that measurement Y occurs after measurement X is valuable, it is more valuable to know the estimated time before the appearance of measurement Y, perhaps, for example, to schedule maintenance at the right time in order to prevent railway damages. In this paper, we propose an algorithm called Minits (MINIng Timed Sequential patterns) to find the frequent sequential patterns and include the transition time between events in these patterns. However, approaches that depend on serial architecture are not effective anymore due to the massive data that are frequently generated from different sources. Therefore, we exploit parallelism using multicore CPUs to improve the performance of Minits to handle big data. Extensive experiments on real and synthetic datasets are reported and show the significance and advantages of this approach. Also, the execution time using a multicore outperform the single core when it deals with big data.**

Keywords— *Sequential pattern mining, Timed sequential patterns, single and multicore.*

I. INTRODUCTION

The goal of sequential pattern mining is to find a set of frequently repeating sequential patterns that occur in some database of ordered sequences. This problem was first introduced in [2]. Since then, many methods and techniques have been proposed to mine sequential patterns. Sequential pattern mining can be widely used in real applications such as transportation arrival time analysis, web log analysis, forecasting, and disease symptom analysis. Knowing frequent sequential patterns, we can answer a question like "in which order does the measurements(s) for train aerodynamic phenomena frequently occur?" For example, high temperature and high wind speed could cause a train crash by warping tracks or overturning of the lightly loaded train [21] [18]. Previously proposed techniques tried to improve the efficiency of methods to discover frequent sequential patterns but discard the time dimension. Timestamps are used to order events within a sequential pattern, but the transition time between these events is not retained. In many applications, it is important to know the time-range to move from one event to another.

For example, we may expect that the travel range of taxis that move from Broadway street to Times Square in NYC is 20 to 40 minutes. Also, in Florida during the hurricane season, we can estimate the transition time range when a tornado hits Miami, Fort Lauderdale, and West Palm Beach in order. In addition, knowing when the next symptom of heart attack will occur assists care providers with diagnosis, providing treatment at the right time, and intervening earlier in critical cases. With sequential patterns that contain temporal information about the transition time between signs, we can answer a question like when will the next measurements of train aerodynamics occur?

In Figure 1 we have the historic weather information (temperature and wind speed) of four sensors, which each sensor denoted as (S) with an ID as shown in the first column. The time is recorded with each measurement taken by each sensor as shown in the second column. Since sequential pattern mining algorithms do not deal with continuous data, we need to apply a partitioning technique in order to segment the data into classes that have similar features or fall within a same group. Therefore, we add an additional column next to each column that contains the equivalent class ID. For instance, the wind speed (W) has five levels [19] and each level refers to the damage that causes: (1) Minimal ($74 \leq W < 95$), (2) Moderate ($96 \leq W \leq 110$), (3) Extensive ($111 \leq W \leq 129$), (4) Extreme ($130 \leq W \leq 156$), and (5) Catastrophic ($W \geq 157$). Thus, the first row in the last column has the wind speed class 3 because the value 112 belongs to the class 3 (Minimal).

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

Sensor ID	Timestamp	Temperature	Temperature Class (TC)	Wind Speed Level	Wind Speed Class (WC)
S1	2/10/15 10:50:00	54	1	112	3
S2	4/9/13 3:15:00	45	1	75	1
S1	3/12/15 11:00:00	50	1	100	2
S2	4/9/13 9:00:00	41	1	115	3
S2	10/10/15 5:23:00	51	1	90	1
S3	5/13/13 2:20:00	47	2	125	3
S4	4/11/17 10:45:00	55	1	118	3
S3	7/23/16 4:29:00	50	1	99	2
S4	1/11/17 11:00:00	48	1	127	3

Figure 1. Sensors' historic weather information and discretize data

A timed event sequence that we want to discover is a sequence of frequently occurring measurements among sensors (or events) and typical transition times between these measurements (in terms of hours for the example case). The following is the format of a pattern that would be discovered in this study:

< (T1, W3) [30,36] W5>

The above sequence represents a timed event sequence that has two events. Anything between the parentheses ( ) appears at the same timestamp and the square bracket [ ] indicates the range time to move from the previous event to the next event. Thus, for the first event, the temperature from the first class and the wind speed from the third class appear together. Then, after thirty to thirty-six hours, the wind speed from class 5 follows that.

Recently, because of the availability of tracking devices such as GPS and sensors, increasing usage of social networks, and the rising amount of stored digital data, the existing traditional sequential pattern mining techniques suffer from many issues. These issues include long mining time, scalability problems, and limited memory. This leads to a need to implement new, fast, and scalable algorithms in order to solve these challenges in serial sequential pattern mining. Thus, instead of designing an algorithm which runs on a single single-core CPU, we design the Minits algorithm to work on multiple-core CPUs.

The contributions of this paper are the following:

- The idea of cooperating time-intervals between events in a frequent sequential pattern, which indicates the requiring time range to move from one event to another.
- The idea of using two different relations: event-relation and sequence-relation, which allow us to maintain the time-intervals between events.
- The idea of creating value indexes to indicate the position of dimension values in the database, so the original database can be projected easily and, thus, improve the efficiency of the algorithm.
- The parallel implementation of the Minits algorithm and the extensive experimental comparing single-core and multi-core on the datasets.

The remainder of the paper is organized as follows. Section 2 defines the timed sequential pattern mining problem. Section 3 reviews the related works. Section 4 describes the Minits algorithm. Section 5 presents the results of the experiments on datasets. Finally, the conclusion and future work are presented in Section 6.

II. Problem Definition

In this section, we first introduce the definitions and concepts used in this paper and declare the problem of timed event sequential pattern mining.

Definition 1. An *n-dimensional event database* is a set of records each of which corresponds to an object and contains identifying information, a timestamp, and an event. Each record has the following schema (I_1, I_2,….. ,I_m, T, D_1, D_2,…,D_n). The values of the *m* identifier attributes I_1, I_2,.., I_m and the timestamp, T, together uniquely identify each record while just the m identifier attributes together uniquely identify the object each event is associated with. D_1, D_2,…, D_n represent the *n* attributes that form the event held by each record. In an n-dimensional event database each event is a sequence of exactly n items.

Example 1. (**Running example**) The following example as shown in Figure 2 illustrates a 2-dimensional event database storing the weather condition measured by sensors. Each event captures the Temperature class (data attribute D_1) and the Wind speed class (data attribute D_2) at a particular time (attribute Timestamp) for a sensor (object) represented by its ID (identifier attribute I_1). The first two rows are two events associated with (object) sensor S1. It should be noted since a sequential pattern mining algorithm is not be able to deal with continuous values, we applied a discretization technique in order to segment the data into classes that have similar features or fall within a given interval. Therefore, we called the data attribute columns as Temperature class and Wind speed class.

Sensor ID	Timestamp	Temperature Class (TC)	Wind Speed Class (WC)
S1	2/10/15 10:50:00	1	3
S2	4/9/13 3:15:00	1	1
S1	3/12/15 11:00:00	1	2
S2	4/9/13 9:00:00	1	3
S2	10/10/15 5:23:00	1	1
S3	5/13/13 2:20:00	2	3
S4	4/11/17 10:45:00	1	3
S3	7/23/16 4:29:00	1	2
S4	1/11/17 11:00:00	1	3

Figure 2. 2-dimensional event database

Definition 2. An *n-dimensional event sequence database* is derived from an n-dimensional event database by grouping all the events associated with an object together and sorting them in the timestamp order. A record in this database has the following schema (I_1, I_2,…, I_m, (Timestamp, D_1, D_2,…, D_n)$^+$) where (…)+ means this set (…) can occur one or more times in the record and is sorted in the timestamp order. Thus, each record is a set containing identifying attributes and a sequence of timestamps and corresponding events < (ts_1 e_1), … (ts_i e_i)>. This sequence of timestamps and events is called an *event sequence*.

Example 2: The following Figure 3 illustrates a 2-dimensional event sequence database derived from the 2-dimensional event database given in Example 1. The first record shows the identifying information and the event

sequence which is a sequence of two events associated with the sensor S1 and sorted in the timestamp order. For the sake of simplicity, we replace timestamps as integer numbers, which could present the number of hours, days, weeks, months, etc. based on the preference of a user.

Sensor ID	Sequences
S1	< (5,T1, W3), (12,T1, W2)>
S2	< (5,T1, W1), (24,T1, W3), (28,T1, W1)>
S3	< (6,T2, W3), (19,T1, W2)>
S4	< (7,T1, W3), (9,T1, W3)>

Figure 3: Simplified 2-dimensional Event Sequence Database

Definition 3. A *timed event sequence* **(TES)** is a sequence of events (e_j) with an initially empty 2-tuple called transition time (t_j) between two events, where t_j = [t_j_min, t_j_max] represents the minimum and maximum transition time between events. A TES can be written as < e_1 [t_1] e_2 [t_2] e_3 …… [t_{j-1}] e_j>. For a TES we specify that each event has between one and n items whereas events in event sequences

A timed event sequence TES = < e_1 [t_1] e_2 [t_2] e_3 …… [t_{j-1}] e_j>, is a *subsequence* of an event sequence ES = <(ts_1 e'_1), … (ts_i e'_i)> if there exist indices n_1<n_2<…<n_j such that $e_1 \subseteq e'_{n_1}, e_2 \subseteq e'_{n_2}, \ldots e_j \subseteq e'_{n_j}$. We write TES ⊑ ES to denote that TES is a subsequence of ES. If TES ⊑ ES, then each transition time can be updated for all t_1 through t_{j-1}, by the following formula:

$$t_k\_min = minimum\left(t_k\_min, ts_{n_{k+1}} - ts_{n_k}\right) (1)$$

$$t_k\_max = maximum\left(t_k\_max, ts_{n_{k+1}} - ts_{n_k}\right) (2)$$

In this paper, the transition time will only be updated once for each event sequence and will use the lowest possible indices n if different indices are possible.
always have n items.

Example 3: The following are examples of timed event sequences.
< T1>, < W3 [] T1>, < (T1, W3)>, and < (T1, W3) [] W1>

Definition 4. The *support* of a timed event sequence, TES, is defined as the percentage of event sequences in an event sequence database of which TES is a subsequence. This can be expressed using the following formula:

$$support\ (TES) = \frac{|\{ES \mid ES \in Database \wedge TES \sqsubseteq ES\}|}{|\{ES \mid ES \in Database\}|} * 100\% \quad (3)$$

Definition 5. A *timed event pattern (TEP)* is a timed event sequence that satisfies the minimum support threshold and is therefore considered to frequently occur in an event sequence database. The minimum support threshold (min_sup) is a user defined value between 0% and 100% and determines the necessary support of a TES for that TES to be considered a TEP. A timed event pattern also has transition time updated for each event sequence that the timed event pattern is a subsequence of, such that the transition times reflect the minimum and maximum transition times between events.

Example 4 The following example illustrates the testing of the first three timed event sequences from Example 3 to see if they are timed event patterns assuming min_sup=75%. The first timed event sequence <T1> is a subsequence of four event sequences and has support (<T1>) = 4/4 * 100 = 100% and thus is a timed event pattern. Since this pattern has length one there are no transition times to be calculated. The second timed event sequence <W3 [] T1> exists in four records so, its support = 100%. The transition time tuple is updated for each event sequence of which the pattern is a subsequence. For example, after the first subsequence test we now write the timed event sequence as (<W3 [7,7] T1>) because the transition time is ts_2 -ts_1 = 12 - 5 = 7. After considering the TES to be a subsequence of S2 we can update our transition time again to be (<W3 [4,7] T1>) as the transition time for S2 is ts_3 -ts_2 = 28 - 24 = 4 and this difference is less than the previous minimum transition time, 7. After the calculation with S3 we write (<W3[4,13] T1>). Finally, after computing that the TES is a subsequence of S4, we write the TES as (<W3 [2,13] T1>) because for S4 the difference (ts_2 -ts_1 = 9 - 7 = 2) is less than minimum transition time, which was 4. Since (<W3[2,13] T1>) is a subsequence of the whole database this TES is a TEP. The third TES < (T1, W1)> is an event with the Temperature class 1 (attribute D1) and the Wind speed class 1 (attribute D2), which is a subsequence of only one event sequence S2. Therefore, its support (< (T1, W1)>) = 25%. Since *min_sup*= 75% this timed event sequence is not a timed event pattern.

Problem Statement. Given an n-dimensional event sequence database and a minimum support threshold, find the complete set of timed event patterns.

III. Related work

The problem of sequential pattern mining first introduced in [2], where the AprioriAll algorithm was proposed for discovering sequential patterns. It produced frequent patterns by the method of candidates' generation and test. Since then, many methods and techniques have been proposed to mine sequential patterns. For example, the GSP algorithm [22] inspired a technique to generate fewer candidates based on the Apriori algorithm for frequent itemset mining. Therefore, it is classified as an Apriori-based algorithm and later many algorithms adopted this concept and incorporated other techniques to improve the performance such as SPADE [25] and MobileSPADE [3]. In contrast, pattern-growth based algorithms, such as FreeSpan [11] and PrefixSpan [17], used the concept of database projection to improve the performance of mining sequential patterns. More literature reviews about the state-of-art sequence mining algorithms can be found in [7] Recently, since a large volume of data is being generated, many of the proposed sequential pattern mining algorithms tried to handle huge amounts of sequences and large databases efficiently. Ha-

GSP [16] adopted the principles of GSP and implemented them on the Hadoop platform for solving the limited computing capacity and insufficient performance with massive data of the traditional GSP. MR-PrefixSpan [23] used the MapReduce platform to implement the parallel version of PrefixSpan to mine sequential patterns on a large database. More literature reviews about the state-of-art parallel sequence mining algorithms are in [8].

In sequential patterns, objects have ordinal correlation based on the timestamp precedence. We can obtain a sequence by sorting all these objects based on their order of timestamps. However, the time between events is discarded. This kind of sequential pattern that incorporates time-intervals is more informative for some applications. Some techniques were proposed to specify some timing constraints, such as the time gaps between adjacent events in sequential patterns. MOWCATL [12] found episodal association rules from multiple sequential data sets that satisfy the time lag constraint. Also, [4] modified Apriori [2] and PrefixSpan [17] algorithms to discover the time-interval sequential patterns that satisfied the interval duration boundaries. I-PrefixSpan [4] algorithm has another input which is called set of time-intervals TI, where each time-interval has a range. The drawback of applying these methods is missing some frequent rules or patterns that do not fulfill the time constraint (even if they are frequent). In order to decide if a pattern is frequent, there are two conditions that must be satisfied: the support of the pattern must be greater than or equal to min_sup and the time range between itemsets must lie within the defined time intervals. Therefore, if a pattern fulfills the first condition, which could be considered as frequent, but not the second condition, the algorithm will not produce it. For example, suppose we have the 1-dimensional event sequence database as shown in Figure 4, and min_sup = 50%, and PI = { I_0, I_1, I_2, I_3}, where I_0: t =0, I_1 : 0 < t <= 3, I_2 : 3 < t <=6, and I_3 : 6 < t <= ∞. The pattern < T0, I_0, T1, I_1, T4 > is a frequent sequential pattern because it has support = 75%, which means that there are three sequences (S1, S2, and S3) that contain the three items appearing in this order T0, T1, and T4 such that the time difference between T0 and T1 lies within I_0 and the time difference between T1 and P4 lies within I_1. However, the pattern <T1, I_1, T4, I_2, T3> is frequent because two sequences (S2 and S3) contain the three items in the same order T1, T4, and T2, but it is not frequent in terms of time-interval because the time difference between T4 and T3 do not fall within I_2 in sequence S3. Thus, the output of the algorithm would be the only frequent sequential patterns that satisfy the time-intervals.

Sensor ID	Sequences
S1	< (T0, 1) (T2, 3) (T0, 4) (T1, 4) (T0, 6) (T4, 6) (T2, 10) >
S2	< (T3, 5) (T0, 7) (T1, 7) (T4, 7) (T3, 9) (T4, 9) (T2, 14) (T3, 14)>
S3	< (T0,8) (T1, 8) (T4, 11) (T3, 13) (T1, 16) (T2, 16) (T2, 20) >
S4	<(T1, 15) (T5, 17) (T4, 18) (T1, 22) (T2, 22) >

Figure 4: 1- dimensional event sequence database

As a special case of sequential pattern mining, [9] incorporated the temporal dimension in the sequential patterns by defining temporally annotated sequences (TAS), and [10] proposed the Trajectory Pattern algorithm (T-pattern) to extract a set of TAS to produce trajectory patterns with a fixed amount of time to travel between places. [24] relaxed the travel time to be a range traveling time in reality. Nevertheless, these algorithms cannot deal with multidimensional data because one location (one dimension) can be registered at the same timestamp.

To the best of our knowledge, there is no existing technique that can find the complete set of timed event sequential patterns that represent the transition time between successive events in a pattern. Therefore, in our study, we re-design PrefixSpan algorithm to find the timed event sequential patterns. The reason for choosing PrefixSpan is because it is one of the well-known algorithms for discovering sequential patterns. It was proven that it produces complete and correct sequential patterns and outperforms other algorithms such as GSP [22] and FreeSpan [11] However, FAST [20] algorithm showed that is outperformed PrefixSpan but there wasn't any proof correctness of the algorithm.

IV. The Proposed Algorithm: Minits

A. Overview

In order to find the complete set of the timed event sequences, the following steps are performed in our algorithm: (1) For each distinct value v_i of each data attribute D_j, compute the support and when it is frequent, then add it to the set of frequent values called FV, (2) For each frequent value v_i, build the pseudo-projected database which is a projection of the database that collecting all pointers that refer to the sequences in the original database in the main memory as a pseudo-projection, (3) Compute the occurrences of event-relation and sequence-relation between v_i and each value in FV, which are defined as:

Definition 8. An *Event-relation* (e-relation): Given two values X and Y of different data attributes, it is said X and Y have an e-relation between them denoted as < (X, Y)> if X and Y occur in the same event.

Definition 9. A *Sequence-relation* (s-relation): Given two values X and Y of different data attributes, it is said X and Y have an s-relation between them denoted as < X Y> if X and Y occur in two different event and the event of X occurs before the event of Y.

(4) Output the result of appending f_K, which is an element from FV, to v_i considering the type of the relation, either event or sequence relation; (5) Update FV to contain the new frequent values in pseudo-projected database; and (6) Repeat steps 2, 3, 4 and 5 until the algorithm discovers no new frequent Timed sequential Patterns TEP. The pseudo code of the Minits is shown in Figure 5. Before we explain the steps in details, we have the following definitions:

In the beginning, the support of each distinct value v_i for each data attribute D_j is computed by scanning the database, then it is checked against the minimum support threshold to

decide if the value is frequent or not. If it is, then the value vi is added to FV. After that, the elements of FV will be added to another set called Frequent Patterns (FP), which represents the complete set of TEPs that are discovered by the algorithm since they are considered to be timed event sequential patterns of length 1. In the second step, the algorithm divides the search space into a number of partitions equal to the number of frequent values in FV and builds a projected database for each v_i. The projected database is defined as a collection of postfixes of sequence in the database S w.r.t. prefix α and denoted as S|α [17]. For example, let prefix α = T1 and S is the database that is shown in Table 3. The projected database S|T1 is shown in Figure 6. Also, in the projected database, the only frequent values are kept because according to Apriori property [1] any infrequent value should be eliminated because it will not generate any frequent sequential patterns. Thus, the algorithm generates candidates based only on frequent values detected in the most recent projected database and only calculates the support by only checking if the non-empty projected sequences contain the candidate TEP. This minimizes the overhead of generating candidates by only inspecting the postfix sequences to generate and calculate the support of candidate TEP.

Sensor ID	Sequences
S1	< (5, W3,), (12,T1)>
S2	< (45), (24, T1, W3,), (28, T1)>
S3	-
S4	< (7, W3,), (9, T1, W3,)>

Figure 6: Projected database for prefix T1 - $S|_{T1}$

In the next step, the algorithm scans the projected database and counts the support of e-relation and s-relation between v_i and each distinct value f_k in FV. The goals of having two kinds of relationships are to distinguish different links between values and identify the time-intervals. For instance, suppose we have the values T1 and W3 from two different data attributes Temperature, and Wind speed, respectively. There are two possible links between them: If T1 and W3 appear at the same event, then they have the same timestamp (i.e., e-relation) or if T1 and W3 appear at different events, then they have different timestamps (i.e., s-relation). For each case, there are different patterns that can be discovered. Therefore, we need to count the support of each possible patterns and consider the type for the relationship. For the second objective, finding the time-intervals, the timestamp of any e-relation must be the same for v_i and f_k, thus the time interval is always be zero. For s-relation, the timestamp must be different for v_i and f_k, hence, the time is always re-calculated according to Line 24 and then the time interval is updated to always hold the minimum and maximum time. By the end of this step, we compare the support of each relation against min-sup to keep only the frequent relations. For example, we found that T1 occurs 75% of the time with T1, and they have an s-relation between them and the

Algorithm 1 (Minits)

Input: n-dimensional event sequence (DB),
minimum support threshold (min_sup)
Output: Frequent Patterns set (FP) that contains all frequent Timed Event patterns TEP

```
1   for each data attribute Di
2     Compute the support of each distinct value;
3     Add frequent values to Frequent Values set (FV);
4   end for
5   Add all elements of FV into (Set_FP);
6   Call Find _TEP (DB, Prefix, FV); // Prefix = Ø at the first call
7   Function Find _TEP (DB, Prefix, FV) {
8       Min_time, Max_time = 0; // to hold the minimum and
          maximum time intervals between events
9   for each value vi in FV
10    Build vi- pseudo projection DB from DB;
11      if (vi- pseudo projection DB is empty) then
12        continue
13      else
14        for each sequence s in the vi- pseudo projection DB
15          for each event ej in a sequence
16            for each value fk in FV
17              if (there is an e-relation between vi and fk in the
                    event ej) then
18                e-relation support count++
19              else if (there is a s-relation between vi in event
                        ej and fk in the event ej+1) then
20                s-relation support count++
21                time = ej+1.timestamp – ej. timestamp;
22                if (time < Min_time) then Min_time = time;
23                if (time > Max_time) then Max_time = time;
24                time-interval = [Min_time, Max_time];
25              end if
26            end for
27          end for
28        end for
29      end if
30      Update FV to contain new frequent events nl, where
          the support of vi ∪ nl >= min_sup
31      for each values nl in FV
32        if (vi and nl has an e-relation) then
33          Prefix = < (vi ∪ nl)>
34          Add Prefix into FP
35          Call Find _TEP (vi- pseudo projection DB, Prefix, FV)
36        else if (vi and nl has a s-relation) then
37          Prefix = vi ∪ nl
38          Add < vi ∪ [time-interval] ∪ nl> into FP
39          Call Find _TEP (vi- pseudo projection Db, Prefix, FV)
40        end if
41      end for
42  end for
43  }
```

Figure 5. Pseudo-code of the Minits Algorithm

transition time to move from event T1 to another event T1 is [2,19].

In its fourth step, the algorithm distinguishes the frequent patterns by comparing its support against min-sup and distinguishes the type of relationship. As it was mentioned above, if the s-relation between T1 and T1 has support >=

75% and the min-sup = 75%, then <T1 [2,19] T1> is a timed event sequential pattern. Also, for instance, the algorithm found that T1 has also an e-relation with W3, which occurs 75% of the time. Therefore, the < (T1, W3)> is another timed event sequential pattern. Then, the algorithm will update FV to contain the new frequent distinct values T1 and W3 for Prefix T1, so FV = {T1, W3}. In its final step, the algorithm repeats Steps 2,3, and 4 by building a projected database for each value in FV, computing the supports of the e-relation and s-relation and finding TEP. The aim of this step is extending a short pattern to a longer pattern that has the same prefix. The algorithm terminates after no new TEP can be found, which means the projected database is empty.

B. *The Proposed Enhancement*

In this section, we describe some effective mechanisms that help to improve the efficiency of Minits.

1) Using pseudo projection technique

The drawback of the previous technique is that building marginally smaller copies of the database for each projection and saving them into main memory until they are needed becomes prohibitively expensive. Therefore, another technique called pseudo projection is proposed in [17] that may improve mining efficiency. Instead of maintaining copies of sequences for each projected database, the pseudo projection technique maintains the original database in memory and maintains a collection of pointers for each projected database. A projected database will maintain pairs of information: (1) a pointer to a sequence and (2) an offset of the postfix in the sequence. For example, the Pseudo projection for prefix T1 contains four pointers and each pointer refers to a sequence. For the sequence Si1 the offset is 3, which indicates that the projection starts from that position. For sequences S2, S3, and S4 the offsets are 3, 6 and 3, respectively. Since the PrefixSpan algorithm does not deal with timestamps, we need to modify this technique to apply it to Minits. Therefore, the modified Pseudo projection will contain the following items: (1) the index to a sequence and (2) the index of a position of the value, which includes the event ID within a sequence and the value ID within an event. Since each timestamp is are kept for calculating time intervals and it is not a value that can generate any patterns, always its position is ignored. Figure 7 shows the modified pseudo projection database for prefix T1.

Frequent 1-Seq	SID 1	SID 2	SID 3	SID 4
(T1)	1,1	1,1	2,1	1,1
…	…	…	…	…

Figure 7: Pseudo projection database for prefix T1 - $S|_{T1}$

For instance, the first row is pointing to the first sequence Si1 in the original database, where T1 appears at event 1 and position 1, thus the index is [1,2]. It means the projection starts from that index. We call the version of Minits that uses the physical projected database *Minits-projected DB* and the modified version of Minits that replaces the physical projected database with the modified pseudo projection database *Minits-pseudo DB*.

2) Using Multicore CPUs

Another enhancement is using multicore CPUs for implementing Minits, which we call *Minits-multicore.* recursive calls to the function *Find _TEP* are divided among multiple threads. A queue of jobs, which are finding all possible candidates and deciding if they are frequent, is created and those jobs are assigned to idle threads. All threads are working to complete the task at the same time, and this reduces the execution time of Minits. The below Figure 8 shows the mechanism of Minits-multicore. When the algorithm first discovers that T1 and W3 are frequent TEP, the two patterns are inserted into the queue and waiting for idle threads to work on extending them and generating candidates. After calculating the support, the algorithm decides that <T1T1>, <T1 W3> and < W3 T1> are the new TEP. Therefore, these three patterns are inserted into the queue and waiting for other idle threads to be launched at the

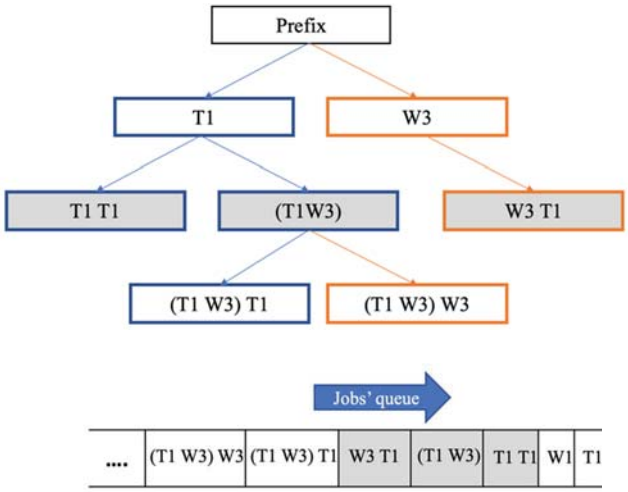


Figure 8: The mechanism of multicore CPUs of Minits

V. PERFORMANCE ANALYSIS

In this section, we report the results of the experiments to test our Minits that are implemented in single and multicore CPUs and compare their performance on real and synthetic data. After running many experiments, we have found that Minits-pseudo DB on a single-core CPU performed much better than Minits-projected DB on a single-core CPU. Therefore, for the next experiments, we have chosen Minits-pseudo DB and reimplemented it on a multicore CPU. From this point on, when we refer to Minits, we mean the Minits-pseudo DB version, and we distinguish between the single-core CPU and multicore CPU implementations by using the following abbreviation: Minits-single and Minits-multi.

A. Experimental Setup

All experiments were performed on a computer with a 2.10 GHz Intel Xeon(R) processor with 62-gigabyte main memory, running Ubuntu 18.04.1 LTS. The Minits algorithm is implemented in Java 1.8.

B. Datasets and Experimenal Parameters

We use one real-life dataset and one synthetic dataset. The real dataset is T-Drive [13] [14] and the synthetic dataset is generated using a tool provided by the SPMF Library [6]. Also, we set a number of parameters to conduct the experiments on each dataset. There are two types of parameters: static parameters and dynamic parameters. The values of the static parameters are not changed in all experiments. In contrast, the values of the dynamic parameters are changed from one experiment to another experiment. Basically, we have four different parameters. The first one is minimum support threshold (min_sup). It is a user-defined threshold applied to find all the frequent sequential patterns in n-dimensional events sequence database. The second parameter is the number of sequences in the event database (# sequences in DB). Since each sequence is represented as a tuple in the database, this parameter refers to the number of tuples in the database. We conducted the experiments on all datasets and the results were consistent. We will explain the type of parameter and for the dynamic parameters we will explain their ranges, and their default values of this analysis as they are summarized in Tables 1 and 2 for the T-Drive and synthetic dataset, respectively. When an experiment was conducted, we chose various values of one parameter within its range and assigned the default value to the other parameters.

T-Drive is a collection of trajectories gathered by Microsoft Research Asia after tracking the movements of 10,357 taxis in Beijing, China for one week. The dataset contains the following attributes: User ID, timestamp, latitude, and longitude as shown in Figure 9. For example, Taxi 1 has a sequence that contains many events to represent its movements. An event (2008-10-23 02:53:04, 39.93, 116.31) refers to (timestamp, latitude, longitude) respectively. Since the sequential pattern mining algorithm cannot deal with continuous data, we discretized the data first by using a density-based clustering algorithm called Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [5] and the results of the discretized sequences are shown in Figure 10. Taxi 1 has now a sequence that contains events in terms of clusters ID. For instance, event (2008-10-23 02:53:04, C1) refer to (timestamp, cluster Id) respectively. DBSCAN generates a number of clusters that contain the close points and replaces the latitude and longitude of a point with a cluster ID (Ci). For more details, we refer the readers to [15].

One of the dynamic parameters is (min-sup) that has a range from 20% to 80% with the default value = 50%, which is the median of the interval. Then, the number of sequences ranges from 1 to 2000 and its default value (median) is 1000.

Parameter Name	Range of values	Default value
min_sup	20% - 80%	50%
# sequences in DB	1-2000	1000

Table 1: Parameters list for T-Drive

Taxi ID	Trajectory sequence
1	< (2008-10-23 02:53:04, 39.93, 116.31),……., (2008-10-23 11:11:12, 40.00, 116.32) >
2	< (2008-10-23 12:45:23, 39.92, 116.33),……., (2008-10-23 16:44:22, 39.93, 116.34) >
3	…
…	…

Figure 9: Sequential database for the T-Drive dataset before discretization by DBSCAN

Taxi ID	Trajectory sequence
1	< (2008-10-23 02:53:04, C1),……., (2008-10-23 11:11:12, C2) >
2	< (2008-10-23 12:45:23, C1),……., (2008-10-23 16:44:22, C4) >
3	…
…	…

Figure 10: Sequential database for the T-Drive dataset after discretization by DBSCAN

As mentioned before, the synthetic dataset is generated by using the implementation tool in [6]. We study the impacts of all five parameters shown in Table 2. The support (min-sup) parameter has a range from 20% to 80% with the default value = 50%, which is the median of the interval. The range of the parameter (number of sequences in a dataset) is chosen to be from 1 to 100,000 and its median value of 50,000 to be the default value.

Parameter Name	Range of values	Default value
min_sup	20% - 80%	50%
# sequences in DB	1-100,000	50,000

Table 2: Parameters list for synthetic dataset

C. Evaluation Metrics

The evaluation metrics include two measurements: (1) the execution times (ET) of the two algorithms (Minits-single and Minits-multi) and (2) the number of patterns (# patterns) that are generated by these two algorithms.

D. Experimental Results

In this section, we present the performance of the two algorithms, Minits-single and Minits-multi, in terms of execution time (ET) and number of discovered patterns (# patterns) for the T-Drive and synthetic datasets. Again, the Minits is inspired by PrefixSpan, which is a well-known algorithm for mining sequential patterns. Even though there are other algorithms that outperform PrefixSpan, such as FAST [20], but there is any proof that shows the correctness of the algorithm.

1) Accuracy

In order to validate that Minits always gives the same patterns in terms of numbers and contents excluding the time-intervals as those produced by PrefixSpan [17], a tool was implemented in Java 1.8.0_91 using Eclipse version 4.5.2.

First, all time-intervals were removed from the patterns that were generated by Minits. Then, these patterns were compared to the patterns that were generated by PrefixSpan to make sure that each pattern generated by one algorithm has a match generated by the other algorithm. For example, a pattern X was generated by PrefixSpan X= < T1 W3 (T1W3) > and a pattern Y was generated by Minits Y= < T1 [2,5] W3 [3,7] (T1W3) >, we took away the time intervals from Y and compared it with the pattern X. In case the order of at least one value was different, the pattern X was not matching the pattern Y. For instance, Z= < W3 [2,5] T1 [3,7] (T1W3) > was not matching pattern X because the value W3 occurs before T1. However, within the last event (T1W3) the order does not matter because all the values appear at the same timestamp. At the end of this experiment, we found that both algorithms: Minits-single and Minits-multi discovered the exact same patterns that were produced by PrefixSpan. In other words, both algorithms produced accurate sequential patterns.

2) Execution Time

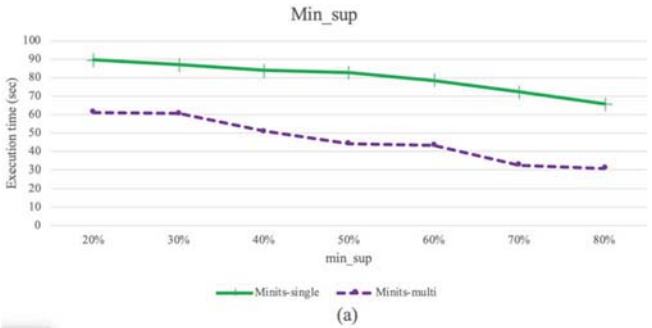
The execution time (in seconds) is recorded starting from the moment that a dataset has been read to the moment that the algorithm produces the patterns. Table 3 shows the average performance of the two algorithms for the T-Drive and synthetic datasets. Table 3 shows that the execution time (ET) of Minits-multi decreases by about 74% compared to that of Minits-single.

Datasets	Minits-singlecore		Mintis-multicore	
	ET (sec)	#patterns	ET (sec)	#patterns
T-Drive	81.413	126	47.604	126
Synthetic data	25.32	3756	6.60	3756

Table 3: Average Execution times and # patterns of the algorithms on the T-Drive and synthetic dataset

3) Impact of Miniumum Support

In these set of experiments, we compared execution time (ET) and number of patterns (# patterns) for different minimum support thresholds (min_sup). From Figure 11, we can see that when the minimum support increased, the execution time of both algorithms decreased. The reason is that the two algorithms generate fewer patterns when the min_sup is high because fewer sequences satisfy the min_sup condition. With a large amount of data and a huge number of discovered patterns, Minits-multi outperformed Minits-single as shown in Figure 11(b). Therefore, using multicore CPUs is more useful when the sequence database and the number of expected patterns are huge.



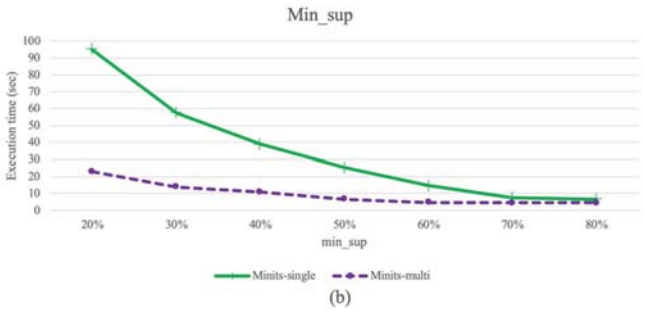


Figure 11: Impact of min_sup on execution time using (a) T-Drive dataset or (b) synthetic dataset

Also, the multicore CPU version was efficient when we have low min_sup. As we observed from Figure 11(b), the ETs of both Minits-single and Minits-multi were very close when the min-sup is greater than 60%. The reason is that the number of candidates, and thus the number of patterns, was getting smaller, so there was no need for a large number of threads.

Another observation was based on the number of frequent timed event patterns that were generated by these algorithms. Both algorithms discovered the same number of patterns, thus their curves were overlapping in Figure 12 and 14. When the min-sup increases, the number of patterns that Minits decreases because the patterns that satisfy the min-sup condition become fewer. By increasing the threshold min_sup, the percentage of sequences in the databse that a candidate sequence must be a subsequence of increase and thus a sequence must be more occurring more frequently in order to be returned by the algorithm. This becomes a rare case especially when the min-sup keeps increasing.

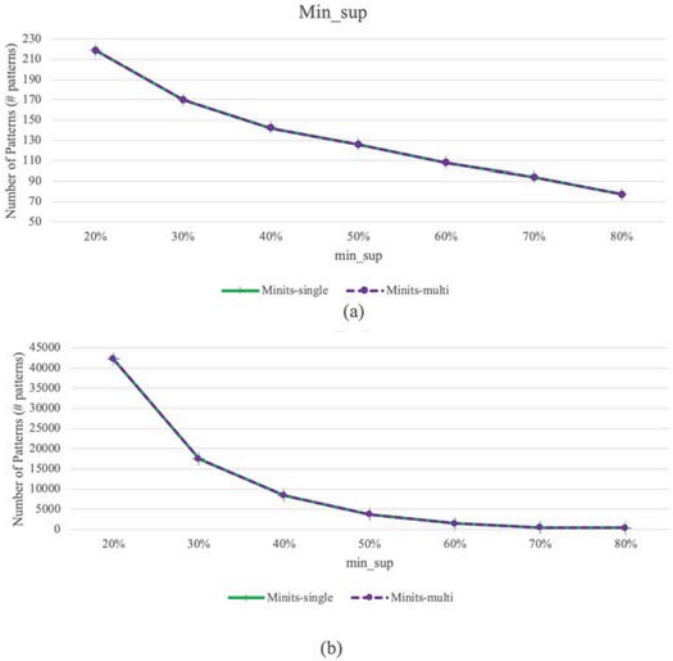


Figure 12: Impact of min_sup on number of patterns using (a) T-Drive dataset or (b) synthetic dataset

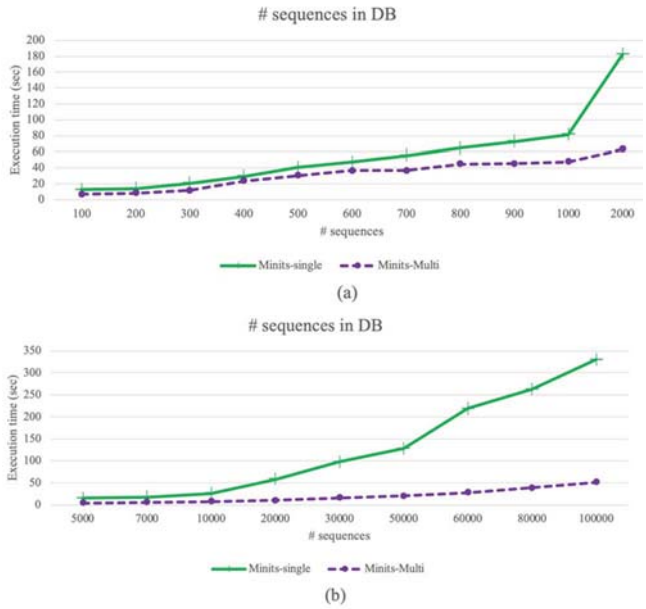
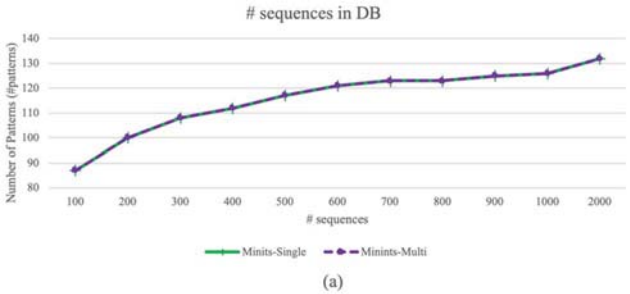


Figure 13: Impact of # sequences in database on execution time using (a) T-Drive dataset or (b) synthetic dataset

Another observation was based on the number of frequent timed event patterns that were generated by these algorithms. When the number of sequences increased as shown in Figure 14, the number of patterns that were discovered by Minits-single and Minints-multi increased because the possibility of finding more patterns in new sequences that satisfy the min-sup (50% as the default value) condition increased. By increasing the number of sequences in the database, Minits needs to check if there are some new patterns that can occur and do not exist in the old sequences. Then, it checks their support against the threshold (min_sup). Also, maybe the support of some previous patterns does not satisfy the min_sup condition because they are not supported by a sufficient number of sequences. However, adding new sequences into the database change the support of these patterns, which change the status of them to be frequent timed event patterns. Thus, the number of newly discovered frequent timed patterns will increase.

For example, if a database had 5000 sequences in synthetic dataset, the number of patterns was 3758, while the number of patterns was 3780 when the database had 20000 sequences.



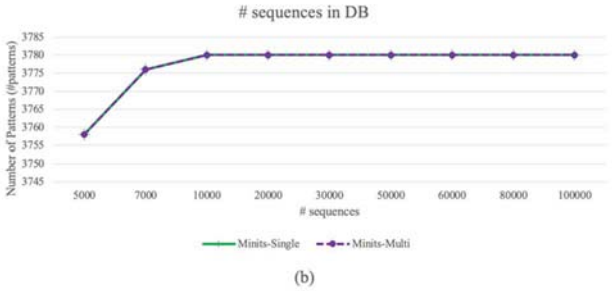


Figure 14: Impact of min_sup on number of patterns using (a) T-Drive dataset or (b) synthetic dataset

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented an algorithm, called Minits, to discover timed event sequential patterns which are frequent sequential patterns that include the transition times between their events. We implemented two versions of Minits: Minits-single on single-core CPUs and Minits-multi on multi-core CPUs. We conducted experiments comparing the accuracy and the execution time of the two algorithms. The experiments showed that both the algorithms produced accurate patterns, but Minits-multi outperformed Minits-single when the dataset is large. For future work, we plan to implement Minits on GPUs to further improve it for Big Data where not only the numbers of sequences and events are large, but also the number of items is large.

ACKNOWLEDGMENT

This work was partially supported by the NSF grant #1302439 and the BHGE grant # 19-0069.

REFERENCES

[1] Agrawal, R., & Srikant, R. (1994, September). Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB (Vol. 1215, pp. 487-499).

[2] Agrawal, R., Srikant, R. Mining sequential patterns. In: Proceedings of the 11th IEEE International Conference, pp. 3-14, IEEE, Taiwan (1995).

[3] Bradley, J., & Rashad, S. Time-based location prediction technique for wireless cellular networks. In Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering, pp. 937-947, Springer, New York, NY(2013).

[4] Chen, Y. L., Chiang, M. C., & Ko, M. T. Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications, Vol.25, No.3, pp.343-354, (2003).

[5] Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In Kdd, vol. 96, no. 34, pp. 226-231, 1996

[6] Fournier-Viger, P., Lin, C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H. T. (2016). The SPMF Open-Source Data Mining Library Version 2. Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, Springer LNCS 9853, Â pp. 36-40.

[7] Fournier-Viger, P., Lin, J. C. W., Kiran, R. U., Koh, Y. S., & Thomas, R. A survey of sequential pattern mining. In Data Science and Pattern Recognition, pp. 54-77, (2017).

[8] Gan, W., Lin, J. C. W., Fournier-Viger, P., Chao, H. C., & Yu, P. S. (2018). A survey of parallel sequential pattern mining. arXiv preprint arXiv:1805.10515.

[9] Giannotti, F., Nanni, M., & Pedreschi, D. Efficient mining of temporally annotated sequences. In Proceedings of the 2006 SIAM International Conference on Data Mining, pp. 348-359, (2006).

[10] Giannotti, F., Nanni, M., Pinelli, F., & Pedreschi, D. Trajectory pattern mining. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 330-339, (2007).

[11] Han, Jiawei, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. "FreeSpan: frequent pattern-projected sequential pattern mining." In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 355-359, (2000).

[12] Harms, S. K., & Deogun, J. S. Sequential association rule mining with time lags. Journal of Intelligent Information Systems, Vol.22, No.1,pp. 7-22, (2004).

[13] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10, pages 99-108, New York, NY, USA,2010. ACM.

[14] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In The 17th ACM SIGKDD international conference on Knowledge Discovery and Data mining, KDD'11, New York, NY, USA, 2011. ACM.

[15] Karsoum, S., Gruenwald, L., & Leal, E. (2018, December). Impact of Trajectory Segmentation on Discovering Trajectory Sequential Patterns. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 3432-3441). IEEE.

[16] Li, Huanhuan, Xiaofeng Zhou, and Chaojun Pan. "Study on GSP algorithm based on Hadoop." In *Electronics Information and Emergency Communication (ICEIEC),5th IEEE International Conference*, pp. 321-324, (2015).

[17] Pei, Jian, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth." In *icccn*, p. 0215, (2001).

[18] Rossetti, M. A. (2007). Analysis of weather events on US Railroads.

[19] 19 -Simpson Hurricane Wind Scale. Retrieve from https://www.nhc.noaa.gov/aboutsshws.php

[20] Salvemini, E., Fumarola, F., Malerba, D., & Han, J. (2011, June). Fast sequence mining based on sparse id-lists. In International Symposium on Methodologies for Intelligent Systems (pp. 316-325). Springer, Berlin, Heidelberg.

[21] Simes,T. (October, 2011). A Blow to Train Operations, Can strong winds cause derailment? In International Railway Saftey Conference. Melbourne: Australia.

[22] Srikant, R., Agrawal, R.Mining sequential patterns: Generalizations and performance improvements. In: International Conference on Extending Database Technology, pp.1-17, Springer, Berlin, Heidelberg (1996).

[23] Wei, Yong-qing, Dong Liu, and Lin-shan Duan. Distributed PrefixSpan algorithm based on MapReduce. In *IEEE Information Technology in Medicine and Education (ITME),International Symposium on*, vol. 2, pp. 901-904, (2012).

[24] Yang, H., Gruenwald, L., & Boulanger, M. A novel real-time framework for extracting patterns from trajectory data streams. In Proceedings of the 4th ACM SIGSPATIAL International Workshop on GeoStreaming, pp. 26-32, (2013).

[25] Zaki, Mohammed J. SPADE: An efficient algorithm for mining frequent sequences. Machine learning 42, no. 1-2, (2001).