Mining Timed Sequential Patterns: The Minits-AllOcc Technique

Somayah Karsoum [a,1], Clark Barrus [a], Le Gruenwald [a], and Eleazar Leal [b]
[a] University of Oklahoma, Norman, OK 73019, USA
E-mail:somayah.karsoum@ou.edu ; ORCID: https://orcid.org/0000-0002-8855-0175
[a] University of Oklahoma, Norman, OK 73019, USA
E-mail: clark.barrus@ou.edu ; ORCID:
[a] University of Oklahoma, Norman, OK 73019, USA
E-mail: ggruenwald@ou.edu; ORCID:
[b] University of Minnesota Duluth, Duluth, MN 55812, USA
E-mail: eleal@d.umn.edu; ORCID:

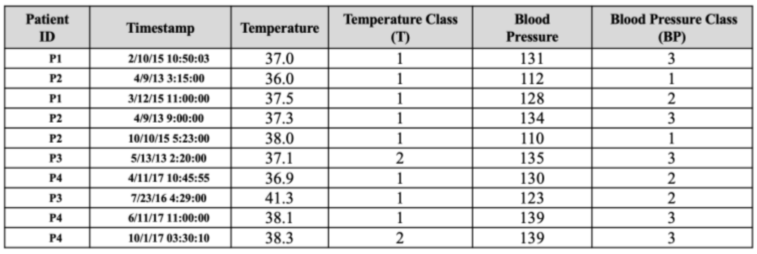
Abstract. Sequential pattern mining is one of the data mining tasks used to find the subsequences in a sequence dataset that appear together in order based on time. Sequence data can be collected from devices, such as sensors, GPS, or satellites, and ordered based on timestamps, which are the times when they are generated/collected. Mining patterns in such data can be used to support many applications, including weather forecasting and transportation recommendation systems. Numerous techniques have been proposed to address the problem of how to mine subsequences in a sequence dataset; however, current traditional algorithms ignore the temporal information between the itemset in a sequential pattern. This information is essential in many situations. For example, doctors, even if they know a symptom B will appear after symptom A for a specific disease, must know the time interval of when symptom B is expected to appear to reduce the disease's risk and provide a suitable treatment. Considering temporal relationship information for sequential patterns raises new issues to be solved, such as designing a new data structure to save this information and traversing this structure efficiently to discover patterns without re-scanning the database. In this paper, we propose an algorithm called Minits-AllOcc (MINIng Timed Sequential Pattern for All-time Occurrences) to find sequential patterns and the transition time between itemsets based on all occurrences of a pattern in the database. We also propose a parallel multi-core CPU version of this algorithm, called MMinits-AllOcc (Multi-core for MINIng Timed Sequential Pattern for All-time Occurrences), to deal with Big Data. Extensive experiments on real and synthetic datasets show the advantages of this approach over the brute-force method. Also, the multi-core CPU version of the algorithm is shown to outperform the single-core version on Big Data by 2.5X.

Keywords. Data mining, Sequential pattern mining, Timed sequential patterns, Singe-core and multi-core processor.

[1] Corresponding Author: Somayah Karsoum , E-mail: somayah.karsoum@ou.edu

1. Introduction

Sequential pattern mining (SPM) [1] analyzes a sequence database, which contains sequences of events that are ordered based on the times when the events occurred or collected, called timestamps, to discover sequential patterns. These sequential patterns are those time-ordered events that frequently occur in the sequence database. An example of a sequential pattern is “Patients of heart attack have cholesterol first, then uncomfortable pressure, then abdominal pain.” SPM has been used in many real-life application areas such as weather prediction [3] and [23], illness symptom pattern prediction [16], network intrusion detection [26], educational data mining [5], and customer shopping behaviors [1]. For example, in healthcare applications, with sequential patterns discovered from a sequence database containing illness symptom occurrence, we can answer questions like “in which order do the symptoms of a heart attack frequently occur?” Similarly, in weather prediction, with sequential patterns discovered from a sequence database recording past tornado events, we can answer questions like “what is the order of the cities that are hit by a tornado frequently?” An example of this is that in the state of Oklahoma in the U.S.A., during the tornado season, tornadoes tend to hit three cities, Oklahoma City, Moore, and Norman, in that order. However, the existing works in SPM, such as [13],[25], and [34], tried to improve the efficiency of techniques to discover the frequent sequential patterns but discard the time dimension completely. The timestamps are used to order events within a sequential pattern, but the transition time between these events is not shown in the discovered sequential patterns. In many applications, it is important to know the time interval [min, max] events in a frequent sequential pattern discovered, which we call a timed sequential pattern. For example, knowing when the following symptom of a heart attack will occur helps healthcare providers in forming diagnoses, providing treatments at the right time, and intervening early in critical cases. Similarly, we may want to have a frequent timed sequential pattern that shows that *after a tornado hits Oklahoma City, within 10 to 15 minutes later, the tornado will hit Moore, and then within 3 to 5 minutes later, the tornado will hit Norman.* Knowing the temporal information (the time intervals of event occurrences) in frequent sequential patterns will help preparing a safety plan to reduce damages and loss.



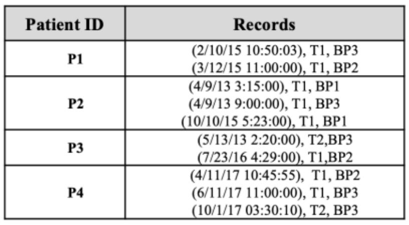
Patient ID	Timestamp	Temperature	Temperature Class (T)	Blood Pressure	Blood Pressure Class (BP)
P1	2/10/15 10:50:03	37.0	1	131	3
P2	4/9/13 3:15:00	36.0	1	112	1
P1	3/12/15 11:00:00	37.5	1	128	2
P2	4/9/13 9:00:00	37.3	1	134	3
P2	10/10/15 5:23:00	38.0	1	110	1
P3	5/13/13 2:20:00	37.1	2	135	3
P4	4/11/17 10:45:55	36.9	1	130	2
P3	7/23/16 4:29:00	41.3	1	123	2
P4	6/11/17 11:00:00	38.1	1	139	3
P4	10/1/17 03:30:10	38.3	2	139	3

Fig. 1. Patients' historic health information and discretize data

As shown in Fig. 1, we have the historical health data consisting of the temperature (T) and blood pressure (BP) of four patients (P) who had a heart attack. The time was recorded every time the measurement of temperature or blood pressure was taken for a patient as shown in the second column. Since a sequential pattern mining algorithm does not deal with continuous data, we need to apply a discretization technique to segment the data into classes that have similar features to fall within the same group. For instance, the blood pressure (BP) has five levels [4]: (1) Normal ($BP < 120$), (2) Elevated ($120 \leq BP \leq 129$), (3) High Stage 1 ($130 \leq BP \leq 139$), (4) High Stage 2 ($140 \leq BP \leq 180$), and (5) Crisis ($BP > 181$). Therefore, we added a column next to each measurement that contains the equivalent class ID and refers to the blood pressure with the abbreviation BP followed by the ID of the class into which the blood pressure falls. Back to the blood pressure levels, we can see that the last column has the blood pressure Class 3 in the first tuple because the value 131 belongs to Class 3 (High Stage 1). We grouped the tuples in Fig. 1 by Patient ID as shown in Fig. 2, which represents the timed sequence Database. The first tuple displays all the symptoms of patient P1 ordered based on the timestamps.

A timed sequential pattern that we want to discover is about the symptoms that frequently occur among patients and the typical transition times between the symptoms (in terms of days in our example). The following is the format of a pattern called a Timed Sequential Pattern (TSP) that would be discovered in this study:

$$TSP = < \{T1, BP3\}\ [2,7]\ \{T2\}>$$



Patient ID	Records
P1	(2/10/15 10:50:03), T1, BP3 (3/12/15 11:00:00), T1, BP2
P2	(4/9/13 3:15:00), T1, BP1 (4/9/13 9:00:00), T1, BP3 (10/10/15 5:23:00), T1, BP1
P3	(5/13/13 2:20:00), T2,BP3 (7/23/16 4:29:00), T1,BP2
P4	(4/11/17 10:45:55), T1, BP2 (6/11/17 11:00:00), T1, BP3 (10/1/17 03:30:10), T2, BP3

Fig. 2. Sequence records

This TSP has two itemsets: itemset 1 consisting of two items T1 and BP3, and itemset 2 consisting of item T2. Itemset 2 occurs within 2 to 7 days after itemset 1. In our notations, all items enclosed within braces {} occur at the same time and constitute an itemset, and the square brackets [min, max] indicate the time duration to move from one itemset to the next. In this algorithm, the time duration represents the temporal relation [min, max]. Thus, the given example TSP shows that frequently when patients have a temperature falling in the Class 1 (T1) and a blood pressure falling in the Class 3 (BP3), then within 2 to 7 days, the patients will have a temperature in the Class 2 (T2). If we apply traditional sequential pattern mining, then this sequential pattern will only be < {T1, BP3} {T2}>, which does not include the transition time [2, 7].

Incorporating the temporal information in a sequential pattern raises additional challenges for mining compared to regular sequential pattern mining. First, while both sequential pattern mining and timed sequential pattern mining need to find out whether a pattern occurs in some tuples of a database, timed sequential pattern mining also needs to find out how many times the pattern occurs in each tuple to compute the temporal relationship between the item3sets in the pattern. Suppose we have a tuple of a patient that has all the measurements within six months and the following symptoms occurring many times: low temperature followed by high blood pressure after some time. Since the timed sequential pattern mining problem wants to know when the high blood pressure occurs, it is not sufficient to find only the first position of this symptom and report the temporal relation. For example, from Fig. 2, we can observe that the fourth patient, P4, has the following symptoms based on the timestamp order: the temperature from Class 1 and the blood pressure from Class 2 {T1, BP2} followed by the temperature from Class 1 and the blood pressure from Class 3 {T1, BP3} followed by the temperature from Class 2 and the blood pressure from Class 3 {T2, BP3}. The tuple for this patient is: P4 = < {T1, BP2}, {T1, BP3}, {T2, BP3}>. To find the temporal relation between the two symptoms {T1} and {BP3} (for the pattern denoted as < {T1} [] {BP3}>), we need to do the following as shown in Fig. 3:

1- Find the timestamp difference t1 between the first occurrence of T1 and the first occurrence of BP3 (solid arrows).
2- Find the timestamp difference t2 between the first occurrence of T1 and the second occurrence of BP3 (dotted arrows).
3- Find the timestamp difference t3 between the second occurrence of T1 and the first occurrence of BP3 (dashed arrows).
4- Find the minimum timestamp difference and the maximum timestamp difference among t1, t2, and t3.
5- Produce the temporal relation as [min, max].

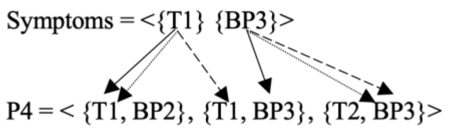


Fig. 3. All possible occurrences of the symptom {T1} {BP3} in P4

So, to find all possible occurrences of a pattern, the naïve method is to scan each tuple until the end in the database. However, a sequential pattern mining algorithm will stop checking the rest of a tuple in the database as soon as the pattern is found. In contrast, timed sequential pattern mining requires checking all the tuples in the database. First, it is necessary to consider all possible occurrences of the pattern and all the different timestamps of each occurrence and find the temporal relation. After the temporal relation is found for one patient, we need to check the temporal relation for the same symptoms among all patients. The final interval [min, max] represents the minimum and maximum time difference among all patients in the database.

This leads to the second challenge of timed sequential pattern mining, which updates the temporal relation between itemsets as soon as a pattern is found. When a timed sequential pattern is defined, it means that the ratio of tuples that contain this pattern is greater than or equal to a user-defined threshold. Then, when we want to extend that pattern to include more symptoms; it does not mean that the pattern must appear at the same tuples because some tuples may not carry it anymore. Accordingly, the time relation is not valid anymore, and we need to update that relationship based on the new timestamps of the new tuples. Let us suppose that we have the timed sequence pattern < {T1, BP3} [t1, t2] {T1}>. From Fig. 1, we can observe that P1 (Tuples 1 and 3) and P2 (Tuples 4 and 5) have these symptoms. So, t1 and t2 are calculated based on the timestamps associated with these symptoms in these tuples. When the pattern is extended to be < {T1, BP3} [t1, t2] {T1} [t3, t4] {T1, BP1}>, we can observe that the records of P2 do not carry this pattern and only P1 had these symptoms. Therefore, the t1 and t2 must be updated based on the timestamps associated with symptoms in Tuples 1 and 3 (not also Tuples 4 and 5). The brute force technique needs to scan the database again to update the temporal relation of the pattern. Thus, for every pattern, we need to scan the entire database many times to make sure that we have the correct temporal relations.

The contributions of this paper are the following:

1. The idea of incorporating transition time between item sets in a sequential pattern indicates all possible time occurrences of the pattern within the whole timed sequence database. The time can be any descriptive statistic based on the user's preference, such as range, average, etc.
2. The parallel implementation of the Minits-AllOcc algorithm can help when dealing with Big Data.
3. The extensive experiments compare the single-core algorithm against the multi-core algorithm on real and synthetic datasets.

The remainder of the paper is organized as follows: Section 2 reviews the related work. Section 3 introduces and defines the timed sequential pattern mining problem. Section 4 explains how the algorithm works. The results of performance evaluations on different datasets are given in Section 5. Finally, Section 6 concludes the paper and discusses future work.

2. Problem Definition

In this section, we review the definitions of the sequential pattern mining problem and introduce new definitions for the timed sequential pattern mining problem. Recalling the traditional sequential pattern mining problem [1], we define an **itemset** I as a set of **items,** such that I ⊆ X, where X = {x_1, x_2, . . . x_l} is a set of items in the database. A **sequence** (tuple) *s* is an ordered list (based on timestamps) of item sets. A sequence A = <{a_1}, {a_2}, …{a_n}> is **contained in** another sequence B = <{b_1}, {b_2}, …{b_m}> and B is a **super-sequence** of A if there exists a set of integers, $1\leq j_1 < j_2 <\ldots< j_n \leq m$, such that$a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{jn}$.

A **sequence database** S is a set of sequences (tuples) <sid, s_i> where sid is a sequence identifier and s_i is a sequence. A tuple <sid, s_i> is said to contain a sequence α if α is a sub-sequence of s_i. Since our problem also considers the temporal data, we incorporate timestamps explicitly in the database and introduce new definitions.

Definition 1. A timed ***event*** is a pair *e* = (*I*, *t*), where *I* am an item set that occurs at the timestamp *t.* We use e. I and e.t to indicate, respectively, the itemset I and the timestamp t associated with the event e. The list of events that is sorted in the timestamp order is called a ***timed sequence*** *TS* = <{e_1}, {e_2}, ... , {e_n}>, such that $e_i.x \subseteq I$ ($1 \leq i \leq n$). A ***timed sequence database TSDB*** is a set of sequences <*TS_id*, *TS*> where *TS_id* is a timed-sequence identifier and *TS* is a timed sequence.

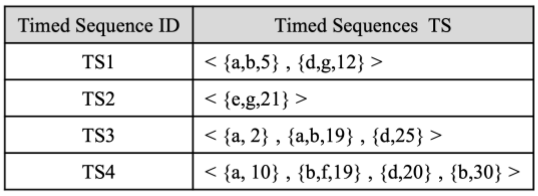
Example 1. (**Running Example**) The timed sequence database in Fig. 4 is used as an illustrative example in this paper. For simplicity, we will use letters to refer to items that represent different properties of objects in the database (e.g., temperature and blood pressure for patients), and integer numbers to refer to timestamps that represent the times when those properties are collected. In this example, there are four timed sequences with IDs from TS1 to TS4. Each timed sequence consists of a set of events ordered in the events' timestamps. For example, TS1 consists of two events: the first event {a, b, 5}, which occurred at timestamp 5, followed by the second event {d, g, 12}, which occurred at timestamp 12.

Definition 2. Given a sequence A = <{I_1}, {I_2}, …{I_n}> and a timed sequence TS = <{e_1}, {e_2}, … , {e_m}>, the ***All-time Occurrences*** of A in TS in the timed sequence database TSDB is defined as an ordered list of indices $1 \leq j_1 < j_2 < \ldots < j_n \leq m$, such that: $I_1 \subseteq e_{j_1}.I, I_2 \subseteq e_{j_2}.I, \ldots I_n \subseteq e_{j_n}.I$. The ***delta*** Δ is defined as$\Delta = e_{p.j_{i-1}}.t - e_{p.j_i}.t$.

Example 2. Let sequence A = <{a}{b}> and timed sequence TS4 = < {a, 10}, {b, f, 19}, {d, 20}, {b, 30}>, as shown in Fig. 4. The indices of the events for the first occurrence of sequence A in TS4 are {e_1, e_2}, as shown by the solid arrow in Fig. 5. The delta Δ is the difference between the timestamps of these two consecutive events, which is $e_{1.}t_1 = 10$ and $e_{2.}t_2 = 19$. Thus, the $\Delta = 19 - 10 = 9$. Then, the second occurrence of sequence A in TS4, as shown by the dotted arrow in Fig. 2, has the events' indices {e_1, e_4}. The delta Δ is the difference between the timestamps of these two consecutive events, which is $e_{1.}t_1 = 10$ and $e_{4.}t_2 = 30$. Thus, the $\Delta = 30 - 10 = 20$. Similarly, we can find the rest of the All-time Occurrence. The **support** of a sequence A in a sequence database, or a timed sequence database, is the percentage of the number of sequences in the database that contains A, such that sup(A) = (#sequences that contain A / #sequences in DB) *100. If the support of sequence A is greater than or equal to a user-defined threshold called minimum support (min_sup), then it is called a **sequential pattern** [1].

Definition 3. A sequence A is called a **timed sequential pattern TSP** if and only if it is a sequential pattern and accompanied by **temporal relationships** τ_i between item sets where it represents any descriptive statistic, such as an average of transition time or range, calculated based on the values of the delta Δ. TSP is denoted as: TSP = <{I_0} [τ_1] {I_1} [τ_2] {I_2}…… [τ_n] {I_n}>. For brevity, in the rest of this paper, when we mention a pattern, we refer to a timed sequential pattern.

Example 3. Let us assume the min-sup =50%; since the support of sequence A= <{a}{b}> is 50%, the sequence is a sequential pattern. In this paper, we assume that a user chooses the temporal relation to be presented as a range of time [min, max]. Thus, the timed sequential pattern version is <{a} [9, 20] {b}>. The timed sequential patterns thus are sequential patterns that satisfy the min_sup condition and include the transition times between item sets.



Timed Sequence ID	Timed Sequences TS
TS1	< {a,b,5} , {d,g,12} >
TS2	< {e,g,21} >
TS3	< {a, 2} , {a,b,19} , {d,25} >
TS4	< {a, 10} , {b,f,19} , {d,20} , {b,30} >

Fig. 4. An Example of Timed Sequence Database

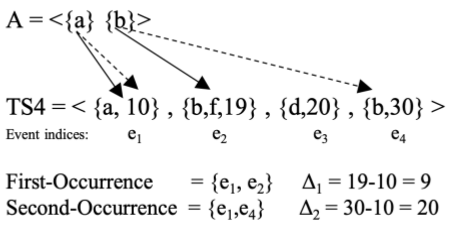

A = <{a} {b}>
TS4 = < {a, 10} , {b,f,19} , {d,20} , {b,30} >
Event indices: e1 e2 e3 e4
First-Occurrence = {e1, e2} Δ1 = 19-10 = 9
Second-Occurrence = {e1,e4} Δ2 = 30-10 = 20

Fig. 5. All-time Occurrence of A in TS4

3. Related Works

Sequential pattern mining was first introduced in [1], where three algorithms, AprioriSome, DynamicSome, and AprioriAll, were proposed to discover sequential patterns. AprioriAll is the basis of many other efficient algorithms that have been proposed to improve its performance. Those algorithms inspired [27] to propose a technique to generate fewer candidates called GSP. Since all algorithms were based on the Apriori algorithm, they were classified as Apriori-based algorithms. Other algorithms, such as SPADE [34], adopted a vertical ID-list database format that reduces the number of database scans. In contrast, pattern-growth-based algorithms, such as FreeSpan [13] and PrefixSpan [25], use database projection, making them more efficient than other Apriori-based algorithms, mainly when they deal with an extensive database. These algorithms generate a smaller database for their next pass because the sequence database is projected into a set of smaller databases, and then sequential patterns in each of them are explored. Thus, they are more efficient. More literature reviews about the state-of-the-art sequential pattern mining algorithms can be found in [9].

Recently, with the existence of a large volume of data in many applications, several sequential pattern mining algorithms have been proposed to efficiently handle large databases consisting of vast amounts of sequences using different platforms. For example, [15] uses the multi-core processor architecture to implement pDBV-SPM to improve processing speed for mining sequential patterns. Ha-GSP [22] adopts the principles of GSP and implements them on the Hadoop platform for solving the limited computing capacity and inadequate performance with massive data of the traditional GSP. MR-PrefixSpan [31] uses 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-the-art parallel sequence mining algorithms are in [10].

Previous algorithms represent the traditional technique of sequential mining patterns; however, the researchers studied if the same dataset can be used to extract more informative patterns. Thus, more techniques have been proposed to find more exciting extensions of sequential patterns and here examples of those patterns but not to exclude others. One of these patterns is closed sequential patterns [35,36] that mine not *all* subsequences but only *closed* subsequences, which those containing no super sequences with the same support. This solved the efficiency problem of dealing with the tremendous number of frequent subsequences for using a very low minimum support threshold or mining very long sequences. Other interesting patterns are negative sequential patterns [37,38,39,40]. The idea of positive (traditional) sequential patterns focuses on finding the positive occurrences correlation between items in a dataset. In contrast, the negative occurrences correlation, or absence, between items is more interesting in some practical applications. High-utility sequential patterns [41, 42, 43] is another type of sequential patterns dealing with a quantity of an item in the dataset. The traditional problem of sequential patterns considers only if an item appears in an itemset of a sequence or not. The goal of High Utility Sequential patterns HUSP is discovering subsequences having a utility (importance) greater than or equal to a minimum utility threshold in a quantitative sequential database. Periodic sequential patterns [44,45,46] is also can be considered as extension of sequential pattern mining by taking duration as a set of partitioned sequences. So, periodic pattern mining is still finding patterns that regularly appear in the time-series database but recurring in specific time interval (e.g., every Friday). This research concentrates on the traditional (positive) sequential patterns type.

Objects have an ordinal correlation in a sequential pattern based on the timestamp precedence. We can obtain a sequence by sorting all these objects based on the order of their timestamps. Because finding frequent itemsets in the association rule mining tasks discards the ordering of items, some techniques such as [24] take advantage of sorting items based on the timestamp. They discover different patterns that represent the different orderings of the items. For example, the *general episode* is a sequence with objects A, B, and C where A must occur first, but B and C can occur in any order. However, the *serial episode* is a sequence with objects A, B, and C where A must occur first, then B, and then C. However, the time between itemsets is still discarded, and they use the time as a gap constraint between itemsets in an episode. So, an expiry constraint T_X is another input besides the sequence database and min_sup threshold. The T_x is an additional control with the support threshold, which specifies that the appearance of symbols in an episode occurs no further than T_X time units apart from each other. Some techniques were proposed to specify some timing constraints, such as the time gaps between adjacent item sets in sequential patterns. For example, [4] modifies the Apriori [1] and PrefixSpan [25] algorithms to discover the time-interval sequential patterns that satisfy the interval duration boundaries. The I-PrefixSpan algorithm in [4] has another input called a set of time-intervals TI, where each time-interval has a range. [14] extends that work and proposes two algorithms: MI-Apriori and MI-Prefix. The time intervals incorporated in the patterns reveal the time between all pairs of items in a pattern; these patterns are called multi-time-interval sequential patterns. A list of intervals (t_{i3}, t_{i2}, t_{i1}) before item d in a pattern like <a t_{i1}, b, (t_{i2}, t_{i1}), c, (t_{i3}, t_{i2}, t_{i1}), d> means the intervals between items a, b, and c and item d are t_{i3}, t_{i2} and t_{i1}, respectively. In educational data mining, a *ti*-pattern model [5] is built based on the I-PrefixSpan algorithm to consider the time between students' activities. So, again, the inputs of this model are a temporal sequence database and a set of time-intervals (I_s, I_{mn}, I_h, I_d, I_w, I_{mt}), which refer to seconds, minutes, hours, days, weeks, and months. For example, one-time intervals were I_h meaning the model will find the activities of students with a gap value between one hour and one day. After the model is applied to a group of students who enrolled in mathematics and computer science program in Learning Management System, one of the time-interval patterns (*ti*-pattern) was found: <Lab $_{\{1,3\}}$ I_h Lab $_{\{2,3\}}$ I_h Lab $_{\{2,3,4\}}$ >, where Lab $_{\{1,3\}}$ means either Lab$_1$ or Lab$_3$. The experts can observe that some students work sequentially on several exercise sheets from this pattern. Since the students spend this gab, it means that the students dig deep into their work. Also, [2] extracts the sequential patterns of diseases from a medical dataset within user-specified time intervals. CAI-PrefixSpan [19] is proposed to apply the confident condition from association rules besides the support condition to filter the timed sequential patterns. The advantage of this is that the decision-makers can be confident about the possibility of an event happening within a certain time interval.

The drawback of these methods is that their results will miss some frequent patterns that do not fulfil the time range constraint. To decide if a pattern is common, two conditions must be satisfied: the support of the pattern must be greater than or equal to the min_sup, and the time ranges between the item sets in the pattern must lie within the defined time intervals. Therefore, if a pattern fulfils the first condition, which means it is common but does not fulfil the second condition, the algorithm will not report it.

[11] incorporates the temporal dimension in the sequential pattern by defining temporally annotated sequences (TAS), and [12] proposes 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. The algorithm only works for one-dimensional data. Also, the times between events in a trajectory pattern are strict, which does not consider the variety of the traveling time spent between locations by using different transportation modes, for example. [33] relaxes the travel time so that it is a realistic range for traveling time. The algorithm still cannot deal with multidimensional data because it deals with only locations in trajectory data. Also, all the previous techniques do not consider all possible occurrences of a pattern in an individual sequence in a database, which means the temporal relations are calculated based on only the first occurrence of a pattern. The issue of calculating the time intervals of the first occurrence of a pattern and ignoring other occurrences is addressed in [21]. However, this approach is beneficial for only a few applications. For example, if a developer wants to evaluate the ease of use of a navigation system, the time of moving from A to B is tested when the users visit those locations for the first time. In contrast, in other applications, such as the healthcare application described in Section 1 above, we must consider all possible occurrences to provide accurate time intervals.

There also exist works that consider other issues related to time-interval sequential patterns. FARPAMp (Fast Robust Pattern Mining with information about prior uncertainty) [30] can deal with timestamp uncertainties. This issue may occur if two events A and B happen during a time interval that can be overlap. This leads to the possibility of event A appearing before event B or vice versa. So, the approach is focused on using time points instead of intervals and fitting probabilistic models for the errors in the timestamps around these time points. It is an interesting research issue; however, this is outside the scope of our research. We are addressing the issues of finding all possible occurrences of timed sequential patterns and producing the most updated temporal relations between itemsets in the discovered patterns.

To the best of our knowledge, there is no existing algorithm that can find the complete set of timed sequential patterns, in which each pattern includes the itemsets that occur in time order and the transition times between them.

4. The Proposed Algorithm: Minutes-AllOcc

We propose an algorithm called Minits-AllOcc to discover the complete set of timed sequential patterns, which is already frequent candidates, from a timed sequence database. We have the following subsections that describe the algorithm: Section 4.1 introduces the core data structure of the algorithm; Section 4.2 gives a brief overview before the details of the algorithm are explained step by step in Section 4.3; Section 4.4 analyzes the time complexity of the algorithm, and Section 4.5 proposes enhancements to improve the efficiency of the algorithm.

4.1. Occurrence Tree (o-Tree)

A data structure called *the occurrence tree* (O-tree) is proposed to represent all possible occurrences of a pattern in a particular timed sequence in TSDB. This tree is the essence of the algorithm because it helps generate timed sequence patterns without scanning the timed sequence database many times. In the tree, the timed sequence ID (TSID) is stored as the root. The rest of the nodes stores an event ID e_{ID} and its timestamp $e_{ID}.t$. A node can have multiple parent nodes and multiple child nodes. The information associated with the link between a parent node and a child node represents the difference Δ between the timestamps of the two nodes: parent and its child. The structure of the tree is shown in Fig. 6. For example, when the TS3 in Fig. 4 is scanned, three occurrence trees for item, *a*, *b*, and *d* are created from the timed sequences <{*a,2*},<{*a,b,19*},{*d,25*}>. Since the candidate sequence <{*a*}> appears twice in TS3, its O-tree in Fig. 7 has two nodes connected to the root. The first one represents the first occurrence at the first event e_1 with its timestamp, and the second represents the

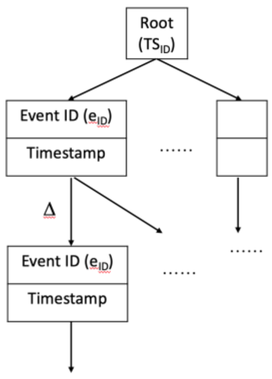


Fig. 6. Occurrence tree data structure

second occurrence at the second event e_2 with its timestamp. However, the sequence *<{a} {a}>* appears once in TS3 that has two nodes too, but one is connected to the root and the other is connected to the other node via a link Δ. The link holds the difference between the parent and child timestamps 19-2=17.

Since each sequence has an O-tree for each timed sequence in TSDB that contains it, the sequence will have a collection of O-trees that identify its occurrence in the whole TSDB. Thus, we give the following definition.

Definition 4. Given a sequence A and timed sequence database TSDB, ***A-Forest*** is a collection of all O-trees that identify all possible occurrences of sequence A in TSDB. Fig. 8. demonstrates the forest of four sequences, <{a}>, <{b}>, <{a} [9, 20] {b}>, and < {a, b}>. Each forest is surrounded by a dotted rectangle, which has a group of O-trees that indicates all time occurrences of a sequence in TSDB.

4.2 Overview

Given a TSDB and a min_sup threshold, the main goal of Minits-AllOcc is to find the complete set of the timed sequential patterns in the TSDB such that each pattern's support is greater than or equal to the min_sup threshold. To achieve this goal, Minits-AllOcc utilizes the forests to store all the required information from the TSDB and

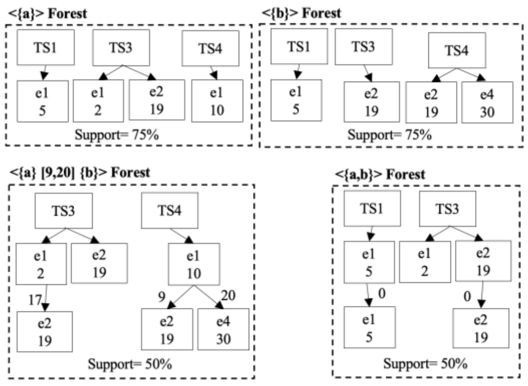


Fig. 8. Merging O-trees of <{a}> and <{b}> to generate < {a, b}>-*forest* and <{a}[9, 20]{b}>-*forest*

uses them to mine the patterns without having to scan the TSDB many times. The following steps are performed:

1) Scan TSDB to build an I_j-forest for each distinct item I_j.

2) Find frequent 1-items by counting the number of O-trees in each forest, compare it against the min_sup threshold, and remove the infrequent 1-items.

3) Merge all O-trees with the same TSID (root node) from different forests to build a new forest for a candidate sequence. It should be noted that there are two relations

between itemsets considered while merging the steps: *event-relation* and *sequence-relation,* which are defined as follows:

Definition 5. Given two itemset X and Y, it is said that X and Y have an ***Event-relation*** e-relation between them, denoted as < {X, Y}> if X and Y occur in the same event. For example, assume that we have the following timed sequential pattern = < {High temperature, High blood pressure} [2,3] {low temperature}>. It means that the patient has both high temperature and high blood pressure simultaneously, and after 2 to 3 days, the patient has a low temperature.

Definition 6. Given two itemset X and Y, it is said that X and Y have a ***Sequence-relation*** s-relation between them, denoted as <{X} {Y}> if X and Y occur in two different events and the event of X occurs before the event Y. For example, suppose that we have the following timed sequential pattern = < {High temperature} [4,6] {High blood pressure} [2,3] {low temperature}>. It means that the patient has only a high-temperature symptom. Then after 4 to 6 days, the patient has the high blood pressure symptom. Later, after 2 to 3 days, the patient has a low-temperature symptom.

4) Count the number of O-trees in each forest, compute the support, and compare it against the min_sup threshold to find the sequential patterns among candidate sequences. By performing step 4, Minits-AllOcc avoids scanning the whole TSDB for each candidate to calculate its support.

5) Compute the temporal relation of the suffix (the new appending part of the pattern) if the candidate sequence is frequent. Then, update the temporal relation of the prefix (the previous part of the pattern) and generate a timed sequential pattern.

6) Repeat Steps 3, 4, and 5 until the algorithm cannot identify any new timed sequential pattern. Minits-All Occ's pseudo-code is presented in Fig. 9.

4.3 The Details of the Minits-AllOcc Algorithm

This section describes the five steps presented in the previous section 4.2 in detail using the running example shown in Fig. 4. The algorithm scans the TSDB tuple by tuple and builds the associated forest for each item by adding the occurrence trees O-tree (lines 1-19). As shown in Fig. 7, for example, after the algorithm finishes scanning the TSDB, the <{a}>-the forest has three O-trees because the sequence <{a}> appears in three timed sequences: TS1, TS3, and TS4. Each O-tree captures all occurrences with their timestamps of an item and in a particular TS. Thus, in TS1, we have one node that shows the item *a* appears in the first event in TS1, and its timestamp is 5. To know the support of distinct items, the algorithm counts the number of O-trees in each forest and compares it against the min_sup threshold. If the support of a forest, which also represents of distinct item's support, is less than the threshold, the forest is removed (lines 20 -27). The two sequences <{e}> and <{f}> are not frequent because their forests have only one tree, which means they appear only in one TS; therefore, their support is 25%. Consequently, two sets are formatted: TSP and -TSP. The first set of TSP contains the complete, timed sequential patterns. It will be updated periodically as a new timed sequential pattern is discovered. The second set is 1-TSP, which contains only the timed sequential patterns of length 1, which will be used as a seed set to extend the patterns in further steps. Both sets TSP and 1-TSP have these values {<{a}, {b}, {d}, {g}>} (lines 24-25). The next step is generating candidates by merging the O-trees of all 1-timed sequential patterns by calling the function find-TSPs (lines 30). The mechanism of merging trees is as follows: if the relationship is an s-relation, the appended node must have an event ID e_i that is greater than the event ID in the parent node (i.e. comparing the event IDs in the two nodes) (line 57). Then, the link holds the difference between the timestamps of the parent and their child (line 59). In contrast, if the relationship is an e-relation, the appended node must have the same event ID e_i as its parent (line 53). For instance, the forests of the two candidates <{a}[ ]{b}>, which represents an s-relation, and <{a, b}>, which represents an e-relation, are shown in Fig. 8. The first <{a}[ ]{b}>-forest has two O-trees that are generated by combining the <{a}>-forest and the <{b}> -forest. Even though both the forests have an O-tree that has a root TS1, the O-tree of <{b}> does not contain a node that has an event ID greater than e_1; thus, it is removed from the <{a}[ ]{b}>-forest. In contrast, the node e_2 from the <{b}>-the forest is attached to the node e1 from the <{a}>-forest and the Δ is calculated

between those nodes, which is 19-2 =17. However, the node that has e_2 from the <{a}>-forest does not connect to any node. Since the algorithm is looking for all possible occurrences of sequence <{a}[ ]{b}>, the node e_1 in TS4 is connected to the two nodes, which have the event IDs e_2 and e_4, from the <{b}> O-tree, and each link between the parent node e_1 and child node e_2 and child node e_4 carries the difference between the timestamps of the two connected nodes. Because in this example, we consider a temporal relation as a range [min, max], the algorithm chooses the minimum and maximum values among all the O-trees in the <{a}[ ]{b}>-forest, which is [9, 20]. The second <{a, b}>-the forest has two O-trees that are generated by combining the <{a}>-forest and the <{b}> -forest. The difference between the s-relation case and the e-relation case when we merge the trees is the condition of appending nodes. Since this is an e-relation, all added nodes must have the same event ID e_i as their parents. Also, the Δ is always 0 because the nodes have the same timestamps. Both patterns <{a} [9, 20] {b}> and <{a, b}> are considered to be timed sequential patterns and they are added to TSP set because their supports are 50% (line 69-76). We calculated the support using the below formula:

$$Support(candidatesequence) = \frac{O_{trees} \in theforest}{timedsequences \in TSDB * 100}$$

These two timed sequential patterns are added into TSP = {<{a}>,<{b}>,<{d}>,<{g}>,<{a} [9, 20] {b}>, <{a, b}>} The algorithm repeats the same steps, by calling function find-TSPs recursively in line 37, to extend the pattern by merging O-trees, generating candidates, finding TSPs, and computing temporal relations until no more TSPs can be found.

As shown in Fig. 10, pattern <{a} [9, 17] {b} [1, 6] {d}> result from merging between <{a} [9, 20] {b}>-forest and <{d}>-forest. The forest consists of only the O-trees that representing the candidate, then the support is calculated. Since the support is 50%, the time between the prefix <{a} [ ] {b}> and suffix <{d}> is calculated as defined before (the range [min, max]). The TSP set is updated to be {<{a}>, <{b}>, <{d}>, <{g}>, <{a} [9, 20] {b}>, < {a, b}>, <{a} [9, 17] {b} [1, 6] {d}>}. As it is noted, the temporal relation between item sets {a} and {b} in the two patterns <{a} [9, 20] {b}> and <{a} [9, 17] {b} [ ]{d}> changed.

Minits-AllOcc continues repeating the steps until the complete set of TSPs is discovered. The reader can verify that the TSPs in this example is = {<{a}>,<{b}>,<{d}>,<{g}>,<{a} [9, 20] {b}>, <{a} [6, 23] {d}>, <{b} [1, 7] {d}>, <{a, b} [6, 7] {d}>, <{a} [9, 17] {b} [1, 6] {d }>}.

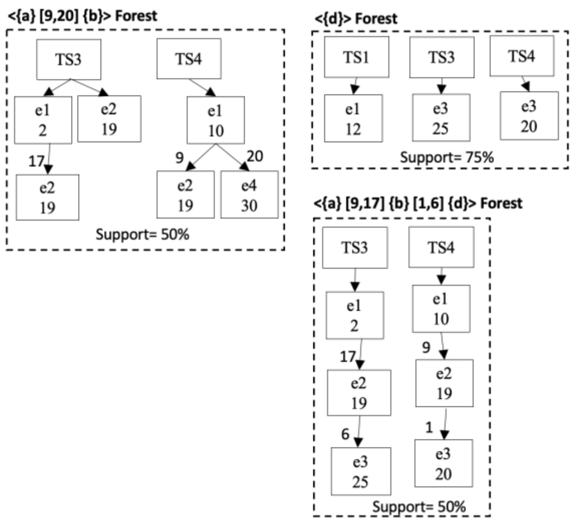


Fig. 10. Merging <{a}[9, 20]{b}>-*forest* and <{d}> to Generate <{a}[9, 17]{b}[1, 6] {d}>-*forest*

4.4 Analysis of Minits-AllOcc

In this subsection, we discuss the worst-case time complexity of the Minits-AllOcc algorithm. We have:

In this subsection, we discuss the worst-case time complexity of the Minits-AllOcc algorithm. We have:

- *S(m),* where | *S* | = the number of timed sequences TS in TSDB.
- *E(r),* where |*E*| = the maximum number of events in a timed sequence.
- *I(c),* where |*I*| = the maximum number of items in an event.
- *G(s),* where | *G* |= the number of singleton items in TSDB.
- *N* , where | N| = the number of all possible candidates

We start with the first part of the algorithm that needs to check each Timed Sequence *S* in TSDB, each event *E* inside that *S*, and each item inside that *E* to build the forest (line 1 -19), which cost O(*S*E*I*). If it is the first time to read an item, that means its forest does not exist. So, we need to build it from scratch and start counting the number of O-trees inside that forest. Otherwise, we just need to update the forest by adding the new O-tree into an existing forest and update the number of O-tress inside that forest, which cost O(log N). Therefore, the total amount of work performed by the end of (line 19) is *O (S *E*I*log N).*

To keep only frequent candidate sequences and remove infrequent ones, the algorithm calculates the support for each forest (line20 - 27) and adds the frequent candidate sequences into the TSP-set. So, the total amount of work performed by the end of (line 27) is *O(G+log I)* because the number of forests is equal to the number of singleton (distinct) items in TSDB and removing O-trees for any infrequent sequence is log I .

After that, the algorithm extends the patterns to generate more candidates by calling the function *Find_TSP()* (line 30). The function tries to combine each item in the 1-TSP set to generate 2-length candidate sequences, for example at the first call. The prefix is the previous k-1-timed sequential patterns, and the suffix is an item from the 1-TSP set. The function will append the suffix to the prefix and check the support of the new candidate sequence to decide if it can be considered as a timed sequential pattern or not. First, we need to find the time complexity of internal functions, then, we will compute the time complexity of the whole *Find_TSP()* function.

The *Find_TSP()* function calls another function called *Merge_Trees ( )* (line 43), sends the forest of the prefix (previous timed sequential pattern), and the 1-TSP to build the forest for each new candidate sequence considering the different types of relationships, either it is an s-relation or e-relation. In line 47, the function picks an occurrence tree *pt* from the forest of the prefix and compares it with all occurrence trees *st* for each 1-length timed sequential pattern (line 48), which cost *S*G*. If two occurrence trees with the same root *TSID* have been found, the function checks if the *event ID* of each leaf node from both trees is the same ID or the *event ID* in the suffix node is greater than the ID in the prefix node (line 53 and 57), which needs to check all events in both tress E^2. Also, this function updates the content of the forest by adding appropriate O-trees and calculating the differences Δ between nodes if the relation type is s-relation. After that, in line 69, each candidate's support is calculated by counting the number of trees in its forest. If the candidate is frequent, then after each itemset, the temporal relation is inserted, which cost E. The total amount of work done by *Merge_Trees ( )* is $O(\log N*S*G*E^3)$.

Recursively, for each frequent 1-item in the suffix list, in which the time complexity is $O(G)$, the function *Find_TSP( )* is called (line 37) until no more candidates can be generated. In the worst case, the function is re-called until the length of a candidate is equal to the length of the longest timed sequence in *TSDB*, so it is $O(E)$. Thus, the total amount of work done by *Find_TSP( )*, ended by line 42, including the work done by nested function *Merge_Trees( )*, is $O(\log N*S*G^2*E^3)$.. Because we are considering all possible combinations between any k-1 sequential patterns, where k >=2, and 1-sequential patterns, the algorithm returns to line 30 and tries another combination between two items in the 1-TSP set. Thus, besides the time complexity of calling *Find_TSP( )*, the algorithm combines all items, so $O(G)$. The total amount of work done by the end of (line 32) is $O(\log N*S*G^3*E^3)$.

The work done by this algorithm for each subsection is $O(S*E*I*\log N) + O(G+\log I) + O(\log N*S*G^3*E^3)$. We conclude that the overall worst-case time complexity of this algorithm is $O(\log N*S*G^3*E^3)$.

4.5 The proposed Enhancement

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

4.5.1 Pruning the forest

This technique defines a sequence's forest after merging the O-trees. So, when those O-trees are used in the next step for generating candidates, they carry only the necessary information and, therefore, save space by removing some nodes and save time by avoiding traversing needless branches in trees. Any branch in an O-tree that does not have a new appended node will be removed after the merging step is executed. Fig. 11 represents the idea by marking the deleted branch of O-trees with a cross symbol. For example, the O-tree that has a TS3 root that results from merging TS3 O-tree from <{a}> and <{b}>-forests. Since there is no appended node to the right branch of <{a}>-forest, this node is removed from <{a} [9, 20] {b}>-forest. Those branches do not exist anymore in the O-trees.

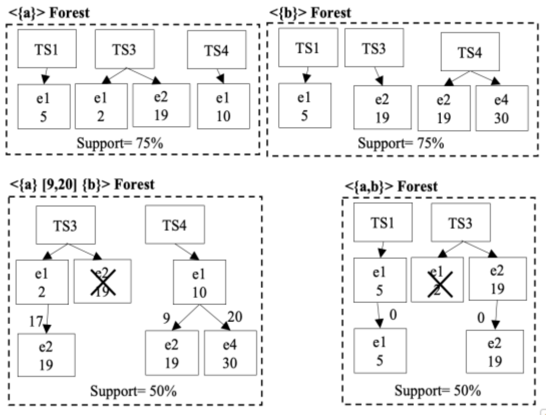


Fig. 11. Pruning the Original <{a} [9,20] {b}>-*forest* and < {a, b} >-*forest* in Fig. 10

4.5.2 *Using frequency matrix*

With this technique, we avoid generating unnecessary candidates, thereby reducing the number of forests. For example, the algorithm uses the 1-sequence-forests to generate 2-sequence candidates, then keeps frequent candidates and removes infrequent ones. Since all required information is already available in the forest, we build a frequency matrix for each sequence to indicate the frequent candidates. For example, the frequency matrix of <{a}> pattern is shown in Fig. 12. The two different relations, events, and sequences (the rows) and all 1-timed sequential patterns that can be combined with {a} (the columns) are considered. The cells under <{b}> column represent the frequency of the two relations between <{a}> and <{b}>. This frequency is calculated from the forests of those patterns, as shown in Fig.7. For an s-relation, there are two O-trees (TS3 and TS4) in which the <{a}> and <{b}> occur at different timestamps within the same timed sequence. For e-relation, there are two O-trees (TS1 and TS3) in which the <{a}> and <{b}> occur at the same timestamps within the same timed sequence. From the matrix, we can infer that <{g}> is not frequent either with an s-relation or e-relation; thus, we do not need to build the forest of sequence <{a}[ ]{g}> or <{a, g}>.

<{a}>	a	b	d	g
s-relation	25%	50%	75%	25%
e-relation	0%	50%	0%	0%

Fig. 12. Frequency Matrix for <{a}>

4.5.3 Using multi-core CPUs

Another enhancement is using multi-core CPUs for implementing Minits-AllOcc, which we call MMinits-AllOcc. The independent jobs that can be done at the same time are finding all possible candidates, merging O-trees for those candidates, and deciding if they are frequent or not. A queue holds all jobs. As soon as one thread becomes idle, the next job in the queue is assigned to it and this reduces the execution time of the algorithm. For instance, in the beginning, the algorithm scans the TSDB to build the forest for each item and finds that <{a}>, and <{b}> are frequent. In the serial version, the algorithm starts with the pattern <{a}> and keeps extending it until no more patterns can be found that have prefix <{a}>. Then, it starts with the pattern <{b}> and does the same thing. With the multi-core version, the algorithm inserts patterns <{a}>, <{b}> into the queue, as shown in Fig. 13, and works on generating their candidates at the same time. Then, the candidates, <{a} [ ]{a}>, <{a}[ ]{b}>,….etc., will be inserted into the queue to let any idle threads work on calculating their supports and report any of them as a time-sequential pattern. If one of these threads is done, then the pattern is extended by finding other candidates, <{a}[ ]{a}[ ]{a}>, <{a} [ ]{b}[ ] {b}, etc., and then inserting them into the queue. Those candidates wait to be assigned to an idle thread again. This process is kept going until no more jobs remain in the queue.

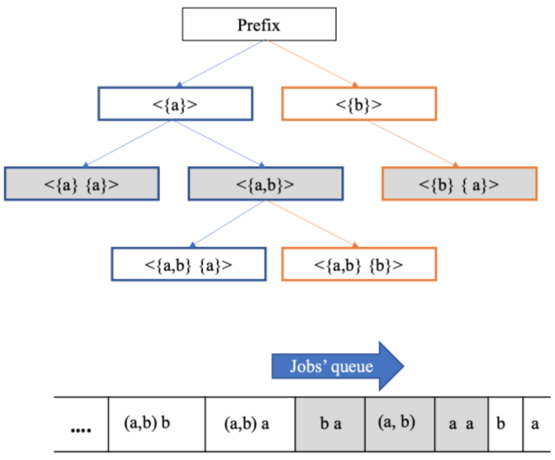


Fig. 13. Multi-core implementation

5. Performance Analysis

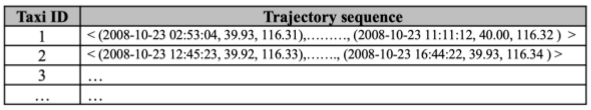
In this section, we describe the environment of experiments and report the evaluation results of testing the algorithms that are implemented in single-core CPUs (Minits-AllOcc) and multi-core CPUs (MMinits-AllOcc). Different parameters are considered when these experiments are conducted on real and synthetic datasets. After running many experiments, we have found that MMinits-AllOcc on a multi-core performed Minits-AllOcc on a single-core.

5.1. Experiment Setup

All experiments were performed on a computer with a 2.10 GHz Intel Xeon(R) processor with 64 gigabytes of RAM, running Ubuntu 18.04.1 LTS CPU with 12 cores. The Minits-AllOcc and MMinits-AllOcc algorithms are implemented in Java 1.8.

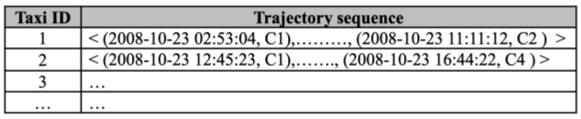
5.2 Datasets and Experimental Parameters

We use two real-life, T-Drive [17] [18] and Oklahoma Mesonet [3], [23], and synthetic datasets. The first real dataset 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 day. The dataset contains the following attributes: User ID, timestamp, latitude, and longitude, as shown in Fig. 14. 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, and 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) [7], and the results of the discretized sequences are shown in Fig. 15. Taxi 1 has a sequence that contains events in terms of clusters ID. For instance, event (2008-10-23 02:53:04, C1) refers to timestamp, and cluster Id, respectively. DBSCAN generates several 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 [20]



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	…
…	…

Fig. 14: Sequential database for the T-Drive dataset before discretization by DBSCAN

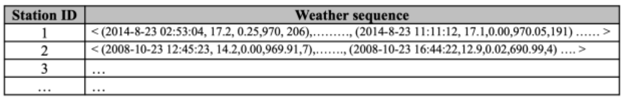


Taxi ID	Trajectory sequence
1	< (2008-10-23 02:53:04, C1),………, (2008-10-23 11:11:12, C2) >
2	< (2008-10-23 12:45:23, C1),……., (2008-10-23 16:44:22, C4) >
3	…
…	…

Fig. 15: Sequential database for the T-Drive dataset after discretization by DBSCAN

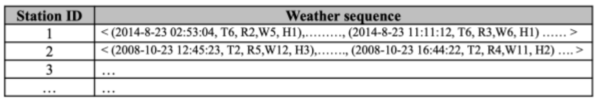
The second real data set from Oklahoma Mesonet is a world-class network of environmental interventions by a group of scientists from the University of Oklahoma (UO) and Oklahoma State University (OSU) for weather monitoring stations. This network was established on January 1, 1994 and consists of 120 stations covering each of Oklahoma's 77 counties. The measurements are packaged into "observations" every 5 minutes, then the observations are transmitted to a central facility every 5 minutes, 24 hours per day year-round. The dataset contains the following attributes: county ID, timestamp, air temperature, rainfall, wind, and moisture-humidity, as shown in Fig. 16. We discretized the data first by using well-known scales in Meteorology.

For air temperature, the index heat [32] is used to have the nine categories based on the temperature degree intervals in Fahrenheit: T1 (Extremely hot) [>54],T2 (Very hot) [53, 46], T3 (Hot) [46, 39], T4 (Very warm) [38, 32], T5 (Warm) [31, 26], T6 (Cold) [25, 0], T7 (Very cold) [0, -10], T8 (bitter cold) [- 11, -29], and 9 (Extreme cold) [> -30]. The recurrence interval [28] is used to categorize the rainfall based on the probability that the given event will be matched or exceeded in any given year. For example, there is a 1 in 50 chance that 6.60 inches of rain will fall in X County in a 24-hour period during any given year. The classes are: R1 (1 year) [1.16– 1.36], R2(2 years) [1.37-1.69], R3(5 years) [1.70-1.98], R4(10 years) [1.99-2.36], R5(25 years) [2.37–2.64], R6(50 years) [2.65–2.90], and R7(100 years) [2.90-3.15]. For wind, the Beaufort scale [29] defines 12 classes based on the speed of wind as: W0 (Calm) [<0.3], W1 (Light Air [0.3–1.5],W2 (light Breeze) [1.6–3.3], W3 (Gentle Breeze) [3.4- 5.5], W4 (Moderate Breeze) [5.5-7.9], W5 (Fresh Breeze) [8.0–10.7], W6 (Strong Breeze) [10.8–13.8], W7 (Near Gale) [13.9–17.1], W8 (Gale) [17.2–20.7], W9 (Strong Gale) [20.8–24.4], W10 (Storm) [28.4], W11 (Violent storm) [28.5–32.6], and W12 (Hurricane) [>= 32.7]. The last attribute, humidity (moisture), has 3 categories based on the "dew point" temperature [6]: H1 (Uncomfortably dry) [0 – 20], H2 (Comfortable) [20 -60], and H3 (Uncomfortably wet) [60-100]. The results of the discretized sequences are shown in Fig. 17.



Station ID	Weather sequence
1	< (2014-8-23 02:53:04, 17.2, 0.25,970, 206),………, (2014-8-23 11:11:12, 17.1,0.00,970.05,191) …… >
2	< (2008-10-23 12:45:23, 14.2,0.00,969.91,7),……., (2008-10-23 16:44:22,12.9,0.02,690.99,4) …. >
3	…
…	…

Fig. 16: Sequential database for the Oklahoma Mesonet dataset before discretization



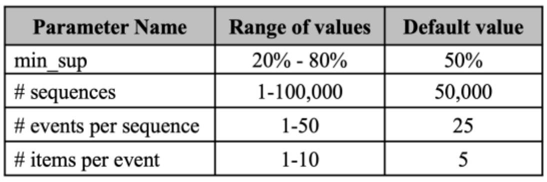
Station ID	Weather sequence
1	< (2014-8-23 02:53:04, T6, R2,W5, H1),………, (2014-8-23 11:11:12, T6, R3,W6, H1) …… >
2	< (2008-10-23 12:45:23, T2, R5,W12, H3),……., (2008-10-23 16:44:22, T2, R4,W11, H2) …. >
3	…
…	…

Fig. 17: Sequential database for the Oklahoma Mesonet dataset after discretization

The synthetic dataset was generated by using a tool provided by the SPMF Library [8]. Also, we set several parameters to conduct the experiments on the dataset. There are two types of parameters: static and dynamic parameters. The values of the static parameters are not changed in experiments. In contrast, the values of the dynamic parameters are changed from one experiment to another. In this experiment, we have four dynamic parameters. The first one is the minimum support threshold (min_sup). It is a user-defined threshold that applies to finding all timed sequential patterns in a timed sequence database TSDB. The second parameter is the number of timed sequences TS in TSDB (#Seq), which refers to the number of tuples in the database.

The third parameter is the length of TS in TSDB, which can also be represented as the number of events per TS (# Events). The last parameter is the number of items in each event (#items). It should be noted that the timestamp is a fixed attribute in all events. When it is said that the number of items per event is 3, for instance, it signifies three items plus the timestamp. We study the effects of all four parameters shown in Table 1 on the synthetic dataset. However, for the T-Drive dataset, the only valid dynamic parameter, that is shown in Table1 is the min-sup. Thus, all other three parameters are static. Now, we explain the range of the parameters and the default values of this analysis, as summarized in Table 1. When the experiment was conducted, we chose various values of one parameter within its range and assigned the default value to the other parameters. The min-sup parameter has a range of 20% to 80% with the default value = 50%, which is the median of the interval. The range of the number of timed sequences parameters is from 1 to 100,000, and its median value of 50,000, is the default value. For the number of events per sequence, the default value is 25 because the range is from 5 to 50. The number of items in the last parameter range has been set at 1 to 10 items per event; thus, the default value is 5, which is the median.

Table 1. Parameter List for the Synthetic Dataset



Parameter Name	Range of values	Default value
min_sup	20% - 80%	50%
# sequences	1-100,000	50,000
# events per sequence	1-50	25
# items per event	1-10	5

5.3Competing Algorithms

Since no existing algorithm can discover the timed sequential patterns and consider All-time Occurrences, we cannot compare Minits-AllOcc against any technique. We will compare it against MMinits-AllOcc.

5.4 Evaluation Metrics

The evaluation metrics include two measurements: (1) Execution Time (ET) of algorithms (Minits-AllOcc, and MMintis-AllOcc) and (2) Number of Patterns (#patterns) that are generated by these algorithms.

5.5 Experimental Results

In this section, we present the performance of the two algorithms, Minits-AllOcc and MMinits-AllOcc, in terms of execution time (ET) and the number of discovered patterns (#patterns) for the real and synthetic datasets.

5.5.1 Accuracy

To validate that Minits-AllOcc always gives the same sequential patterns in terms of the numbers and contents, excluding the temporal relation, PrefixSpan was used [25]. PrefixSpan was chosen because it is one of the well-known algorithms for discovering sequential patterns. It has been proven to produce complete and correct sequential patterns. First, all temporal relations were removed from the patterns that were generated by Minits-AllOcc. Next, these patterns were compared to the patterns that were generated by PrefixSpan to make sure that each sequential pattern generated by PrefixSpan has a matching one generated by Minits-AllOcc and MMinits-AllOcc. For example, a sequential pattern X= <{a} {b} {a, b}> was generated by PrefixSpan, and a timed sequential pattern Y= < {a} [2, 5] {b} [3, 7] {a, b}> was generated by Minits-AllOcc and MMinits-AllOcc. We took away the temporal relations from Y and compared them with pattern X. In case the order of at least one item set was different, the pattern X was not matching the pattern Y. For instance, Z= <{b} [2, 5] {a} [3, 7] {b, a}> was not matching pattern X because the item <{b}> occurred before <{a}>. However, within the last itemset {a, b} the order did not matter because all the items appeared at the same timestamp. At the end of this experiment, we found that the two algorithms—Minits-AllOcc and MMinits-AllOc—discovered the exact patterns that were produced by PrefixSpan. All algorithms produced the complete and correct set of sequential patterns.

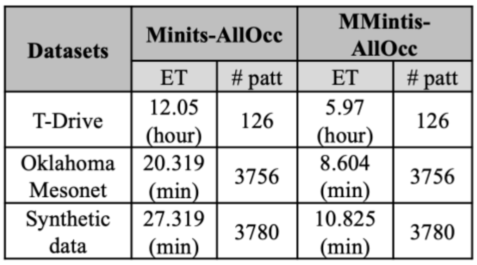
5.5.2 Execution Time

The execution time was recorded from the moment that a dataset had been read to the moment that an algorithm produced the timed sequential patterns. Table 2 shows the average performance of the two algorithms: Minits-AllOcc and MMinits-AllOcc. The execution time (ET) of MMinits-AllOcc decreases by 50% to 60% for T-Drive, Oklahoma Mesonet, and synthetic datasets, respectively, compared to the execution time of Minits-AllOcc.

5.5.3 Impact of Minimum Support

In this set of experiments, we compared execution time (ET) and the number of patterns (#patterns) for different values of

Table 2. Average Execution Time ET and #patterns



Datasets	Minits-AllOcc		MMintis-AllOcc	
	ET	# patt	ET	# patt
T-Drive	12.05 (hour)	126	5.97 (hour)	126
Oklahoma Mesonet	20.319 (min)	3756	8.604 (min)	3756
Synthetic data	27.319 (min)	3780	10.825 (min)	3780

minimum support threshold (min_sup) for datasets T-Drive, Oklahoma Mesonet, and synthetic. From Fig. 18 (a), Fig. 19 (a) and Fig. 20 (a), we can see that when the minimum support increased, the execution time of all algorithms decreased. This is because the algorithms generate fewer timed-sequential patterns when the min-sup is high, because of fewer candidate sequences that satisfy the min-sup condition. With a large amount of data and discovered timed sequential patterns, MMinits-AllOcc outperformed Minits-AllOcc, as shown in Fig. 18(a), Fig. 19(a) and Fig. 20(a). Therefore, multi-core CPUs ought to be used when the size of the timed sequence database is large.

The multi-core CPU version was also efficient when we had low min-sup. As shown in Fig. 19(a), and 20 (a), the ETs of both Minits-AllOcc and MMinits-AllOcc were very close when the min-sup was greater than 60%. This is because the number of candidate sequences, and thus the number of timed sequential patterns, was getting smaller, so most of the threads were idle. Therefore, MMinits-AllOcc did not need to use all the available threads and behaved almost like a single-core version Minits-AllOcc. Another observation was made based on the number of timed sequential patterns that were generated by these algorithms. All algorithms discovered the same number of patterns; thus, their curves were overlapping in Fig. 18(b), 19(b), 20(b), 20(d), 21(b), and 21(d). When the min-sup increased, the number of timed sequential patterns decreased because the patterns that satisfied the min-sup condition became fewer. By increasing the threshold min_sup, the percentage of timed sequences in the timed sequence database that was supposed to contain a candidate sequence decreased, as shown in Fig. 18(b), Fig. 19(b), and Fig. 20(b).

5.5.4 Impact of the Number of Sequences in the Database

In this set of experiments, we compared the execution time (ET) and the number of discovered timed sequential patterns (#patterns) according to the number of the timed sequences (#Seq). From Fig. 20(c), we can see that when the number of timed sequences increased, the execution times of all algorithms increased. This is because the algorithms needed more time to check the extra timed sequences that were added to the timed sequence database to decide if they contained a timed sequential pattern or not. We observed that the number of timed sequential patterns, which were generated by these algorithms, increased when the number of timed sequences increased, as shown in Fig. 20(d). The number of timed sequential patterns that were discovered by the algorithms also increased because the possibility of finding more patterns in the new timed sequences that satisfy the min-sup (50% as the default value) condition also increased. With an increased number of timed sequences in the database, the algorithms needed to check if some new patterns could occur and did not exist in the old timed sequences. Next, the algorithm checked their support against the threshold (min-sup). It is possible that the support of some old patterns in the database before new sequences was added did not satisfy the min-sup condition because they were not supported by enough timed sequences; but with a new timed sequence database, these patterns became timed-sequential patterns. Thus, the number of newly discovered timed sequential patterns would increase. For example, if a database had 1,000 sequences in the synthetic dataset, the number of timed sequential patterns was 3,720, while the number of timed sequential patterns was 3,780 when the timed sequence database had 10,000 timed sequences.

5.5.5 Impact of the Number of Events per Sequence

Fig. 21(a) and (b) show the impact of the number of events (#Events) per timed sequence on the execution time (ET) and the number of discovered sequential patterns (#patterns). There was a strong relationship between the length of a timed sequence and the number of discovered patterns. Increasing the length of timed sequences (#Events) drove the discovery of more patterns because the algorithm could extend a pattern up to the length of the timed sequence. If we have a timed sequence that contains *n* events, we can discover a set of timed sequential patterns such that their length varies from 1 to *n*. Subsequently, the required time of discovering those patterns will increase as shown in Fig. 21 (a).

5.5.6 Impact of the Number of Items per Event

In the last experiment, we increased the number of unique items in each event. That means many new items appear in the timed sequence database TSDB which leads to detecting new timed sequential patterns. When the number of items increases, the number of possible combinations between those items to generate candidates also increases. Thus, the number of patterns increased, as shown in Fig. 21(d). Growing the length of events led to the growth of the number of candidates, which means the algorithms needed more time, as shown in Fig. 21(c), to check those events, generate candidates, and determine if they were timed sequential patterns and reported the temporal relations.

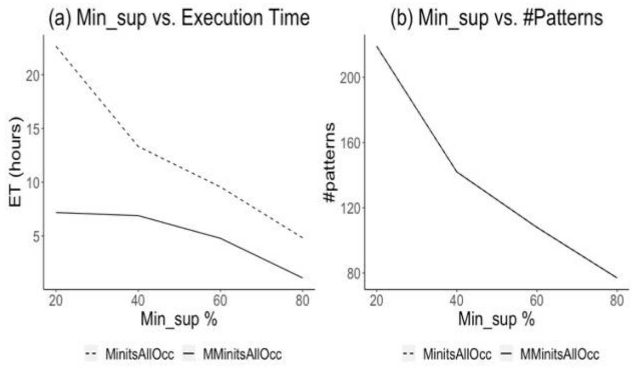


Fig. 18. Parameter Study for T-Drive Dataset

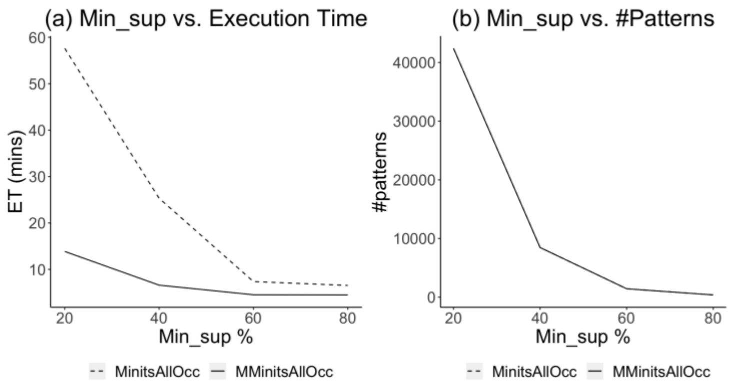


Fig. 19. Parameter Study for Oklahoma Dataset

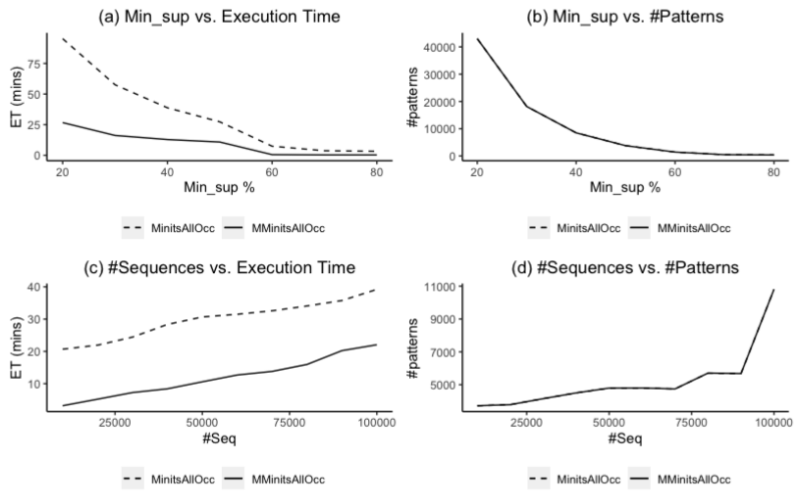


Fig. 20. Parameter Study for Synthetic Dataset

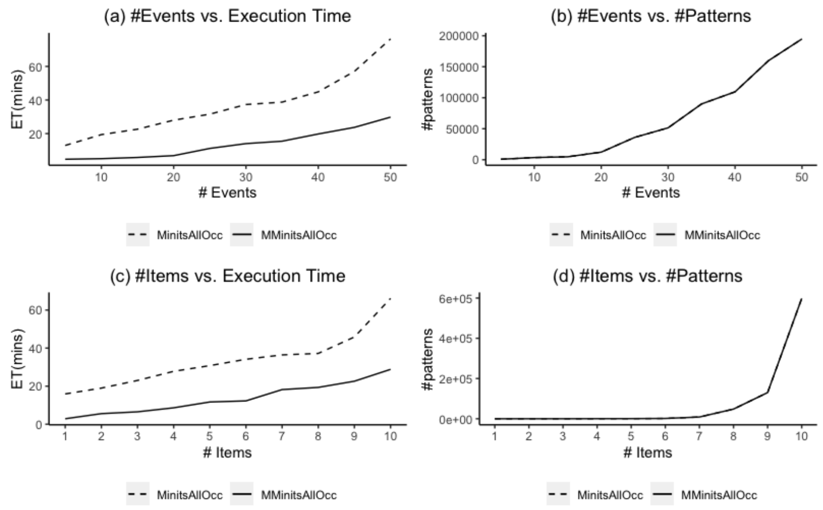


Fig. 21. Parameter Study for Synthetic Dataset

6. Conclusion and Future Work

In this paper, we presented an algorithm called Minits-AllOcc, to discover timed sequential patterns TSP, which are sequential patterns that include the transition times between all timesets. A temporal relation in the timed sequential patterns is calculated after considering all possible pattern occurrences across the timed sequence database

TSDB. We implemented two versions of Minits-AllOcc: (1) Minits-AllOcc using single-core CPUs, and (2) MMinits-AllOcc on multi-core CPUs. We conducted experiments to compare the accuracy and execution time of the algorithms. The experiments showed that the algorithms produced accurate patterns. Also, MMinits-AllOcc outperformed Minits-AllOcc when the dataset was enormous in size, in the length of timed sequences, or in the number of items per event. For future work, we plan to improve Minits-AllOcc to account for both long timed sequences and Dynamic Timed Sequence Database (DTSDB). The algorithm will be able to mine TSP without re-executing everything from scratch.

References

[1] Agrawal R, Srikant R. Mining sequential patterns. In Proceedings of the eleventh international conference on data engineering 1995 Mar 6 (pp. 3-14). IEEE.

[2] AlZahrani MY, Mazarbhuiya FA. Discovering constraint-based sequential patterns from medical datasets. Int. J. Recent Tech. Eng.(IJRTE). 2019 Nov.

[3] Brock FV, Crawford KC, Elliott RL, Cuperus GW, Stadler SJ, Johnson HL, Eilts MD. The Oklahoma Mesonet: a technical overview. Journal of Atmospheric and Oceanic Technology. 1995 Feb;12(1):5-19.

[4] Chen YL, Chiang MC, Ko MT. Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications. 2003 Oct 1;25(3):343-54.

[5] Dermy O, Brun A. Can We Take Advantage of Time-Interval Pattern Mining to Model Students Activity?. International Educational Data Mining Society. 2020 Jul.

[6] Dew Point vs Humidity [Internet]. (n.d.). Cited [2021 Oct 17] Retrieved from https://www.weather.gov/arx/why_dewpoint_vs_humidity

[7] Ester M, Kriegel HP, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. Inkdd 1996 Aug 2 (Vol. 96, No. 34, pp. 226-231).

[8] Fournier-Viger P, Lin JC, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam HT. The SPMF open-source data mining library version 2. InJoint European conference on machine learning and knowledge discovery in databases 2016 Sep 19 (pp. 36-40). Springer, Cham.

[9] Fournier-Viger P, Lin JC, Kiran RU, Koh YS, Thomas R. A survey of sequential pattern mining. Data Science and Pattern Recognition. 2017 Feb;1(1):54-77.

[10] Gan W, Lin JC, Fournier-Viger P, Chao HC, Yu PS. A survey of parallel sequential pattern mining. ACM Transactions on Knowledge Discovery from Data (TKDD). 2019 Jun 7;13(3):1-34.

[11] Giannotti F, Nanni M, Pedreschi D. Efficient mining of temporally annotated sequences. In Proceedings of the 2006 SIAM international conference on data mining 2006 Apr 20 (pp. 348-359). Society for Industrial and Applied Mathematics.

[12] Giannotti F, Nanni M, Pinelli F, Pedreschi D. Trajectory pattern mining. InProceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining 2007 Aug 12 (pp. 330-339).

[13] Han J, Pei J, Mortazavi-Asl B, Chen Q, Dayal U, Hsu MC. FreeSpan: frequent pattern-projected sequential pattern mining. InProceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining 2000 Aug 1 (pp. 355-359).

[14] Hu YH, Huang TC, Yang HR, Chen YL. On mining multi-time-interval sequential patterns. Data & Knowledge Engineering. 2009 Oct 1;68(10):1112-27.

[15] Huynh B, Vo B, Snasel V. An efficient method for mining frequent sequential patterns using multi-core processors. Applied Intelligence. 2017 Apr;46(3):703-16.

[16] Yuan J, Zheng Y, Zhang C, Xie W, Xie X, Sun G, Huang Y. T-drive: driving directions based on taxi trajectories. InProceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems 2010 Nov 2 (pp. 99-108).

[17] Yuan J, Zheng Y, Xie X, Sun G. Driving with knowledge from the physical world. InProceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining 2011 Aug 21 (pp. 316-324).

[18] Jou C, Shyur HJ, Yen CY. Timed sequential pattern mining based on confidence in accumulated intervals. InProceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014) 2014 Aug 13 (pp. 771-778). IEEE

[19] Karsoum S, Gruenwald L, Leal E. Impact of trajectory segmentation on discovering trajectory sequential patterns. In2018 IEEE International Conference on Big Data (Big Data) 2018 Dec 10 (pp. 3432-3441). IEEE.

[20] Karsoum S, Gruenwald L, Barrus C, Leal E. Using timed sequential patterns in the transportation industry. In2019 IEEE International Conference on Big Data (Big Data) 2019 Dec 9 (pp. 3573-3582). IEEE.

[21] Li H, Zhou X, Pan C. Study on GSP algorithm based on Hadoop. In2015 IEEE 5th International Conference on Electronics Information and Emergency Communication 2015 May 14 (pp. 321-324). IEEE.

[22] McPherson, R. A., C. Friedrich, K. C. Crawford, R. L. Elliott, J. R. Kilby, D. L. Grimsley, J. E. Martinez, J. B. Basara, B. G. Illston, D. A. Morris, K. A. Kloesel, S. J. Stadler, A. D. Melvin, A.J. Sutherland, and H. Shrivastava, 2007: Statewide monitoring of the mesoscale environment: A technical update on the Oklahoma Mesonet. J. Atmos. Oceanic Technol., 24, pp. 301–321.

[23] McPherson RA, Fiebrich CA, Crawford KC, Kilby JR, Grimsley DL, Martinez JE, Basara JB, Illston BG, Morris DA, Kloesel KA, Melvin AD. Statewide monitoring of the mesoscale environment: A technical update on the Oklahoma Mesonet. Journal of Atmospheric and Oceanic Technology. 2007 Mar;24(3):301-21.

[24] Hanauer DA. Experiences with mining temporal event sequences from electronic medical records: initial successes and some challenges. InProceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining 2011 Aug 21 (pp. 360-368).

[25] Han J, Pei J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu M. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. Inproceedings of the 17th international conference on data engineering 2001 Apr (pp. 215-224). IEEE.

[26] Pramono YW. Anomaly-based intrusion detection and prevention system on website usage using rule-growth sequential pattern analysis: Case study: Statistics of Indonesia (BPS) website. In2014 International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA) 2014 Aug 20 (pp. 203-208). IEEE.

[27] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. InInternational conference on extending database technology 1996 Mar 25 (pp. 1-17). Springer, Berlin, Heidelberg.

[28] The 100-Year Flood [Internet]. (n.d.). cited [2021 Oct 17] Retrieved from https://www.usgs.gov/special-topic/water-science-school/science/100-year-flood?qt-science_center_objects=0#qt-science_center_objects

[29] The Beaufort Scale, How is wind speed measured? [Internet]. July 2018 Cited [2021 Oct 17]. Retrieved 10/17/2021 from https://www.rmets.org/resource/beaufort-scale

[30] Titarenko SS, Titarenko VN, Aivaliotis G, Palczewski J. Fast implementation of pattern mining algorithms with time stamp uncertainties and temporal constraints. Journal of Big Data. 2019 Dec;6(1):1-34.

[31] Wei YQ, Liu D, Duan LS. Distributed PrefixSpan algorithm based on MapReduce. In2012 International Symposium on Information Technologies in Medicine and Education 2012 Aug 3 (Vol. 2, pp. 901-904). IEEE.

[32] What is the heat index? [Internet].(n.d.) cited [2021 Oct 17] Retrieved from https://www.weather.gov/ama/heatindex

[33] Yang H, Gruenwald L, Boulanger M. A novel real-time framework for extracting patterns from trajectory data streams. InProceedings of the 4th ACM SIGSPATIAL International Workshop on GeoStreaming 2013 Nov 5 (pp. 26-32).

[34] Chiu DY, Wu YH, Chen AL. An efficient algorithm for mining frequent sequences by a new strategy without support counting. InProceedings. 20th International Conference on Data Engineering 2004 Apr 2 (pp. 375-386). IEEE.

[35] Yan X, Han J, Afshar R. CloSpan: Mining: Closed sequential patterns in large datasets. InProceedings of the 2003 SIAM international conference on data mining 2003 May 1 (pp. 166-177). Society for Industrial and Applied Mathematics.

[36] Wang J, Han J. BIDE: Efficient mining of frequent closed sequences. InProceedings. 20th international conference on data engineering 2004 Apr 2 (pp. 79-90). IEEE

[37] Cao L, Dong X, Zheng Z. e-NSP: Efficient negative sequential pattern mining. Artificial Intelligence. 2016 Jun 1;235:156-82

[38] Wang W, Cao L. VM-NSP: vertical negative sequential pattern mining with loose negative element constraints. ACM Transactions on Information Systems (TOIS). 2021 Feb 17;39(2):1-27.

[39] Zheng Z, Zhao Y, Zuo Z, Cao L. Negative-GSP: An efficient method for mining negative sequential patterns. InConferences in Research and Practice in Information Technology Series 2009 Dec 1.

[40] Qiu P, Gong Y, Zhao Y, Cao L, Zhang C, Dong X. An efficient method for modeling nonoccurring behaviors by negative sequential patterns with loose constraints. IEEE Transactions on Neural Networks and Learning Systems. 2021 Mar 17.

[41] Qiu P, Gong Y, Zhao Y, Cao L, Zhang C, Dong X. An efficient method for modeling nonoccurring behaviors by negative sequential patterns with loose constraints. IEEE Transactions on Neural Networks and Learning Systems. 2021 Mar 17.

[42] Gupta SK. HUFTI-SPM: high-utility and frequent time-interval sequential pattern mining from transactional databases. International Journal of Data Science and Analytics. 2022 Apr;13(3):239-50.

[43] Yin J, Zheng Z, Cao L. USpan: an efficient algorithm for mining high utility sequential patterns. InProceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining 2012 Aug 12 (pp. 660-668).

[44] Han J, Dong G, Yin Y. Efficient mining of partial periodic patterns in time series database. InProceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337) 1999 Mar 23 (pp. 106-115). IEEE.

[45] Fournier-Viger P, Yang P, Lin JC, Duong QH, Dam TL, Frnda J, Sevcik L, Voznak M. Discovering periodic itemsets using novel periodicity measures. Advances in Electrical and Electronic Engineering. 2019 Mar 17;17(1):33-44.

[46] Surana A, Kiran RU, Reddy PK. An efficient approach to mine periodic-frequent patterns in transactional databases. InPacific-Asia Conference on Knowledge Discovery and Data Mining 2011 May 24 (pp. 254-266). Springer, Berlin, Heidelberg.

[47] Tanbeer SK, Ahmed CF, Jeong BS, Lee YK. Discovering periodic-frequent patterns in transactional databases. InPacific-Asia Conference on Knowledge Discovery and Data Mining 2009 Apr 27 (pp. 242-253). Springer, Berlin, Heidelberg.