This paper was published in the BDA 2022 conference.
This is an updated version where some minor errors
have been fixed and a few more details are added to make the paper clearer.

FastTIRP: Efficient discovery of Time-Interval Related Patterns

Philippe Fournier-Viger[1 [0000−0002−7680−9899]], Yuechun Li[1],
M. Saqib Nawaz[1 [0000−0001−9856−2885]], and Yulin He[2,1 [0000−0002−3415−0686]]

[1] Shenzhen University, Shenzhen, China
[2] Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen China
{philfv, msaqibnawaz}@szu.edu.cn, 935764184@qq.com, yulinhe@gml.ac.cn

Abstract. Finding frequent patterns in discrete sequences of symbols or events can be useful to understand data, support decision-making and make predictions. However, many studies on analyzing event sequence do not consider the duration of events, and thus the complex time relationships between them (e.g. an event may start at the same time as another event but end before). To find frequent sequential patterns in data where events have a start and end time, an emerging topic is time-interval related pattern (TIRP) mining. Several algorithms have been proposed for this task but efficiency remains a major issue due to the very large search space. To provide a more efficient algorithm for TIRP mining, this paper presents a novel algorithm called FastTIRP. It utilizes a novel Pair Support Pruning (PSP) optimization to reduce the search space. Experiments show that FastTIRP outperforms the state-of-the-art VertTIRP algorithm in terms of runtime on four benchmark datasets.

Keywords: Time-intervals · Event sequences· Efficiency · Pair Support Pruning · Discrete sequences.

1 Introduction

A popular data representation is sequences. A sequence is a list of symbols or events, and can encode various information such as a list of moves in a chess game, and a list of events that occur in a computer network. To identify useful knowledge that may be hidden in sequence data, several pattern mining algorithms have been designed. The aim is to detect patterns that appear frequently in one or more sequences and satisfy some user-defined constraints. Some of the most popular types of patterns are sequential patterns [1, 7, 15] and episodes [4, 6, 10, 11, 16], which are respectively patterns found in a set of sequences, or in a very long sequence.

Several algorithms have been proposed to find patterns in sequences, to help users to understand the data and make prediction. However, most studies assume a simple time representation, where each event is represented by a single timestamp. An example of sequence using this representation is shown in Fig. 1(a).

That sequence indicates that a person attended a meeting at 9:00 AM, made a phone call at 10:00 AM and then another phone call at 11:00 AM. A major problem with this time representation is that the duration of events is not taken into account (e.g. there is no information about the duration of the meeting or that of the phone calls). As a result, most algorithms can only find patterns involving two types of relationships between events: two events are either simultaneous or one appear before the other. For instance, a traditional sequential pattern mining algorithm could find a pattern indicating that a meeting is followed by a phone call.

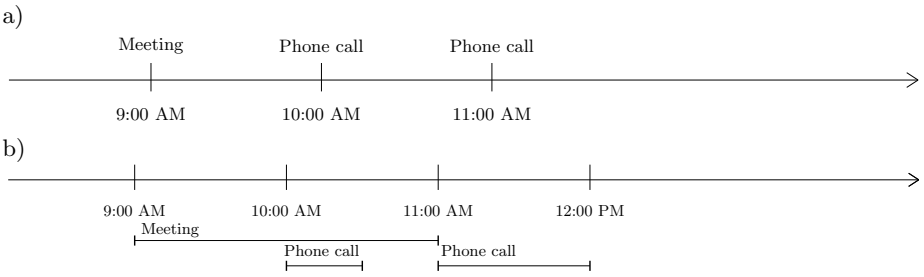

Fig. 1: A (a) simple event sequence and (b) a time-interval sequence

Recently, a richer time representation has been studied, called *time-interval sequence*, where events are represented as time-intervals with a start and end time. An example of time-interval sequence is shown in Fig. 1(b). It indicates that a person attended a meeting from 9:00 AM till 11:00 AM, and made a phone call *during* that meeting from 10:00 AM to 10:30 AM, and then made another phone call from 11:00 AM to 12:00 PM. It is easy to see that this representation shown in Fig. 1(b) carries more information than the one shown in Fig. 1(a). For instance, we can observe that the first phone call did not occur after the meeting (as it seemed in Fig. 1(a)), but during the meeting.

A popular task to find frequent patterns in such data is time-interval related pattern (TIRP) mining [9, 12, 13]. The goal is to find a form of sequential patterns called TIRP where events may be connected by various complex temporal relationships (i.e. an event overlaps with another event, an event starts at the same time as another event but end before, etc.). TIRP mining is a harder problem than traditional sequential pattern mining (SPM) [3] because each pair of events may be connected by multiple relationship types. TIRP mining has been used in several applications such as to analyze medical treatments [8], discover patterns in stock prices [14], and analyzing skating data [5].

Many algorithms have been proposed for TIRP mining such as ZMiner [9], VertTIRP [12], KarmaLego and DarmaLego [13]. However efficiency remains a major issue as the number of possible patterns is very large. To the best of our knowledge, the state-of-the-art algorithm is VertTIRP. It performs a depth-first search and relies on a vertical data structure to encode data about patterns.

The main drawback of VertTIRP is that it explores the search space by combining pairs of patterns and perform a costly join operation using vertical data structures to calculate their support.

In this paper, we aim to address the efficiency problem of TIRP mining by proposing a novel algorithm called FastTIRP. It is an enhanced version of VertTIRP where a novel technique called Pair Support Pruning (PSP) is applied. This technique consists of pre-calculating the frequency information about pairs of events to avoid performing join operations. Experiments on several benchmark datasets show that FastTIRP outperforms VertTIRP in terms of runtime as it performs less join operations, and also has good performance compared to KarmaLego.

The remaining sections of this paper are structured as follows. The problem definition of TIRP mining is explained in Section 2. The proposed FastTIRP algorithm is presented in Section 3. Section 4 provides the experiments results, followed by a conclusion in Section 5.

2 Problem Definition

The problem of discovering TIRPs is defined based on the following definitions, here presented using a similar notation as in the VertTIRP paper [12, 13].

Definition 1 (Event type). *Let there be a set of event types* $E = \{e_1, e_2, \dots e_n\}$. *Furthermore, assume that there exists a total order on these events denoted as* $<$ *which is the lexicographical order.*

Example 1. Consider a set of events $E = \{A, B, C\}$ containing three event types, and the lexicographical order defined as $A < B < C$.

Definition 2 (Symbolic time interval). *A symbolic time interval* I *is a triple of the form* $I = (start, end, e)$ *where start and end are numbers respectively indicating the start time and end time of an event of type* $e \in E$. *The notation* $I.start$, $I.end$ *and* $I.e$ *will be used in the following to refer to the elements* ***start***, ***end*** *and* ***e*** *of a symbolic time interval* I.

To compare the timestamps of events, it is useful to use an approximate comparison of events so as to handle noise. This is done using two relationships, defined based on a user-defined epsilon ($\epsilon \geq 0$) value [13].

Definition 3 (Temporal relations between timestamps). *Let there be two timestamps* t_i *and* t_j *and a user-defined number* ϵ. *It is said that* t_i *and* t_j *are quasi-equal if* $|t_i - t_j| \leq \epsilon$, *which is denoted as* $t_i =_\epsilon t_j$. *Moreover, it is said that* t_i *precedes* t_j *if* $t_j - t_i > \epsilon$, *which is denoted as* $t_i <_\epsilon t_j$ *or* $t_j >_\epsilon t_i$.

Definition 4 (Time-interval sequence). *A time-interval sequence (TIS)* S *is an ordered list of symbolic time intervals* $S = \langle I_1, I_2, \dots I_q \rangle$. *Symbolic time intervals within a TIS* S *are sorted according to a total order* $\prec$ *such that* $I_i \prec I_j$ *for any* I_i *and* I_j *if* $(I_i.start <_\epsilon I_j.start) \vee (I_i.start =_\epsilon I_j.start \wedge I_i.end <_\epsilon I_j.end) \vee (I_i.start =_\epsilon I_j.start \wedge I_i.end =_\epsilon I_j.end \wedge I_i.e < I_j.e)$.

Definition 5 (Time-interval sequence database). *A time-interval sequence database (TISD) D is a list of time-interval sequences $D = \langle S_1, S_2, \ldots S_m \rangle$. The sequence identifier of a sequence $S_i \in D (1 \leq i \leq m)$ is said to be i.*

Example 2. To illustrate these definitions, consider the TISD $D = \{S_1 : \langle (8, 11, C), (8, 12, A), (10, 12, B) \rangle\ S_2 : \langle (8, 12, C), (10, 16, A), (15, 18, B) \rangle\ S_3 : \langle (10, 16, C), (14, 16, B), (14, 19, B), (15, 19, A) \rangle\}$, which is represented visually in Fig. 2 [12]. This TISD contains three time-interval sequences called S_1, S_2 and S_3. The first sequence indicates that an event C started at time 8 and ended at 11, an event A started at time 8 and ended at time 12, and an event B started at time 10 and ended at time 12. The two other time-interval sequences can be interpreted in a similar way.

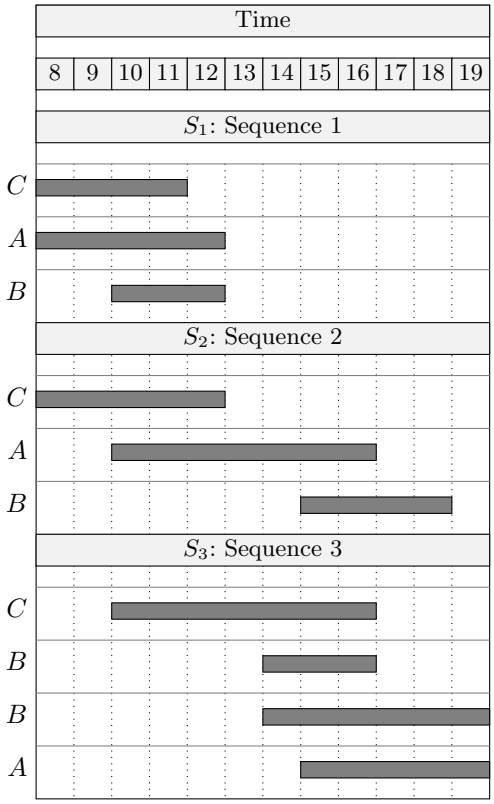

Fig. 2: An example time-interval sequence database

To be able to compare events, it is important to have a set of temporal relations. In this paper, the set of temporal relations from VertTIRP [12] is used. It

is based on the 13 temporal relations introduced by Allen, with some modifications to handle epsilon [13] and resolve some ambiguities in the definitions used in prior work.

Definition 6 (Temporal relations between time intervals). *Let $r(I_i, I_j)$ be a function that determines the temporal relation between any pair of symbolic time intervals I_i and I_j. There are eight temporal relationships $\omega = \{b, m, o, l, c, f, e, s\}$, called before, meets, overlaps, left contains, contains, is finished by, equal, and starts, respectively. Assume also that two optional constraints called $mingap \geq 0$ and $maxgap \geq 0$ are defined by the user. The function $r(I_i, I_j)$ returns*
$r(I_i, I_j) = \mathbf{b}$ *if* $(I_j.start - I_i.end) > \epsilon \wedge (I_j.start - I_i.end) < maxgap \wedge (I_j.start - I_i.end) > mingap$.
$r(I_i, I_j) = \mathbf{m}$ *if* $|I_j.start - I_i.end| \leq \epsilon \wedge (I_j.start - I_i.start > \epsilon \wedge (I_j.end - I_i.end) > \epsilon$
$r(I_i, I_j) = \mathbf{o}$ *if* $(I_j.start - I_i.start) > \epsilon \wedge (I_i.end - I_j.start > \epsilon \wedge (I_j.end - I_i.end) > \epsilon$
$r(I_i, I_j) = \mathbf{l}$ *if* $\epsilon > 0 \wedge |I_j.start - I_i.start| \leq \epsilon \wedge (I_i.end - I_j.end) > \epsilon$
$r(I_i, I_j) = \mathbf{c}$ *if* $(I_j.start - I_i.start) > \epsilon \wedge (I_i.end - I_j.end) > \epsilon$
$r(I_i, I_j) = \mathbf{f}$ *if* $(I_j.start - I_i.start) > \epsilon \wedge |I_j.end - I_i.end| \leq \epsilon$
$r(I_i, I_j) = \mathbf{e}$ *if* $|I_j.start - I_i.start| \leq \epsilon \wedge |I_j.end - I_i.end| \leq \epsilon$
$r(I_i, I_j) = \mathbf{s}$ *if* $|I_j.start - I_i.start| \leq \epsilon \wedge (I_j.end - I_i.end) > \epsilon$.

The problem of finding frequent time-interval related patterns (TIRPs) aims at finding TIRPs in a time-interval sequence database. A TIRP is defined as follows.

Definition 7 (Time-interval related pattern). *A time-interval related pattern (TIRP) X is a pair of the form $X = (ev, rel)$ where **rel** is an ordered list of temporal relations from ω and **ev** is an ordered list of events from E. Assume that **ev** contains k events. Then, the list **rel** indicates the temporal relations as follows: relations $= \langle r(I_1, I_2), \ldots\ r(I_1, I_k),\ r(I_2, I_3), \ldots\ r(I_{k-1}, I_k)\rangle$. In the following, the notation $r_{i,j}$ refers to $r(e_i, e_j)$ where e_i and e_j are the i-th and j-th events in* **ev**.

Example 3. Continuing the running example, the TIRP $(CB, \mathbf{o})$ occurs in sequence 1 and sequence 3 in the symbolic time intervals $\langle (8, 11, C),\ (10, 12, B)\rangle$ and $\langle (10, 16, C),\ (14, 19, B)\rangle$, respectively. The TIRP $(CAB, \mathbf{sof})$ appears in sequence 1 at $\langle (8, 11, C), (8, 12, A),\ (10, 12, B)\rangle$.

An important observation is that multiple TIRPs may have the same events but only differs by their temporal relations. This gives rise to the definition of Symbol-TIRP [12].

Definition 8 (Symbol-TIRP). *Let the notation $\overline{ev}$ denotes the set of all TIRPs having the same list of events ev. $\overline{ev}$ is said to be a Symbol-TIRP.*

Example 4. $\overline{BC} = \{(BC, \mathbf{o}), (BC, \mathbf{f})\}$ is the Symbol-TIRP containing the TIRPS having the list of events $ev = \langle BC\rangle$.

To find TIRPs in a database, it is necessary to rely on the concept of *match* between a TIRP and a time-interval sequence.

Definition 9 (TIRP matching). *Let there be a time-interval sequence* $S = \langle I_1, I_2, \ldots, I_q \rangle$ *and a TIRP* $X = (ev, rel)$*. Then,* X *is said to match* S *if* $\forall e_i, e_j \in ev$, $\exists I_\alpha, I_\beta \in S$ *such that* $e_i = I_\alpha.e$ *and* $e_j = I_\beta.e$ *and* $r(I_\alpha, I_\beta) = r_{i,j}$*.*

Example 5. Consider the time-interval sequence from the running example, $S = \langle (8, 11, C), (8, 12, A), (10, 12, B) \rangle$. For the TIRP $X = (CAB, \mathbf{sof})$, we have $I_1.e = C$, $I_2.e = A$, $I_3.e = B$, $r(I_1, I_2) = s$, $r(I_1, I_3) = o$ and $r(I_2, I_3) = f$. Hence, X is said to match S.

To find interesting TIRPs in a sequence, various interestingness function can be used. The two main functions are the horizontal support and vertical support [12].

Definition 10 (Horizontal support). *Let there be a TIRP* X *and a time-interval sequence* S*. The horizontal support of* X *in* S *is defined as* $h(X) = |S_X|$*, where* $|S_X|$ *is the total number of symbolic time intervals that match with* X*.*

Definition 11 (Vertical support of a TIRP). *Let there be a TIRP* X *and a time-interval sequence database* D*. The vertical support of* X *in* D *is defined as* $v(X) = |D_X|$*, where* $|D_X|$ *is the total number of sequences that match with* X*.*

Example 6. Continuing the running example, consider that $X = (CB, \mathbf{o})$. It can be found that X is matching with $\langle (8, 11, C), (10, 12, B) \rangle$ in sequence 1, and it also matches with $\langle (10, 16, C), (15, 19, B) \rangle$ in sequence 3. Thus, the vertical support of X is $v(X) = 2$.

Definition 12 (Vertical support of an S-TIRP). *Let there be a S-TIRP* $\overline{X} = \{X_1, X_2, \ldots, X_z\}$ *and a time-interval sequence database* D*. The vertical support of* $\overline{X}$ *in* D *is defined as* $v(\overline{X}) = |\bigcup_{i=1}^{z} D_{X_i}|$*.*

Example 7. Continuing the running example, consider that $\overline{X} = \{(CB, \mathbf{o}), (CB, \mathbf{b}), (CB, \mathbf{f})\}$. It can be found that the TIRP $(CB, \mathbf{o})$ matches with sequences 1 and 3, that the TIRP $(CB, \mathbf{b})$ matches with sequences 2, and that the TIRP $(CB, \mathbf{f})$ matches with sequences 3. Hence, the S-TIRP $\overline{X}$ matches with sequences 1, 2 and 3, and its vertical support is $v(\overline{X}) = 3$.

Definition 13 (Problem of discovering all frequent TIRPS). *Let there be a time-interval sequence database* D*, a user-defined minimum support threshold* $minsup > 0$ *and some optional constraints:* $mingap \geq 0$*,* $maxgap \geq 0$*,* $\epsilon \geq 0$*. The problem of discovering all frequent S-TIRPS is to enumerate all frequent S-TIRPs, that is each S-TIRP* s *such that* $v(s) \geq minsup$ *under the constraints [12].*

Example 8. Back to the running example, if *minsup* is set to 2, the frequent Symbol-TIRPs are $\overline{A}$, $\overline{AB}$, $\overline{B}$, $\overline{C}$, $\overline{CA}$, $\overline{CAB}$, and $\overline{CB}$ with a support of 3, 2, 3, 3, 3, 2, 3, respectively.

Note that it is possible to add some additional constraints such as the minimum and maximum duration of a TIRP (see [12]).

3 The FastTIRP Algorithm

This section presents the proposed FastTIRP algorithm. The motivation for developing this algorithm is that the state-of-art VertTIRP algorithm can have very long runtimes. After doing some preliminary analysis of the performance of VertTIRP, we found that its most costly operation is the join operation. This operation is used to calculate the support of a new pattern obtained by combining two existing patterns. To reduce the number of join operations, FastTIRP integrates a novel optimization called Pair Support Pruning (PSP) in VertTIRP. This optimization is based on a structure called Pattern Support Matrix (PSM).

The next subsections first explains briefly the search process and then describes the proposed PSP technique in details.

3.1 The search process

The pseudocode of the proposed FastTIRP algorithm is shown in Algorithm 1. The input is a time-interval sequence database D, a minimum support threshold $minsup$ and some optional parameters ϵ, $mingap$ and $maxgap$. The output is the set of all frequent S-TIRPs. The algorithm first configures the function $r(I_i, I_j)$ (to evaluate temporal relations between time-intervals) based on ϵ, $mingap$ and $maxgap$, and also configures the temporal relation $<_\epsilon$ (to evaluate temporal relations between timestamps) based on ϵ. Then, a function $FindFrequentEvents$ is called, which scans the input database to identify the set $FrequentEvents$ of all the TIRPs that are frequent and contain a single event. At the same time, the algorithm builds a vertical structure for each of these TIRPs. This structure is the same as in the VertTIRP algorithm [12] and is used to compute the support of each TIRP. After this, the algorithm scans the database to build the proposed PSM data structure, which will be presented in the next subsection. Then, the algorithm initializes a variable $Patterns$ to store all frequent TIRPs. Then a loop is done to try to extend each pattern p that is in $FrequentEvents$. This is done by calling a procedure called $Search$ with p, the list of frequent events $FrequentEvents$ that could be appended to p to form larger patterns, the minimum support threshold, and the PSM structure. The $Search$ function performs a depth-first search and returns all patterns that are frequent and obtained by appending events at the end of p. These patterns are then added to the set $Patterns$. Finally, the algorithm returns the set $Patterns$ containing all frequent TIRPs.

The search procedure is shown in Algorithm 2. The input is a Symbol-TIRP $\overline{p}$, a set $FrequentEvents$ containing Symbol-TIRPs each having a single event that could be appended to $\overline{p}$ to form larger TIRPs, the $minsup$ threshold, and the PSM structure. The procedure first initializes a set $NewPatterns$ and put the pattern $\overline{p}$ inside. Then, a variable $LocalFrequentEvents$ is initialized to the empty set, and will be used to store single events that are locally frequent. After that a loop is done on each frequent Symbol-TIRP $\overline{q}$ from the set $FrequentEvents$ to try to form a larger pattern by appending $\overline{q}$ at the end of $\overline{p}$ to create a larger Symbol-TIRP New. But before doing this step, the new

Algorithm 1: FastTIRP

input : D: a time-interval sequence database,
$minsup$: the required minimum support,
$\epsilon, mingap$, and $maxgap$: optional parameters
output: all frequent S-TIRPs

1 Configure the temporal relations based on ϵ, $mingap$ and $maxgap$;
2 $FrequentEvents$ = `FindFrequentEvents(`$D, minsup$`)`;
3 PSM = `BuildPSM(`D`)`;
4 $Patterns = \emptyset$;
5 **foreach** $\overline{p} \in FrequentEvents$ **do**
6 $\quad NewPatterns$ = `Search(`$\overline{p}, FrequentEvents, minsup$`)`;
7 $\quad Patterns = Patterns \cup NewPatterns$;
8 **end**
9 Return $Patterns$;

pruning strategy PSP is checked, which will be presented in the next section. If that strategy determines that $\overline{p}$ and $\overline{q}$ cannot be combined, then this combination is skipped. Otherwise, the algorithm applies a join operation on the vertical structures of $\overline{p}$ and $\overline{q}$ to create the vertical structure of the resulting pattern $\overline{New}$. If the support of the resulting pattern $\overline{New}$ is no less than $minsup$, then it is added to the set of frequent TIRPs $NewPatterns$ and $\overline{q}$ is added to the set of locally frequent events $LocalFrequentEvents$. After that that loop ends, another loop is done to recursively try to extend each frequent pattern $\overline{New}$ in $NewPatterns$. This is done by calling the $Search$ procedure with $\overline{New}$, the locally frequent events $LocalFrequentEvents$, $minsup$ and the PSM. Finally, all the patterns that have been found are returned by the search procedure.

The above description did not explain all the details of the algorithm such as how to create and join the vertical data structures of TIRPs. This is because these operations are the same as in the VertTIRP algorithm. Interested reader can refer to the paper describing VertTIRP for detailed explanations [12]. The difference between FastTIRP and VertTIRP is the addition of Line 3 in Algorithm 1 to build the PSM structure, and Line 5 in Algorithm 2 to use the PSM structure to avoid performing joins. The next subsection explains these differences in details.

3.2 The Pair Support Pruning technique

The key difference between the proposed FastTIRP algorithm and VertTIRP is a novel data structure called Pattern Support Matrix (PSM) which is used to reduce the number of join operations that are performed. The PSM is built by reading the input database once and storing the co-occurrence frequency of each pair of events. Formally, the PSM can be defined as follows:

Definition 14 (Pair Support Matrix). *The Pair Support Matrix of a time-interval sequence database D is denoted as PSM. For each pair of event types*

Algorithm 2: Search

input : $\overline{p}$: a Symbol-TIRP,
$FrequentEvents$: a set of Symbol-TIRPs having a single event,
$minsup$: the minimum support threshold
output: $\overline{p}$ and all frequent S-TIRPs that extend $\overline{p}$

```
1  NewPatterns = {p̄};
2  LocalFrequentEvents = ∅;
3  foreach q̄ ∈ FrequentEvents do
4  |  // Check the PSP pruning condition
5  |  if CheckPSP(p̄, q̄, minsup) then
6  |  |  New̄ = Join(p̄, q̄);
7  |  |  if v(New̄) ≥ minsup then
8  |  |  |  NewPatterns = NewPatterns ∪ {New̄};
9  |  |  |  LocalFrequentEvents = LocalFrequentEvents ∪ {q̄};
10 |  |  end
11 |  end
12 end
13 foreach New̄ ∈ NewPatterns do
14 |  Extensions = Search(New̄, LocalFrequentEvents, minsup, PSM);
15 |  NewPatterns = NewPatterns ∪ Extensions;
16 end
17 return NewPatterns;
```

$e_1, e_2 \in E$, *the PSM stores a triple of the form* (e_1, e_2, $h(e_1, e_2)$) *where* $h(e_1, e_2)$ *is the number of symbolic time intervals containing* $e_1 <_\epsilon e_2$. *In case the user utilizes the maximum gap constraint maxgap, then* $h(e_1, e_2)$ *is defined as the number of symbolic time intervals containing* $e_1 <_\epsilon e_2$ *within a time maxgap.*

There are several ways of implementing the PSM structure. The most simple way is to define it as a two dimensional matrix where there is a row and column for each event. Then, for a given column and row representing some events e_1 and e_2, the content of the cell is the number of symbolic time intervals containing $e_1 <_\epsilon e_2$. As example, Table 1 shows the PSM structure built for the running example implemented as a two dimensional matrix. The PSM structure contains several values indicating for instance that event type B appears in one time-interval before A, but A appears in 2 symbolic time-intervals before B.

Table 1: The PSM structure as a full matrix

	A	B	C
A	0	1	3
B	2	1	3
C	0	0	0

To reduce memory, the PSM can be also implemented as a triangular matrix as shown in Table 2. In this case, for any two events $e_1, e_2 \in E$, the two cells (e_1, e_2, $h(e_1, e_2)$) and (e_2, e_1, $h(e_2, e_1)$) are merged as: (e_1, e_2, $h(e_1, e_2) + h(e_2, e_1)$). This is the implementation used in the experiments of this paper.

But note that the full matrix or triangular matrix can still contain many zeros. In this case, the PSM could also be implemented as a sparse matrix (e.g. using a hash map of hash maps) to avoid storing cells containing zeros. This could be beneficial for datasets with a very large number of event types. It is also possible to redefine the PSM to store the vertical support instead of the horizontal support.

Table 2: The PSM structure as a triangular matrix

	A	B	C
A	0	3	3
B		1	3
C			0

The PSM structure is used to reduce the number of join operations done by FastTIRP as follows. Before joining an event $\overline{q}$ with a Symbol-TIRP $\overline{p}$ in Line 6 of Algorithm 2, FastTIRP checks the support of the last event of $\overline{p}$ with event $\overline{q}$ in the PSM. If that value is no greater than *minsup*, it is unnecessary to append $\overline{q}$ to p to form a larger pattern, as the result will be an infrequent TIRP. Thus, the join operation is not performed.

For example, consider that $minsup = 3$, $\overline{p} = \overline{CB}$, which matches with $\langle(10, 16, C), (14, 16, B)\rangle$ and that $\overline{q} = \overline{B}$, which matches with $(14, 19, B)$. By looking at the PSM, we can find that the support of B with B is 1 and thus that $\overline{p}$ should not be combined with $\overline{q}$ to obtain a Symbol-TIRP $\overline{CBB}$.

The PSM technique can be used to effectively reduce the number of join operations as it will be demonstrated in the experiments. The design of PSP is inspired by a similar structure called CMAP used in sequential pattern mining [1].

4 Experimental Evaluation

This section presents the experimental evaluation of the proposed FastTIRP algorithm and obtained results are discussed. In experiments, the FastTIRP was compared with two algorithms for TIRP mining, which are: VertTIRP [12] and KarmaLego [13]. A prior study [12] compared VertTIRP with KarmaLego and obtained results show that VertTIRP was on overall faster than KarmaLego. However, in the experiment, the two algorithms were implemented in different programming languages (Python and C#, respectively), which may make the results unreliable. To avoid this issue, in this work, the three algorithms are implemented in C#. The C# implementations of VertTIRP and KarmaLego

were obtained from: `http://github.com/TIRPClo`. The FastTIRP algorithm was implemented by modifying the C# code of VertTIRP.

All experiments were run on a laptop running Windows 11, with 16 GB of RAM and an 11th generation Core i7-11800H processor. Four datasets, namely *Diabetes*, *Hepatitis*, *Smarthome* and *ASL*, having various characteristics were used to evaluate the performance of the three algorithms. The first two datasets contain medical data, while the third and fourth datasets contain smarthome events and sign language utterances transcribed from video recordings, respectively. The main characteristics of the four datasets are listed in Table 3. Diabetes (Hepatitis) and Smarthome (ASL) contain 2,038 (498) and 89 (65) entities (sequences), 80,538 (48,029) and 23,213 (2,037) symbolic time intervals, and 35 (63) and 95 (146) event (symbol) types respectively. These datasets were downloaded from `http://github.com/TIRPClo`. For all experiments, epsilon was set to zero, *mingap* to 0 and *maxgap* to 30.

Table 3: Characteristics of the datasets

Dataset	**Sequences**	**Time intervals**	**Event types**
Diabetes	2,038	80,538	35
Hepatitis	498	48,029	63
Smarthome	89	23,213	95
ASL	65	2,037	146

4.1 Influence of *minsup* on runtime, number of joins and patterns

In this section, experiments were carried out to first evaluate the efficiency of the algorithms in terms of runtime and number of joins operations that are performed. Both runtime and number of joins were measured while varying the *minsup* parameter for each dataset. Fig. 3 shows the execution time (left side) of the three algorithms and the number of joins operations (right side) for two algorithms (FastTIRP, VertTIRP) on four datasets.

For runtime, it is observed that FastTIRP was faster, overall, than VertTIRP on all datasets. For lower *minsup* values, the difference in runtime is higher. However, the runtime difference between FastTIRP and VertTIRP tend to decrease as *minsup* is increased. Interestingly, on all datasets, FastTIRP was about 20% faster than VertTIRP. On the other hand, KarmaLego performed better than FastTIRP on two datasets: Hepatitis and ASL. KarmaLego was indeed faster than FastTIRP for lower *minsup* values. For the Diabetes and ASL datasets, the average execution times for KarmaLego was high. That is why they are not added in the figure. KarmaLego took more than half hour (the time limit we set for runtime execution).

The number of joins performed by FastTIRP and VertTIRP is also observed to assess the ability of the PSM technique at reducing the number of joins.

FastTIRP performed less number of joins than VertTIRP on all datasets (Fig. 3). This shows that the PSM technique in FastTIRP was able to reduce the number of joins in the Diabetes, Hepatitis, Smarthome and ASL datasets by up to 24 %, 61 %, 94%, and 91%, respectively.

Next, we investigate the number of S-TIRPs discovered by the FastTIRP algorithm on four datasets for the smallest and largest *minsup* values used in the experiments. The results are listed in Table 4. More S-TIRPs were found in the Diabetes dataset, followed by Hepatitis, Smarthome and ASL. The main reason for this is that Diabetes contain more sequences and time intervals, followed by Hepatitis, Smarthome and ASL respectively. Moreover, more (less) S-TIRPs can be discovered by decreasing (increasing) the *minsup* value.

Table 4: Number of frequent S-TIRPs found in each dataset

Dataset	*minsup* **range**	**Number of S-TIRPs**
Diabetes	[40, 90]	[69692, 30094]
Hepatitis	[40, 90]	[44148, 25802]
Smarthome	[40, 90]	[19944, 4684]
ASL	[10, 90]	[1807, 72]

4.2 Influence of *minsup* on the overall memory usage

This section analyzes the memory used by algorithms during execution. The results for the three algorithms are listed in Table 5.

FastTIRP used less memory compared to VertTIRP on the Diabetes dataset that contain more sequences and time intervals. On the Smarthome and ASL datasets, that contain less sequences, the memory usage of FastTIRP and VertTIRP is almost similar. Interestingly, the memory usage of FastTIRP and VertTIRP on the Smarthome dataset was higher than on the Hepatisis dataset. However, the Smarthome dataset has less sequences and time intervals than the Hepatitis dataset. KarmaLego performed better than FastTIRP and VertTIRP on the Diabetes and ASL datasets. KarmaLego's results for Hepatitis and Smarthome are not added in the Table 5 as it was unable to terminate for various *minsup* values within the time limit of half hour that we set for termination.

We can further analyze the memory usage of FastTIRP by calculating the size of the proposed PSM data structure. Let $|E|$ be the number of event types. If the PSM is implemented as a full matrix, then the size can be calculated as $SizeFullMatrix = |E|^2 \times SizeOfValue$, that is the number of events to the power of two, multiplied by the number of bytes required to store a support value. In the C# implementation of FastTIRP, support values are stored using integers of four bytes. Thus, the size of the PSM as a full matrix for each dataset is as follows: Diabetes: $35^2 \times 4$ bytes$= 4,900$ bytes, Hepatitis: $63^2 \times 4$ bytes$= 15,876$ bytes, Smarthome: $95^2 \times 4$ bytes$= 36,100$ bytes, and ASL: $146^2 \times 4$

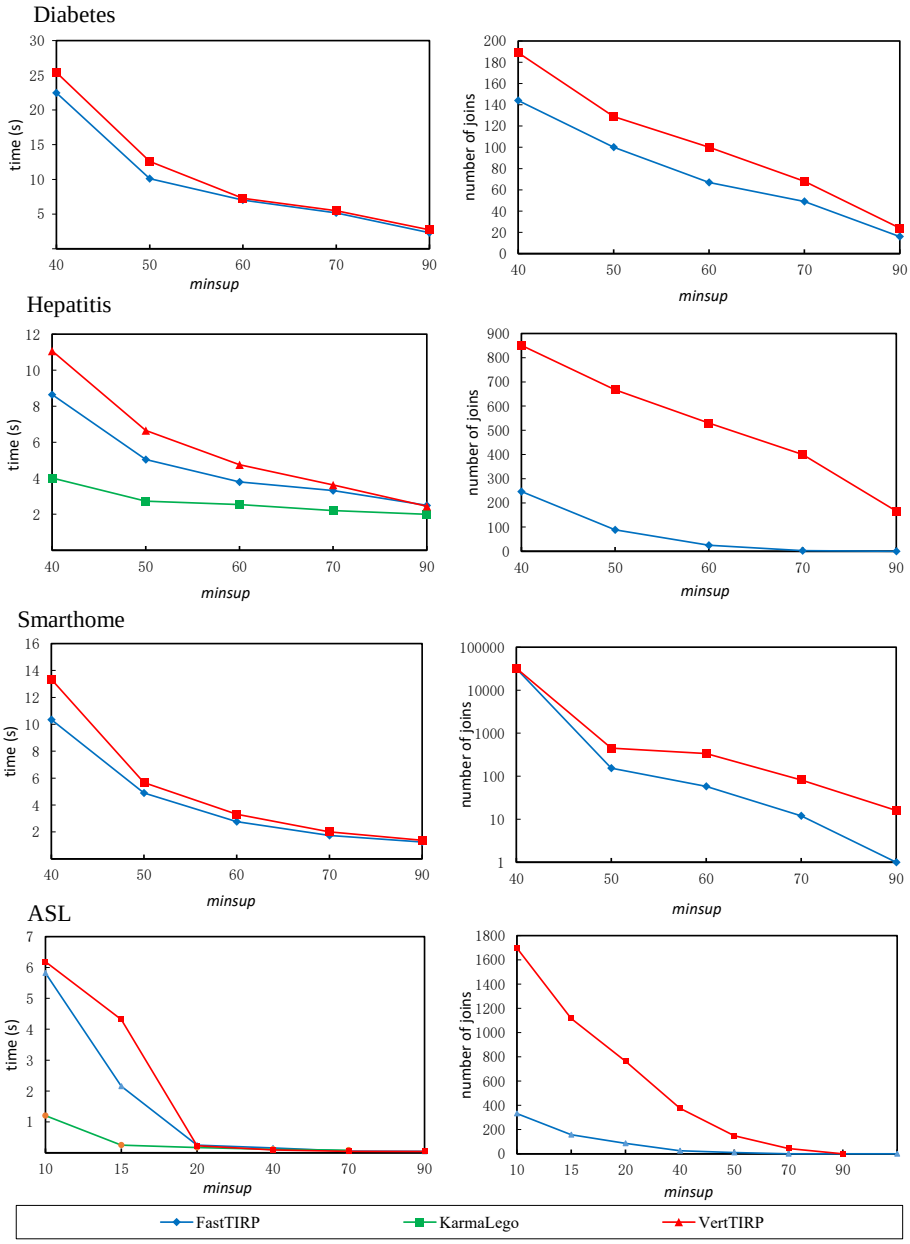

Fig. 3: Comparison of runtime and number of joins for different *minsup* values on four detests

bytes= 85, 264 bytes. In all cases, the size is quite small (less than a megabyte), which is acceptable.

Table 5: Comparison of the memory usage on the four datasets

Dataset	Algorithm	Avg. memory usage (MB)	Max. memory usage (MB)
Diabetes	FastTIRP	559	687
	VertTIRP	538	663
	KarmaLego	280	284
Hepatitis	FastTIRP	283	368
	VertTIRP	280	364
Smarthome	FastTIRP	437	634
	VertTIRP	437	630
ASL	FastTIRP	213	216
	VertTIRP	212	215
	KarmaLego	29	35

For datasets with a very large number of event types, it may be worthwhile to implement the PSM as a triangular matrix or sparse matrix to save some memory. As a triangular matrix, the size of the PSM is $SizeTriangularMatrix = \frac{|E|\times(|E|-1)}{2} \times SizeOfValue$, and the PSM requires the following amount of memory for each dataset: Diabetes: $\frac{35\times34}{2} \times 4$ bytes= 2380 bytes, Hepatitis: $\frac{63\times62}{2} \times 4$ bytes= $7,812$ bytes, Smarthome: $\frac{95\times94}{2} \times 4$ bytes= $17,860$ bytes, and ASL: $\frac{146\times145}{2} \times 4$ bytes= $42,340$ bytes. This is the implementation used in experiments.

5 Conclusion

This paper has presented a novel algorithm for mining TIRPs in a time-interval sequence database, called FastTIRP. It utilizes the same basic search procedure as VertTIRP but applies a Pair Support Pruning (PSP) technique to reduce the number of join operations. Experiments show that FastTIRP outperforms the state-of-the-art VertTIRP algorithm in terms of runtime on several benchmark datasets and that the number of join operations can be greatly reduced.

In future work, we will consider designing other optimizations and algorithms for this problem, as well as a distributed version of FastTIRP to run on a big data framework.

Java source code of VertTIRP and FastTIRP will be released in the next version of the SPMF data mining library at `http://www.philippe-fournier-viger.com/spmf/` [2].

References

1. Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R.: Fast vertical mining of sequential patterns using co-occurrence information. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 40–52. Springer (2014)
2. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The SPMF open-source data mining library version 2. In: Proc. 20th European conference on machine learning and knowledge discovery in databases. pp. 36–40. Springer (2016)

3. Fournier-Viger, P., Lin, J.C.W., Kiran, U.R., Koh, Y.S.: A survey of sequential pattern mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
4. Fournier-Viger, P., Yang, Y., Yang, P., Lin, J.C.W., Yun, U.: TKE: Mining top-k frequent episodes. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. pp. 832–845. Springer (2020)
5. Huang, J.W., Jaysawal, B.P., Chen, K.Y., Wu, Y.B.: Mining frequent and top-k high utility time interval-based events with duration patterns. Knowl. Inf. Syst. **61**(3), 1331–1359 (2022)
6. Iwanuma, K., Takano, Y., Nabeshima, H.: On anti-monotone frequency measures for extracting sequential patterns from a single very-long data sequence. In: Conference on Cybernetics and Intelligent Systems. vol. 1, pp. 213–217 (2004)
7. Jiang, C., Coenen, F., Zito, M.: A survey of frequent subgraph mining algorithms. Knowledge Engineering Review **28**, 75–105 (2013)
8. Le, H.H., Kushima, M., Araki, K., Yokota, H.: Differentially private sequential pattern mining considering time interval for electronic medical record systems. International Database Applications & Engineering Symposium pp. 13:1–13:9 (2019)
9. Lee, Z., Lindgren, T., Papapetrou, P.: Z-miner: An efficient method for mining frequent arrangements of event intervals. In: SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 524–534. ACM (2020)
10. Liao, G., Yang, X., Xie, S., Yu, P.S., Wan, C.: Mining weighted frequent closed episodes over multiple sequences. Tehnički vjesnik **25**(2), 510–518 (2018)
11. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovering frequent episodes in sequences. In: International Conference on Knowledge Discovery and Data Mining. pp. 210–215. AAAI Press (1995)
12. Mordvanyuk, N., López, B., Bifet, A.: verttirp: Robust and efficient vertical frequent time interval-related pattern mining. Expert Syst. Appl. **168**, 114276 (2021)
13. Moskovitch, R., Shahar, Y.: Classification of multivariate time series via temporal abstraction and time intervals mining. Knowl. Inf. Syst. **45**(1), 35–74 (2015)
14. Wu, S.Y., Chen, Y.L.: Mining nonambiguous temporal patterns for interval-based events. IEEE Transactions on Knowledge and Data Engineering **19**(6), 742–758 (2007)
15. Zheng, Z., Wei, W., Liu, C., Cao, W., Cao, L., Bhatia, M.: An effective contrast sequential pattern mining approach to taxpayer behavior analysis. World Wide Web **19**(4), 633–651 (2016)
16. Zhou, W., Liu, H., Cheng, H.: Mining closed episodes from event sequences efficiently. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 310–318. Springer (2010)