



TIRPC1₀: efficient and complete mining of time intervals-related patterns

Omer Harel¹ · Robert Moskovitch^{1,2}

Received: 24 June 2022 / Accepted: 23 May 2023 / Published online: 30 June 2023

© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2023

Abstract

Mining frequent *Time Intervals-Related Patterns (TIRPs)* from series of *symbolic time intervals* offers a comprehensive framework for heterogeneous, multivariate temporal data analysis in various application domains. While gaining a growing interest in recent decades, the efficient mining of frequent TIRPs is still a high computational challenge which has also not yet been investigated in its full complexity. The majority of previous methods discover only the first instances of the TIRPs within each series of symbolic time intervals, whereas their re-occurring instances are ignored. This eventually results in an *incomplete* discovery of frequent TIRPs, a problem that lies also in the challenge of mining only the frequent *closed TIRPs*, which was only recently investigated for the first time. In this paper, we introduce TIRPC1₀—an efficient algorithm for the complete mining of either the entire set of frequent TIRPs, or only the frequent closed TIRPs. The algorithm proposes a non-ambiguous sequential representation of symbolic time intervals series through the intervals' end-points, as well as a memory-efficient index and a novel method for data projection, due to which it is the first algorithm to guarantee a complete discovery of frequent closed TIRPs. The experimental evaluation conducted on eleven real-world and four synthetic datasets demonstrates that TIRPC1₀ is up to 10 times faster when mining the entire set of frequent TIRPs, and up to more than 100 times faster when mining only the frequent closed TIRPs compared to four state-of-the-art methods, while also reporting lower memory measurements.

Keywords Time interval mining · Closed temporal pattern · Temporal knowledge discovery · Frequent pattern mining

Responsible editor: Michelangelo Ceci.

Extended author information available on the last page of the article

1 Introduction

Along the recent two decades, there has been a growing interest in temporal knowledge discovery through frequent temporal patterns, and particularly through the discovery of frequent *Time Intervals-Related Patterns (TIRPs)* from series of *symbolic time intervals (STIs)* data. STIs may be raw, i.e., describing events which have a non-zero time duration, such as a time period a patient is prescribed on a medication or the period of time the green light is on in a traffic light. Alternatively, STIs can be created from raw time-points series after employing *temporal abstraction* (Shahar 1997; Lavrac et al. 2000; Höppner 2002; Lin et al. 2003; Mörchen and Ultsch 2005; Moskovitch 2022; Moskovitch and Shahar 2015a; Mordvanyuk et al. 2022). Through the process of temporal abstraction various forms of temporal variables, whether sampled regularly or irregularly, are transformed into a uniform representation as series of STIs, as will be elaborated in Sect. 2. Thus, enabling the discovery of frequent TIRPs from multivariate heterogeneous temporal data, after being abstracted into STIs series representation.

Due to the constant growth not only in the availability of temporal data but also in the data heterogeneity, mining frequent TIRPs has become highly relevant in a large variety of real-world applications. That is, either by directly using the discovered TIRPs for knowledge discovery (Sacchi et al. 2007), or utilizing them for a wide range of downstream tasks such as STIs series classification or clustering (Patel et al. 2008; Batal et al. 2009; Moskovitch et al. 2009; Moskovitch and Shahar 2015b; Rebane et al. 2021; Shknevsky et al. 2021), as well as outcome prediction (Moskovitch et al. 2015; Itzhak et al. 2023; Novitski et al. 2020). In dynamic graphs, for example, frequent STIs series mining was also used for the discovery of frequent temporal patterns of edge-interactions (Kostakis and Gionis 2017). In addition, as will be explained in Sect. 2, TIRPs are explicitly represented by the temporal relations among their STIs. Therefore, they can be easily interpreted by domain experts, which makes time interval mining a very attractive technique for explainable temporal data analytics in real-life data.

The discovery of frequent *closed TIRPs*, in particular, has a significant potential advantage over the discovery of the entire set of frequent TIRPs. While a single frequent closed TIRP may contain an exponential number (in the size of the TIRP) of frequent sub-TIRPs that are all not closed, their entire information is contained within the closed TIRP itself. Thus, mining frequent closed TIRPs potentially produces a much more compact output of frequent TIRPs, which contains the complete information of all the frequent TIRPs in the underlying data. However, the detection and pruning of unclosed TIRPs during runtime is a quite expensive task, which is sometimes slower than mining the entire set of frequent TIRPs. In this paper's evaluation, we empirically exemplify just this trade-off.

Several methods were proposed for the discovery of frequent TIRPs from a dataset of STIs series. However, the complete discovery of frequent TIRPs was not addressed properly until recently in (Moskovitch and Shahar 2015c; Lee et al. 2020; Mordvanyuk et al. 2022). Earlier methods (Winarko and Roddick 2007; Wu and Chen 2007; Patel et al. 2008; Papapetrou et al. 2009; Chen et al. 2015) mostly

discovered only the first instance of a TIRP within each series of STIs. The following instances of the TIRP, on the other hand, had been ignored, which eventually resulted in an incomplete discovery of frequent TIRPs, as will be explained in detail in Sect. 2.4. The same completeness problem lies also in the related task of frequent closed TIRP mining, which has only been investigated for the first time in (Chen et al. 2016). In this paper we demonstrate the said completeness problem in each of the two TIRP mining tasks and introduce a novel algorithm which is complete. The algorithm also keeps track of *the complete set of specific instances* of the frequent TIRPs that it discovers.

The main contributions of the paper are the following:

- *Novelty* We introduce `TIRPClO`—An efficient algorithm for the complete mining of either the entire set of frequent TIRPs, or only the frequent closed TIRPs. The algorithm’s main technical novelty includes (1) a novel method for STIs series transformation into sequential representation, avoiding ambiguity; (2) a complete method for data projection which aims to overcome the main challenge of current projection mechanisms, i.e., the detection of the re-occurring instances of TIRPs within a single series of STIs; and (3) a closure-checking scheme for the detection and pruning of TIRPs that are not closed early during the mining process, when closed TIRPs discovery is desired.
- *Completeness* Theoretical and empirical analysis of the completeness problem in the previous TIRP mining methods is provided, and proofs for `TIRPClO`’s completeness are supplied. To the best of our knowledge, `TIRPClO` is the first complete algorithm for frequent closed TIRP mining, and the first sequence-based complete algorithm for mining the entire set of frequent TIRPs.
- *Performance* We conducted a rigorous runtime and memory consumption evaluation of `TIRPClO` compared to four state-of-the-art methods, including: `KarmaLego` (Moskovitch and Shahar 2015c), `CCMiner` (Chen et al. 2016), `ZMiner` (Lee et al. 2020), and `VertTIRP` (Mordvanyuk et al. 2021), on a wide benchmark of eleven real-world datasets as well as several novel synthetic datasets, demonstrating
 - 2–18 times faster runtimes compared to `KarmaLego` and `ZMiner`, and 1.5–10 times faster runtimes compared to `VertTIRP` when mining the entire set of frequent TIRPs, while also reporting up to ten times lower memory measurements.
 - One to more than two orders of a magnitude shorter runtimes compared to `CCMiner` when mining only the frequent closed TIRPs, with competitive memory requirements.
- *Code and Data Availability* The code of the `TIRPClO` algorithm, as well as all the evaluation datasets and our synthetic datasets generator are publicly available.¹

The rest of the paper is organized as follows: Sect. 2 reviews the related work and lays the foundations for the discussion on completeness in frequent TIRP mining.

¹ <https://github.com/TIRPClO/Complete-Time-Intervals-Related-Patterns-Mining>.

Section 3 introduces the proposed TIRPCLO algorithm, and Sect. 4 details the experimental setup which has been designed for TIRPCLO's performance evaluation. Finally, Sect. 5 reports the experimental results and Sect. 6 concludes the paper and discusses future research directions.

2 Background

While this paper deals with the discovery of frequent TIRPs from symbolic time intervals data, we start with a concise subsection about the discovery of sequential patterns, from time-points based sequential data. That is since the time interval mining algorithm that we introduce here consists of sequential representation and thus, principles from sequential pattern mining are relevant.

2.1 Sequential mining

Sequential pattern mining is a problem which has been investigated more intensely in past years and refers to the discovery of the frequent sequential patterns within a sequences database (Fournier-Viger et al. 2017). According to (Zhao and Bhowmick 2003; Mabroukeh and Ezeife 2010), sequential pattern mining algorithms mainly differ in two aspects: candidate sequences' generation and support counting techniques. Based on these criteria, sequential pattern mining methods can be broadly divided into two approaches: (1) Apriori-based algorithms, such as GSP (Srikant and Agrawal 1996), SPADE (Zaki 2001) and SPAM (Ayres et al. 2002); and (2) Pattern-growth algorithms, which focus the search on a restricted portion of the initial database recursively. Among these algorithms are FreeSpan (Han et al. 2000), PrefixSpan (Han et al. 2001) and LAPIN (Yang et al. 2007), which are projection-based; and the WAP-mine (Pei et al. 2000) and PLWAP (Ezeife et al. 2005) methods that use a tree structure-mining technique.

Few of these methods, in fact, have been further extended to address the problem of symbolic time interval mining, on which we focus in this paper. An example is the PrefixSpan algorithm (Han et al. 2001) which has been extended in (Wu and Chen 2007) to form the TIRP mining algorithm TPrefixSpan. There are also quite few methods that deal with the problem of mining only the frequent *closed* sequential patterns (Yan et al. 2003; Wang and Han 2004; Tzvetkov et al. 2005; Huang et al. 2006; Chang et al. 2008; Gomariz et al. 2013; Zhang et al. 2015; Fumarola et al. 2016), on which we will elaborate later in the Closed TIRPs Sect. 2.3.3.

2.2 Symbolic time intervals data

Temporal data include not only time-stamped raw data or time-points series, but also time intervals, which are events having a type and a non-zero time duration. Such time intervals are referred to as *symbolic time intervals (STIs)*.

Definition 1 (*STI*) A *symbolic time interval* $I = (\text{symbol}, s, f)$ is a triplet of a symbol (i.e., the event type), a start-time, and a finish-time.

Definition 2 (*Lexicographical STIs Series*) A lexicographical STIs series (I_1, I_2, \dots, I_k) is a sorted series of STIs, such that $\forall I_i, I_j: i < j \equiv I_{i,s} < I_{j,s} \vee (I_{i,s} = I_{j,s} \wedge I_{i,f} < I_{j,f}) \vee (I_{i,s} = I_{j,s} \wedge I_{i,f} = I_{j,f} \wedge I_i.\text{symbol} < I_j.\text{symbol})$.

While STIs may indeed be raw, describing events that have a non-zero time duration, they can be also created from time-points series after employing *temporal abstraction*. Temporal abstraction refers to the segmentation and aggregation of a series of raw, timestamped, multivariate temporal data into a uniform representation as a series of STIs. Quite few methods have been proposed for temporal abstraction (Shahar 1997; Lavrac et al. 2000; Höppner 2002; Lin et al. 2003; Mörchen and Ullsch 2005; Azulay et al. 2007; Moskovitch and Shahar 2015a; Mordvanyuk et al. 2022). Most of them are based on either *state* or *gradient* abstraction (Moskovitch and Shahar 2015b). After state abstraction, STIs represent periods of time during which a variable is in a specific state (defined by cutoffs), or after gradient abstraction—a segment of an increasing or decreasing period of the values according to the first derivative. When temporal data are given as time series or data streams, temporal abstraction may be applied as a preliminary stage prior to TIRP mining.

2.3 Frequent TIRP mining

Frequent TIRPs are mined from a dataset of STIs series, in which each series of STIs is associated with an entity (e.g., patient) that has a unique identifier, i.e., the entity ID. While the entire set of methods that we review in this subsection focus on the discovery of frequent TIRPs from STIs data, not all of them maintain the time intervals-based representation. Looking at the methods that were published, it is clear that two types of approaches have been developed. Several methods are time intervals-based (Winarko and Roddick 2007; Patel et al. 2008; Papapetrou et al. 2009; Moskovitch and Shahar 2015c; Sharma and Patel 2018; Lee et al. 2020), which means that they directly operate the time intervals-based representation. Others are sequence-based (Wu and Chen 2007; Chen et al. 2015, 2016), which refer to the end-points of the STIs as time-points sequences, to which sequential mining style algorithms are applied in order to mine the frequent TIRPs. In both approaches, the most commonly used method to define the temporal relations among the STIs is based on Allen's temporal relations (Allen 1983)—either explicitly in time-intervals based methods, or implicitly in sequence-based methods.

Allen (1983) formulated a finite set of 13 temporal relations between a pair of STIs. The set includes *before*, *meet*, *overlap*, *start*, *contain*, *finished-by*; their corresponding inverse relations: *after*, *met-by*, *overlapped-by*, *started-by*, *during*, *finish*; and *equal*. Allen's temporal relations can be referred as if they include seven basic relations, six of which have an inverse relation, while *equal* is its own

inverse. When the STIs are lexicographically ordered (Definition 2) it is sufficient to use the seven relations, without their inverse relations. Figure 1 shows Allen's seven temporal relations, their inverse relations, and the sequence definition of each relation based on the end-times of the STIs.

Definition 3 (TIRP) A *Time Intervals-Related Pattern (TIRP)* is defined as $T = \{T_{Intervals}, T_{Relations}\}$ where $T_{Intervals} = (I_1, I_2, \dots, I_k)$ is a lexicographically ordered set of k STIs, and $T_{Relations} = \bigwedge_{i=1}^k \bigwedge_{j=i+1}^k \text{AllenRelation}(I_i, I_j)$ defines the conjunction of Allen's temporal relations among each of the $\binom{k}{2}$ pairs of STIs within $T_{Intervals}$.

Note that the timestamps of the STIs' end-points are, of course, not part of the TIRP definition, but only their symbols and the temporal relations among them. Otherwise, frequent TIRPs would be hardly discovered.

Definition 4 (Vertical Support) The *vertical support* of a TIRP T is the number of entities' STIs series $|E_T|$ in which T appears at least once, divided by the total number of entities in the dataset $|E|$. Therefore, $\text{vertical support}(T) = \frac{|E_T|}{|E|}$.

Example 1 In Fig. 2, both of the TIRPs $T_1 = \langle A \text{ overlaps } B \rangle$ and $T_2 = \langle A \text{ overlaps } B \wedge B \text{ overlaps } C \wedge A \text{ before } C \rangle$ appear in each of the two STIs series (a) and (b). Therefore, assuming a dataset of only these two STIs series shown in Fig. 2,

$$\forall i \in \{1, 2\} : \text{vertical support}(T_i) = \frac{|E_{T_i}|}{|E|} = 1.0.$$

Definition 5 (Frequent TIRP Mining Task) Given a dataset of $|E|$ entities' STIs series, the goal of the *frequent TIRP mining task* is to discover the entire set of frequent TIRPs, with respect to a given minimum vertical support threshold (including each of the TIRPs' instances within each entity in the dataset, for the sake of completeness, as will be demonstrated in Sect. 2.4).

2.3.1 Time intervals-based TIRP mining

Time intervals-based TIRP mining methods directly discover frequent TIRPs that are composed of STIs and the conjunction of Allen's temporal relations among them, as defined in Definition 3. Most methods, especially in early studies, used a subset of Allen's temporal relations. For example, Villafane et al. (2000), in which containments of STIs within a multivariate STIs series were discovered. The first to use the entire set of Allen's temporal relations were Kam and Fu (2000). However, since they have not defined the temporal relations among the patterns' components that are not successive, these patterns have been ambiguous. The first to define a





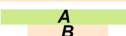


Allen Relation	Example	Sequence Definition	Inverse Relation
<i>A before B</i>		$A.s < A.f < B.s < B.f$	<i>B after A</i>
<i>A meets B</i>		$A.s < A.f = B.s < B.f$	<i>B met-by A</i>
<i>A overlaps B</i>		$A.s < B.s < A.f < B.f$	<i>B overlapped-by A</i>
<i>A starts B</i>		$A.s = B.s < A.f < B.f$	<i>B started-by A</i>
<i>A contains B</i>		$A.s < B.s < B.f < A.f$	<i>B during A</i>
<i>A finished-by B</i>		$A.s < B.s < A.f = B.f$	<i>B finishes A</i>
<i>A equals B</i>		$A.s = B.s < A.f = B.f$	<i>B equals A</i>

Fig. 1 Allen’s 13 temporal relations between a pair of STIs

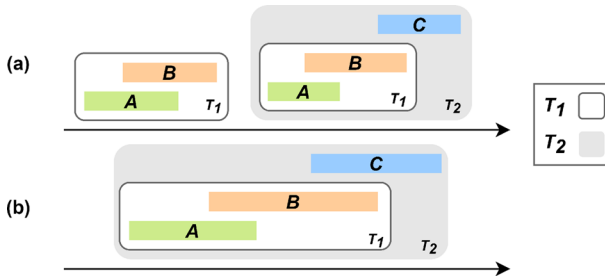


Fig. 2 Two STIs series which include the TIRPs $T_1 = \langle A \text{ overlaps } B \rangle$ and $T_2 = \langle A \text{ overlaps } B \wedge B \text{ overlaps } C \wedge A \text{ before } C \rangle$. While T_1 appears twice in (a) and once in (b), T_2 appears only once in each of the two STIs series

non-ambiguous representation of TIRPs based on Allen’s relations was Höppner (2001), using a k^2 matrix to represent all of the pairwise relations within a k -STIs TIRP.

In Winarko and Roddick (2007), the ARMADA algorithm was introduced as an extension of the sequential pattern mining algorithm MEMISP (Lin and Lee 2002). Papapetrou et al. (2009) proposed two approaches for the generation of the discovered TIRPs tree—using either a breadth first search (BFS) or a greedy depth first search (DFS), and also a hybrid approach (H-DFS) combining the two methods inspired by the SPAM sequential mining method (Ayres et al. 2002). Patel et al. (2008) introduced IEMiner, improving the performance over Papapetrou et al. (2009) by directly extending the TIRPs during the discovery process. The KarmaLego algorithm (Moskovitch and Shahar 2015c) introduced a novel direct extension approach, employing the transitivity property of Allen’s temporal relations for a more efficient candidate generation. Since KarmaLego’s index is quite not scalable, in Moskovitch et al. (2015) an improvement on KarmaLego was proposed, using a more memory-efficient hash-based index.

Sharma and Patel (2018) introduced the STIPA algorithm as a memory efficient extension of ARMADA (Winarko and Roddick 2007), shrinking its index to fit in devices with strong memory requirements. In Lee et al. (2020), the ZMiner algorithm was proposed. The algorithm utilizes a hierarchical lookup hash structure

which is used to index the frequent two-sized TIRPs, similar to H-DFS (Papapetrou et al. 2009) and KarmaLego (Moskovitch and Shahar 2015c). However, in ZMiner candidates for the extension of a current pattern are generated based on the indexed pairwise temporal relations, and without employing the transitivity property, as opposed to KarmaLego. ZMiner also stores longer, discovered TIRPs within an additional data structure, to enhance the recursive frequent TIRP mining process. In Mordvanyuk et al. (2021) the VertTIRP algorithm was introduced. To accelerate the candidate generation process, VertTIRP uses a pairing strategy which sorts the temporal relations to be assessed, beyond just utilizing the transitivity property as made in KarmaLego.

2.3.2 Sequence-based TIRP mining

Sequence-based TIRP mining methods intend to take advantage of the advancements made already in sequential mining algorithms. For that, the input STIs series data are first transformed into a sequential representation of the STIs' start and finish end-points. In this paper, the start and finish end-points of an STI which has the symbol A are denoted by A^+ and A^- respectively. Then, typically a sequential mining-based method is applied to the database of end-points sequences in order to discover the frequent TIRPs, which are thus represented as frequent sequences of their STIs' end-points. In that respect, it is important to note that such a discovered frequent end-points sequence does not correspond to a valid TIRP if it contains only one of the two end-points (either start or finish) of some STI.

Allen's temporal relations among the TIRPs' STIs are implicitly represented by the sequential order of their end-points. For that, a non-ambiguous sequential representation is essential. Ambiguity means either (1) having different representations for the same temporal relation, or (2) having one representation which expresses different temporal relations. Note that due to sequentially representing the whole STIs series, time flexibility extensions over Allen's relations, which have been addressed in several time intervals-based methods (Moskovitch and Shahar 2015c; Lee et al. 2020; Mordvanyuk et al. 2021), are not addressed in sequence-based methods.

In Wu and Chen (2007) introduced the sequence-based TPrefiXSpan algorithm as an extension of the PrefixSpan sequential mining method (Han et al. 2001). However, due to representation, the method does not prevent the generation of multiple candidates of the same pattern, as described in Moskovitch and Shahar (2015c). Inspired by PrefixSpan, Chen et al. (2015) proposed TPMiner and P-TPMiner using an extended sequential representation, adding the STIs' end-times to the end-point based TIRPs representation used in PrefixSpan. An incremental version of TPMiner was also presented later in Hui et al. (2016). In Chen et al. (2016) the CCMiner algorithm was introduced. CCMiner is inspired by the BIDE method for closed sequential pattern mining (Wang and Han 2004), and is the only method proposed in the literature for the discovery of only the frequent closed TIRPs. While we elaborate more on CCMiner in the next subsection on closed TIRP mining, here we focus on the sequential representation they introduced.

As a sequence-based TIRP mining method, CCMiner transforms the STIs series input into a string representation of the STIs' end-points. However, looking carefully

into the proposed representation, it is ambiguous due to not uniquely representing Allen’s *overlap* temporal relation. Figure 3 illustrates an ambiguous scenario which results from the current representation. It presents two series of STIs (a) and (b), and their corresponding representation in the current format which are shown on the right. First, the STIs’ end-points are chronologically ordered. When a start end-point is followed by a finish end-point (of another STI) and there is no other end-point between them, they are put within brackets (as happens in sequential mining with events that happen within the same time), as shown at the top of both series’ representations. Then, the symbols within brackets are ordered alphabetically, as shown at the bottom.

Looking at the representation of the STIs A and B , in (a) the sequential order of their end-points is $A^+, B^+, A^-,$ and B^- ; while in (b) their order is $A^+, A^-, B^+,$ and B^- . That is despite Allen’s temporal relation among them is *overlap* in both series, which means that there are at least two different representations of the same temporal relation in the described representation. This stands for an ambiguous scenario, due to which we will not be able to know for sure what series contains the TIRP $\langle A \text{ overlaps } B \rangle$. The difference is due to the STI C , whose start time in (a) is between the start time of B and the finish time of A , unlike in (b). Hence, in (a) B^+ and A^- are separated, while in (b) they are grouped together within brackets. This is just a single example of the ambiguity, but there are more obviously. In this paper, we propose a novel non-ambiguous transformation method from an input series of STIs into sequential representation, which overcomes the representation challenges described in this subsection.

2.3.3 Closed TIRP mining

Definition 6 (Super-pattern) In sequential representation, given two sequential patterns $p_1 = \langle t_1, t_2, \dots, t_k \rangle$ and $p_2 = \langle t'_1, t'_2, \dots, t'_n \rangle$, p_2 is a *super-pattern* of p_1 if and only if there exist indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$ such that $t_1 = t'_{i_1}, t_2 = t'_{i_2}, \dots, t_k = t'_{i_k}$.

Definition 7 (Super-TIRP) Let T_1 and T_2 be two TIRPs. Then T_2 is a *super-TIRP* of T_1 if and only if 1) $T_{1Intervals} \subseteq T_{2Intervals}$ and 2) $\forall I_i < I_j \in T_{1Intervals}: T_{1Relations}(I_i, I_j) = T_{2Relations}(I_i, I_j)$.

Definition 8 (Closed TIRP) A TIRP T_1 is a *closed TIRP* if and only if there is no super-TIRP T_2 of T_1 , which has the same vertical support.

Example 2 In Fig. 2, the TIRP $T_2 = \langle A \text{ overlaps } B \wedge B \text{ overlaps } C \wedge A \text{ before } C \rangle$ is a super-TIRP of the TIRP $T_1 = \langle A \text{ overlaps } B \rangle$. That is since 1) $T_{1Intervals} = (A, B) \subseteq (A, B, C) = T_{2Intervals}$, and 2) $T_{1Relations}(A, B) = T_{2Relations}(A, B) = \text{overlap}$. In addition, T_1 and T_2 have the same vertical support of 1.0, as they appear in each of the two STIs series (a) and (b) in Fig. 2. Therefore, according to Definition 8, T_1 is not a closed TIRP, while its super-TIRP T_2 is indeed closed.

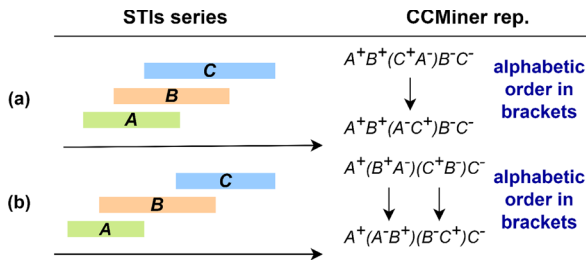


Fig. 3 Two STIs series which include the TIRP $\langle A \text{ overlaps } B \rangle$, and their corresponding sequential representation in CCMiner (Chen et al. 2016). While the sequential order of the end-points of the TIRP's STIs is A^+ , B^+ , A^- , and B^- in (a), their sequential order in (b) is A^+ , A^- , B^+ , and B^- ; which stands for an ambiguous scenario in which there are at least two different representations of the same temporal relation in the described representation

Definition 9 (*Frequent Closed TIRP Mining Task*) Given a dataset of $|E|$ entities' STIs series, the goal of the *frequent closed TIRP mining task* is to discover the set of frequent closed TIRPs, with respect to a given minimum vertical support threshold (including each of the closed TIRPs' instances within each entity in the dataset, for the sake of completeness, as will be demonstrated in Sect. 2.4).

The discovery of frequent closed temporal patterns has recently gained significant interest. That is mainly due to the fact that the set of frequent closed patterns contains the entire information of all the frequent patterns in the underlying data, although it is potentially much more compact. Therefore, mining only the frequent closed temporal patterns typically shrinks the output of frequent patterns, without any loss of information. Prior research on closed temporal pattern mining has mainly focused on sequential data (Yan et al. 2003; Wang and Han 2004; Tzvetkov et al. 2005; Huang et al. 2006; Chang et al. 2008; Gomariz et al. 2013; Zhang et al. 2015; Fumarola et al. 2016). The CCMiner algorithm (Chen et al. 2016), however, is the first and only method proposed so far for the discovery of frequent closed temporal patterns from STIs-based data (i.e., closed TIRPs). The algorithm is inspired by the BIDE method (Wang and Han 2004) originally used for mining closed sequential patterns.

Finally, there are, of course, several other interesting approaches for time interval mining, e.g., high-utility pattern mining (Huang et al. 2019; Mirbagheri and Hamilton 2020a, 2021), sub-sequence searching within series of STIs (Yang et al. 2017), and similarity matching of multiple STIs series (Kostakis et al. 2011; Kotsifakos et al. 2013; Mirbagheri and Hamilton 2020b) to name a few. However, these approaches are not part of the core topics and methods covered in this paper, which focus on the two specific tasks of mining either the entire set of frequent TIRPs (Definition 5), or only the frequent closed TIRPs (Definition 9).

2.4 Completeness and horizontal support

Definition 10 (*Horizontal Support*) The *horizontal support* of a TIRP T within an entity e is the number of instances of T within e 's STIs series.

Example 3 In Fig. 2, the TIRP T_1 appears twice within the series (a) and once within (b). Thus, $\text{horizontal support}(T_1, (a)) = 2$, while $\text{horizontal support}(T_1, (b)) = 1$.

Completeness in frequent TIRP mining means to discover all the frequent TIRPs in the underlying data, with respect to the predefined minimum vertical support threshold, which is mandatory for compliance with the TIRP mining problem definition (Definition 5). In this subsection we analyze and demonstrate a completeness problem in the majority of previous methods for either the discovery of the entire set of frequent TIRPs, or only the frequent closed TIRPs. That is due to not entirely discovering the TIRPs' horizontal support (Definition 10). A supplementary empirical analysis which substantiates the main claims stated in this subsection and exemplifies them on real-world data will be provided in experiment 3.

2.4.1 Completeness in the entire frequent TIRP mining

The majority of previous TIRP mining methods—either time intervals-based (Winarko and Roddick 2007; Patel et al. 2008; Papapetrou et al. 2009) or sequence-based (Wu and Chen 2007; Chen et al. 2015), intended to discover only the first TIRP instance within each entity's STIs series, and not all the horizontally supporting instances of the TIRPs. That is probably due to meaningful complexity and computational requirements. However, in this subsection we show that in order to correctly count the TIRPs' vertical support values, their horizontal support discovery is essential. Otherwise, when intending to discover only the first instance of a TIRP within its supporting entities (i.e., the entities in which the TIRP appears), TIRPs' vertical support values are potentially undercounted. Consequentially, TIRPs that are indeed frequent, i.e., their true vertical support value is above the minimum vertical support threshold, are wrongly considered infrequent. Thus, they are not discovered by the algorithm, which directly results in an *incomplete* discovery of frequent TIRPs according to Definition 5.

A common scenario which exemplifies the said completeness problem is illustrated in Fig. 2. Assume that only the first instance of a TIRP within each entity's STIs series is discovered, as made in Winarko and Roddick (2007), Wu and Chen (2007), Patel et al. (2008), Papapetrou et al. (2009), Chen et al. (2015). Then, in the series (a) only the first instance of T_1 is discovered, whereas its second instance is ignored, which is indeed sufficient for T_1 's vertical support counting. However, due to ignoring the second instance of T_1 within (a), the only instance of T_2 within this series is not discovered as well. That is since typically, in order to discover an instance of a TIRP, its prefix has to be discovered first, and then extended (e.g., an instance of the one-sized TIRP $\langle A \rangle$ should be discovered and extended in order to discover an instance of the two-sized TIRP $\langle A \text{ overlaps } B \rangle$).

As a result, T_2 's vertical support is undercounted, and it might be wrongly considered infrequent with respect to the predefined minimum vertical support threshold. In such case, the TIRP T_2 is not discovered despite it is indeed frequent, which results in an incomplete discovery of frequent TIRPs. Therefore, we conclude that *all the horizontally supporting instances of the TIRPs must be discovered to guarantee a complete discovery of frequent TIRPs* as defined in Definition 5. The crucial need in

the discovery of the horizontal support for completeness when mining the entire set of frequent TIRPs will be empirically demonstrated in experiment 3.

2.4.2 Completeness in frequent closed TIRP mining

As described in Sect. 2.3, CCMiner (Chen et al. 2016) is the first and only method proposed in the literature for the discovery of only the frequent closed TIRPs. The algorithm, similar to the majority of previous methods for the discovery of the entire set of frequent TIRPs, intends to discover at most a single TIRP instance within each entity's STIs series in the dataset. In the previous subsection it has already been demonstrated that the discovery of the TIRPs' horizontal support is essential for completeness when mining all the frequent TIRPs. In this subsection, we elaborate on the example from Fig. 2 to show that the same completeness problem lies in the discovery of only the frequent closed TIRPs as well.

Suppose that only the first instance of a TIRP is discovered within each entity's STIs series, as made in Chen et al. (2016). Then, as described in the previous subsection, the TIRP T_2 in Fig. 2 is not discovered within the series (a). Therefore, its vertical support is undercounted and T_2 might be wrongly considered infrequent, in which case it is not discovered. Since T_2 does not have any super-TIRP that maintains the same vertical support, it is a closed TIRP according to Definition 8. Thus, not discovering T_2 stands for an incomplete discovery of frequent closed TIRPs. Furthermore, according to Definition 8, T_1 is not a closed TIRP. That is due to its super-TIRP T_2 , which has the same vertical support as T_1 in the example dataset shown in Fig. 2. However, since the vertical support of T_2 is undercounted, T_1 might be wrongly considered a closed TIRP. Consequently, assuming that T_1 is frequent, it would be discovered by such a closed TIRP mining algorithm, although it is not a closed TIRP. Therefore, we conclude that *all the horizontally supporting instances of the TIRPs must be discovered to guarantee a complete discovery of frequent closed TIRPs* as defined in Definition 9. This conclusion will be empirically demonstrated in experiment 3 as well.

3 Methods

In this section we introduce TIRPCLo—a complete sequence-based TIRP mining algorithm, which is designed to discover either the entire set of frequent TIRPs, or only the frequent closed TIRPs. First, a couple of terms are introduced, which are necessary for the algorithm's description.

Definition 11 (*Tiep*) A *time-interval-end-point (tiep)* represents an STI's end-point through a pair of a symbol and an *end-type*, which can be either *start* or *finish*.

In this paper, the start-tiep and finish-tiep of an STI which has the symbol A are denoted by A^+ and A^- respectively. Since in real-life data multiple STIs having the same symbol can occur, we allow multiple instances of them within a

single entity's STIs series. For that, an index is added to differentiate the ties' instances within the same entity. For example, the i^{th} instance of an STI A is represented by A_i^+ for its start-tie and A_i^- for its finish-tie.

Definition 12 (*Complementing-tie*) Given a start-tie $t = A^+$, its *complementing tie* is A^- and vice versa. We denote a tie t 's complementing tie by t^C .

3.1 TIRPCLO

The main steps of TIRPCLO are outlined in Algorithm 1, which focuses on the discovery of the entire set of frequent TIRPs. For the complete discovery of only the frequent closed TIRPs, the algorithm also introduces a closure-checking scheme which detects and prunes TIRPs that are not closed early during the mining process and is described later in Sect. 3.5.

Since TIRPCLO is a sequence-based TIRP mining algorithm, it first transforms the input STIs series data into a novel ties-based sequences database representation (line 2). The frequent TIRPs, which are actually frequent sequences of the start-ties and finish-ties of the original STIs, are then discovered from the sequential representation. During the input transformation process, which is performed by the STIs2Seq method (Algorithm 3), the created ties are also indexed within the *ties-index*. The index enables the retrieval of the ordered instances of the ties within their supporting entities in a constant time, due to which no repeated scans of the data records are performed in TIRPCLO throughout the complete mining process. TIRPCLO's novel sequential representation of the STIs series, as well as the ties-index are described in great detail in the next subsection.

Algorithm 1 TIRPCLO

Input: db – lexicographically ordered STIs series dataset

Output: T – complete set of frequent TIRPs

```

1:  $T \leftarrow \emptyset$ 
2:  $sdb \leftarrow STIs2Seq(db)$ 
3: for  $t \in TiesIndex$  do
4:   if  $|TiesIndex[t].E_t| < minSup$  then
5:      $TiesIndex.remove(t)$ 
6:   end if
7: end for
8:  $FilterInfrequentTies(sdb)$ 
9: for  $t \in TiesIndex$  do
10:  if  $t.endType = START$  then
11:     $sdb_t \leftarrow ESProject(sdb, t, TiesIndex[t].E_t)$ 
12:     $ExtendTIRP(t, sdb_t, T, NULL)$ 
13:  end if
14: end for
15: return  $T$ 

```

Then (lines 3–7), the infrequent ties are filtered out of the index according to the Apriori-All principle. Thus, shrinking the size of the index as well as the search space of the recursive mining process. In addition, the infrequent ties are also filtered out of the initial sequences database, as an additional step of input data compaction (line 8). TIRPCLO uses a projection-based DFS approach for patterns growth, for which a novel, complete Entities Spawning projection method is introduced (Algorithm 4). For each frequent start-tie t , this method is applied to project the initial sequences database, referring to all the re-occurring instances of t for completeness (lines 9–11). Afterwards, all the frequent TIRPs that begin with t are recursively discovered through the ExtendTIRP method (Algorithm 2) in line 12. Finally, the complete set of frequent TIRPs is discovered and returned by the algorithm.

The ExtendTIRP method (Algorithm 2) extends a current frequent pattern p recursively. Note that as described in Sect. 2, the pattern p is actually a frequent mined ties sequence that cannot always be a valid TIRP. For that, all of its ties must appear paired with their complementing ties (either start or finish—Definition 12), which is verified in line 1. If p is a valid TIRP, its data, including the entire set of discovered instances, are added to the set of frequent TIRPs that are discovered by the algorithm (line 2). Then, to generate the candidate ties for the extension of p , the ties' support-indices are generated via Algorithm 5 (line 4). A support-index is a data structure that is created for each tie $cn d_t$ which is a valid candidate for the extension of p . That is, the extended pattern $p' = p + cn d_t$ can possibly be re-extended to form a valid TIRP. The conditions for the validity of a candidate tie are formally listed in the Candidate Generation Sect. 3.4. The support-index holds $cn d_t$'s current vertical support value, and points at its first instance within each record in the projected sequences database sdb_p .

Algorithm 2 ExtendTIRP

Input: p – current pattern
 sdb_p – projected sequences database of p
 T – set of discovered frequent TIRPs
 $prevSIs$ – previous support-indices

- 1: **if** $isValidTIRP(p)$ **then**
- 2: $T \leftarrow T \cup \{p\}$
- 3: **end if**
- 4: $cn dSIs \leftarrow GetSupportIndices(p, sdb_p, prevSIs)$
- 5: **for** $\langle cn d_t, cn d_tSI \rangle \in cn dSIs$ **do**
- 6: **if** $cn d_tSI.verticalSupport \geq minSup$ **then**
- 7: $p' = p + cn d_t$
- 8: $sdb_{p'} \leftarrow ESProject(sdb_p, cn d_t, cn d_tSI)$
- 9: $ExtendTIRP(p', sdb_{p'}, T, cn dSIs)$
- 10: **end if**
- 11: **end for**

For each such valid candidate tiep cmd_i whose support-index indicates it to be currently frequent, it is selected for the extension of p (lines 5–7). For that, the projected sequences database sdb_p is first re-projected with respect to cmd_i (line 8). Then, the `ExtendTIRP` method is repeatedly applied to further extend the pattern $p' = p + cmd_i$ (line 9). Note that the vertical support of p' is known beforehand and it equals to cmd_i 's current vertical support value, which is above the minimum vertical support threshold by selection. Therefore, all the extended patterns in `TIRPCLo` are guaranteed to be frequent, and `TIRPCLo` performs *the least pattern extension steps* that are required for the discovery of the complete set of frequent patterns based on only a single scan of the data.

3.2 TIRPCLo sequence-based TIRP representation and tieps-index

As a sequence-based TIRP mining algorithm, `TIRPCLo` first transforms the STIs series data into a sequences database representation, from which the frequent TIRPs are then discovered. For that, `TIRPCLo` introduces a novel tieps-based sequential representation of an entity's STIs series. The proposed representation is non-ambiguous, unlike `CCMiner`'s representation (Chen et al. 2016), and we also believe it is more intuitive in terms of the conversion to/from Allen's temporal relations compared to the other existing representations inspired by sequential mining employed in `TPrefixSpan` (Wu and Chen 2007) and `TPMiner` (Chen et al. 2015). But most importantly, in contrast to both `TPrefixSpan`, `TPMiner`, and `CCMiner`, that do not properly handle the re-occurrences of symbols and TIRPs within STIs series and are thus incomplete (Sect. 2.4); `TIRPCLo`'s representation, combined with our proposed projection method on which we elaborate in the next subsection, enable the discovery of all the horizontally supporting instances of the TIRPs, due to which it is the first complete sequence-based TIRP mining method.

Definition 13 (*ct-group*) A *coinciding-tieps group* (*ct-group*) is a triplet $CT = \langle ts, et, (t_1, \dots, t_n) \rangle$, where ts is a timestamp, et is an *end-type* (i.e., *start* or *finish*), and (t_1, \dots, t_n) is a set of coinciding-tieps that occur at the timestamp ts and have the end-type et . The set is alphabetically ordered according to the tieps' symbols, i.e., $\forall 1 \leq i < j \leq n : t_i.symbol < t_j.symbol$.

Definition 14 (*ct-sequence*) A *coinciding-tieps sequence* (*ct-sequence*) $CTSeq = \langle \langle CT_1, isMet_1 \rangle, \dots, \langle CT_k, isMet_k \rangle \rangle$ is a sequence of pairs for which: 1) $\forall 1 \leq i \leq k : CT_i$ is a *ct-group*, 2) $\forall 1 \leq i < j \leq k : CT_i.ts < CT_j.ts \vee (CT_i.ts = CT_j.ts \wedge CT_i.et = finish \wedge CT_j.et = start)$, which defines the order of the *ct-groups* within the sequence; and 3) $\forall 1 \leq i \leq k : isMet_i \equiv 1 < i \wedge CT_i.ts = CT_{i-1}.ts$, which indicates whether the STIs that start with the tieps of CT_i are *met-by* (Allen relations, Fig. 1) the STIs that end with the tieps of CT_{i-1} , or not.

In this paper, a *ct-group* CT is denoted by its set of coinciding-tieps (t_1, \dots, t_n) . For simplicity, brackets are omitted if $n = 1$. In addition, within a *ct-sequence*, a pair $\langle CT, isMet \rangle$, would be denoted by $^M(t_1, \dots, t_n)$ if $isMet = true$, and (t_1, \dots, t_n)

otherwise. As will be elaborated later in this subsection, the *isMet* flag enables to distinguish between the *before* and *meet* temporal relations within a *ct*-sequence, which is crucial for the non-ambiguity of TIRPClo's representation.

TIRPClo's data transformation method—STIs2Seq, transforms an entity's lexicographically ordered series of STIs (Definition 2) into a sequential representation as a *ct*-sequence (Definition 14), and is described in Algorithm 3. First, each STI is broken into its start-tiep and finish-tiep (line 1). Then, the tieps are partitioned to *ct*-groups (Definition 13), that are ordered chronologically as defined in condition (2) of Definition 14 (line 2). Next, the *ct*-groups are traversed, in order, so that each group's tieps are appended to the transformed *ct*-sequence *ctseq* (lines 4–11). When two successive *ct*-groups share the same timestamp (in which case the former has a *finish* end-type and the latter has a *start* end-type), their STIs are *met* (Allen relations, Fig. 1). Thus, as defined in Definition 14 (3), the *isMet* flag is set to *true* for the latter group, whose tieps' STIs are *met-by* the STIs of the former group's tieps (lines 6–7). Finally, the transformed *ct*-sequence *ctseq* is returned (line 14).

Algorithm 3 STIs2Seq

Input: *stis* – lexicographically ordered STIs series

Output: *ctseq* – transformed *ct*-sequence

```

1: tieps ← ExtractTieps(stis)
2: ctGroups ← PartitionToCTGroups(tieps)
3: ctseq ← ∅
4: for i ← 0 to |ctGroups| do
5:   tieps ← ctGroups[i].coTieps
6:   if i > 0 ∧ ctGroups[i].timestamp = ctGroups[i - 1].timestamp then
7:     isMet ← true
8:   else
9:     isMet ← false
10:  end if
11:  ctseq.append((tieps, isMet))
12:  addToTiepsIndex(tieps)
13: end for
14: return ctseq

```

The complete transformation process carried by the STIs2Seq procedure is illustrated in Fig. 4, in which the output *ct*-sequence on the right (3) is created from the STIs series input on the left (1). First the STIs' tieps are partitioned to *ct*-groups (2). When multiple coinciding tieps have the same end-type, as happens with A_0^+ and B_0^+ at timestamp 1, they are grouped together and ordered alphabetically according to the symbol. However, when the tieps' end-types are opposed, as happens with B_0^- and C_0^+ at timestamp 4, they appear separately, starting with the finish-tieps' group which is followed by the start-tieps' group. This scenario stands for the STI *C* being *met-by* the STI *B*. Therefore, when the *ct*-groups are collected to create the output *ct*-sequence, the *isMet* flag is set to *true* for C_0^+ , to which an

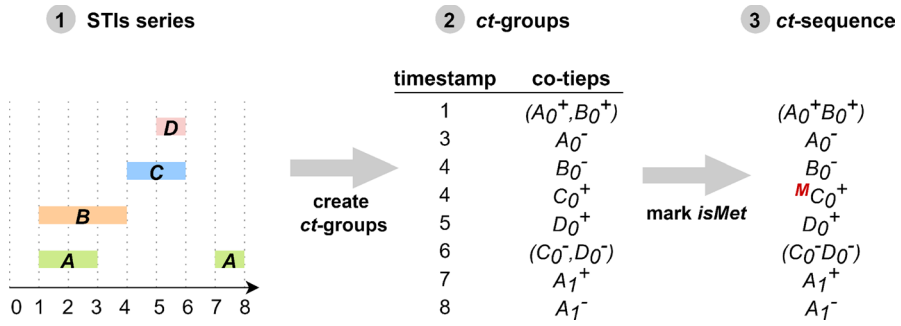


Fig. 4 The input transformation process—from an STIs series (1) into a *ct*-sequence (3). Coinciding-tieps having the same end-type are grouped. Otherwise, they appear separately, starting with the finish-tieps, which are followed by the start-tieps (2). In addition, an ^M character is inserted among them in the output *ct*-sequence to mark that their STIs are met

^M character is thus pre-pended in Fig. 4 (3). This is essential for a non-ambiguous representation, in which the *before* and *meet* temporal relations are distinguishable.

Figure 5 exemplifies TIRPClo’s sequential representation versus the existing sequence-based representations employed in CCMiner (Chen et al. 2016), TPrefixSpan (Wu and Chen 2007), and TPCMiner (Chen et al. 2015). In TIRPClo, all of Allen’s temporal relations are uniquely represented due to the following properties. First, TIRPClo always keeps the chronological order of the tieps, unlike in CCMiner, in which tieps having close timestamps are considered coinciding and marked by brackets (a), which may change their order sometimes and result in the ambiguity described in Sect. 2. In addition, TIRPClo groups together coinciding tieps which have the same *end-type* (i.e., start or finish), unlike in TPrefixSpan. For example, A_0^+ and B_0^+ in the series (b).

However, two tieps that do not have the same end-type are never grouped together in TIRPClo, unlike in both CCMiner and TPCMiner. When tieps have the same timestamp but their end-type is opposed, as happens with B_0^- and C_0^+ in series (b), their STIs are *met*. This is directly indicated in TIRPClo’s representation by the ^M character that is inserted among them.

After the STIs2Seq method is applied to all the entities’ STIs series in the dataset, their transformed *ct*-sequences are collected to form the initial *sequences database*. Along the projection-based mining process of the frequent TIRPs, the sequences database shrinks typically, or at least does not change, corresponding to

STIs series	TIRPClo rep.	CCMiner rep.	TPrefixSpan rep.	TPCMiner rep.
(a)	$A_0^+ B_0^+ C_0^+ A_0^- B_0^- C_0^-$	$A^+ B^+ (A^- C^+) B^- C^-$	$A^+ < B^+ < C^+ < A^- < B^- C^-$	$A^+ B^+ C^+ A^- B^- C^-$
(b)	$(A_0^+ B_0^+) B_0^- M C_0^+ A_0^- C_0^-$	$(A^+ B^+) B^- @ (A^- C^+) C^-$	$A^+ = B^+ < C^+ = B^- < A^- < C^-$	$(A^+ B^+) (B^- C^+) A^- C^-$

Fig. 5 Three STIs series and their sequential representations in TIRPClo, CCMiner (Chen et al. 2016), TPrefixSpan (Wu and Chen 2007), and TPCMiner (Chen et al. 2015)

the current projected pattern. In TIRPCLO, the output is designed to include the complete set of specific instances of the discovered frequent TIRPs. Therefore, a record in a projected sequences database of a discovered pattern is a pair $\langle ctseq, pi \rangle$, where $ctseq$ is a ct -sequence from which the specific pattern instance pi has been projected. Note that since in TIRPCLO it is allowed to have multiple occurrences of the same symbol within a single entity's STIs series, as often happens in real-world data, the projection of an entity's sequence can result in multiple projects of the same pattern. We elaborate more on that in the next subsection.

Finally, in line 12 of Algorithm 3, the ct -groups' ties are incrementally indexed in the *ties-index*, which is used for the retrieval of the ordered instances of the ties within their supporting entities in $O(1)$ time. The ties-index maps each tie's common representation (e.g., A^+) to a *master-tie* using a hash map. The master-tie indexes all of the tie's instances by entity, ordered by their timestamp. For example, a tie t 's i^{th} instance within an entity e , is accessed through $TiepsIndex[t][e][i]$ in a constant time. Figure 6 illustrates an example of two entities' STIs series, their sequential representation in TIRPCLO, and a part of the corresponding ties-index. Since there are three types of symbols in the data—i.e., A , B and C , the index consists of six ties entries— A^+ , A^- , B^+ , B^- , C^+ , and C^- . Each entry maps a tie's representation to its master-tie, which indexes the tie's instances by entity ID. For example, the master-tie of A^+ (in green) has two entries—one for e_1 and another for e_2 , since A^+ appears in both entities' records. In e_1 's record A^+ has two instances (ordered by their timestamp), while in e_2 's record A^+ has only a single instance. The master-tie of C^+ (in blue), however, contains only a single entry. That is since C^+ is solely supported by the entity e_2 , in which it appears just once.

3.2.1 Complexity

Assume an input dataset in which the set of entities is $E = \{e_0, \dots, e_{|E|}\}$ and the number of STIs of an entity e_i is n_i (the number of STIs of each entity may vary). Since each STI is comprised of exactly two ties, i.e., its start-tie and finish-tie, the time complexity of applying the STIS2SEQ method to a specific entity e_i is: $O(n_i)$ for breaking the STIs into their corresponding ties (line 1), $O(n_i \cdot \log(n_i))$ for the construction and sorting of the ct -groups (line 2), $O(n_i)$ for the ct -sequence construction from the ct -groups (lines 4–11), and $O(1)$ for indexing (line 11); which sums to a total of $O(n_i \cdot \log(n_i))$. Thus, the time complexity of transforming the entire dataset into the ties-based sequential representation through STIS2SEQ is $O(\max_{1 \leq i \leq |E|} n_i \cdot \log(n_i))$. In

addition, assuming a total number of $N = \sum_{i=1}^{|E|} n_i$ STIs in the data, the size of both the ties-index and the initial sequences database is bounded at $2N$, which equals to the total number of ties and yields an overall memory complexity which is linear in N . Note that effectively, the size of both the index and the initial sequences database is expected to be much smaller, as the infrequent ties are immediately filtered-out of them in lines 3–8 of Algorithm 1.

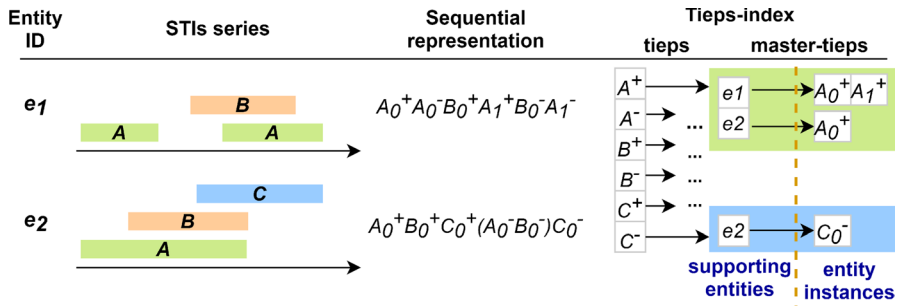


Fig. 6 Two entity’s STIs series, their sequential representation in TIRPC₁₀, and a part of the corresponding tieps-index

3.3 The entities spawning method for data projection

Sequences database projection methods typically search for the first instance of a tiep within an entity’s record, without intending to discover the horizontal support (Definition 10). A projected record then includes the elements appearing after the tiep’s first instance, as illustrated in Fig. 7a. In this figure it can be seen that typically, projection with respect to the tiep B^+ refers only to the first instance of the tiep, while there are another two instances which are ignored. Thus, using current projection methods for sequence-based TIRP mining eventually results in an *incomplete* discovery of frequent TIRPs due to not discovering all the TIRPs’ horizontally supporting instances, as described in Sect. 2.4.

The Entities Spawning projection method that we introduce here addresses this very challenge of horizontal support discovery, to guarantee a complete discovery of frequent TIRPs. For that, projection of an entity’s record by a tiep that appears N times within it, requires N projections—which result in multiple projected records. Figure 7b illustrates the multiple projections performed in TIRPC₁₀ for a complete discovery of frequent TIRPs. In this figure, it is shown that TIRPC₁₀’s projection refers to each of the three instances of the tiep B^+ within the original record, and results in three projected records. Furthermore, when having N instances of a tiep t within an entity’s record, TIRPC₁₀’s projection of the record by the i^{th} instance of t within it, keeps the next $N - i$ instances of t within the projected record. An example is shown in Fig. 7b, in which the first projected record (that corresponds to TIRPC₁₀’s projection of the original record by the first instance of the tiep B^+) includes the second and third instances of B^+ within it. Thus, TIRPs that include multiple occurrences of STIs which have the same symbol are discovered in TIRPC₁₀ as well.

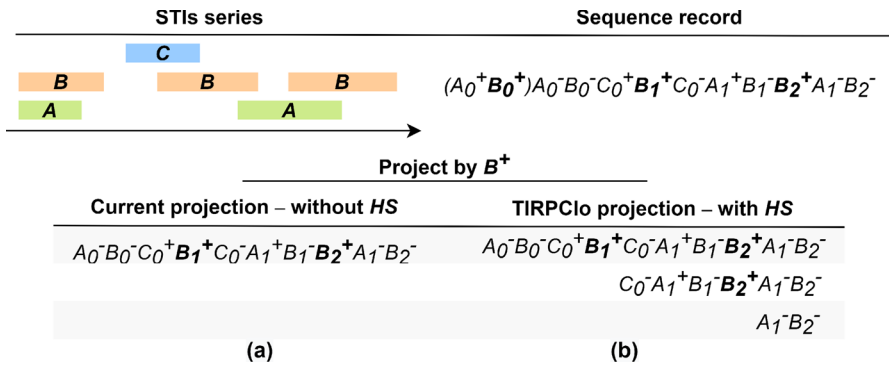


Fig. 7 Projection without discovering the horizontal support, as currently made (a), versus TIRPCLO's projection that includes horizontal support discovery (b). The sequence record stands for TIRPCLO's representation of the given STIs series, and it is projected by the tiep B^+ which appears three times within it (in bold)

To limit the discovery of potentially meaningless frequent TIRPs, which have very long time durations between their STIs, TIRPCLO uses a *maximal gap* time constraint, by which the projection is limited. The maximal gap has been first introduced in Papapetrou et al. (2009) and is defined in Definition 15. The effect of the maximal gap on the discovery of the frequent TIRPs is empirically investigated in experiment 5, followed by which several recommendations for choosing the appropriate maximal gap value for a given dataset are provided in the Discussion Sect. 6.

Definition 15 (Maximal gap) The *maximal gap* is the maximal time duration allowed between two STIs among which the temporal relation is *before*, for TIRPs discovery.

TIRPCLO's novel projection method is described in Algorithm 4, which receives three parameters as input: a sequences database *sdb* for further projection, a tiep *t* based on which the database is projected, and *t*'s support-index. First, using the support-index, *t*'s instances within each record are traversed from the first instance for projection (lines 2–4). The projection is conditioned by the maximal gap (Definition 15), making sure that the duration till the next STI is below it. Once a potential start-tiep is beyond the maximal gap relatively to the last finish-tiep of the current pattern, it means that its following instances will be for sure also beyond it. Thus, a break stops the routine (lines 5–7). Otherwise, projection is applied only to the specific *ct*-group which includes *t*'s current instance within the record's sequence (lines 8–9). The projected record (i.e., the projected *ct*-group concatenated to the rest of the sequence by a pointer), and the extended pattern instance are then added to the projected sequences database *sdb_t* (lines 10–11), which is eventually returned in line 14. Note that in line 11 of Algorithm 1 projection is employed to project the *initial* sequences database, in which case support-indices still do not exist (as they are only constructed during

the `ExtendTIRP` method, Algorithm 2). Thus, the `ties-index` is used instead for the traversal over all of the ordered instances of the tiep t within its supporting entities in lines 2–4 of Algorithm 4.

Algorithm 4 `ESProject`

Input: sdb – current sequences database
 t – tiep to project from sdb
 tSI – t 's support-index

Output: sdb_t – projected sequences database

```

1:  $sdb_t \leftarrow \emptyset$ 
2: for  $\langle record, firstInstance \rangle \in tSI.instances$  do
3:    $eInstances \leftarrow TiesIndex[t][record.entityID]$ 
4:   for  $tInstance \in eInstances[firstInstance : ]$  do
5:     if  $t.endType = START \wedge \neg maxGapHolds(record.pi, tInstance)$  then
6:       break
7:     end if
8:      $tCTG \leftarrow tInstance.ctGroup$ 
9:      $projectedRecord \leftarrow$  project  $tCTG$  by  $tInstance$ 
10:     $pi' \leftarrow$  extend  $record.pi$  with  $tInstance$ 
11:     $sdb_t.add(projectedRecord, pi')$ 
12:   end for
13: end for
14: return  $sdb_t$ 

```

Finally, besides discovering all the re-occurring instances of the tieps within their supporting entities, the `Entities Spawning` projection method is also much more efficient compared to typical projection mechanisms. That is since in `TIRPCLo` the tieps' instances are directly accessed through the index in a constant time, instead of re-scanning repetitively the entities' records. The proposed approach is also efficient memory-wise, since a projected record of an entity e points at its start position within e 's initial record, as made in pseudo-projection, rather than repeatedly making sub-copies of it.

3.3.1 Complexity

Assume a sequences database sdb of a current pattern p , and a tiep t for further projection. Then, due to the `ties-index` and the pseudo-projection approach, projection of a specific record in sdb with respect to a specific instance of the tiep t is performed in `TIRPCLo` in a constant time (lines 8–11). Since `TIRPCLo`'s projection refers to all the re-occurring instances of the tieps, the total number of performed projections in Algorithm 4 equals to the number of instances of t within sdb , which stands for the number of instances of the extended pattern $p' = p + t$. Let $\tau_{p'}$ denote the total number of instances of the pattern p' , which equals to $\sum_{e \in \text{supportingEntities}(p')} \text{horizontal support}(p', e)$, i.e., summing the horizontal support values of p' over all of its supporting entities. Then the overall time complexity of `TIRPCLo`'s projection is $O(\tau_{p'})$.

3.4 Candidate generation and support-indices

TIRPCLO's candidate generation scheme aims at minimizing not only the number of generated candidate ties for the extension of a current frequent pattern p , but also the computational cost of extending p with a selected generated candidate tie cmd_t . For that purpose, TIRPCLO uses the support-indices. As was introduced in Sect. 3.1, support-indices are created only for ties that are valid candidates for the extension of p . Each support-index keeps track of such tie's current vertical support value, and points at the tie's first instance within each record in the projected sequences database of p . As described in the previous subsections, due to these properties, all the extended patterns in TIRPCLO are guaranteed to be frequent, and the performance of TIRPCLO's projection is enhanced. In this subsection we describe the creation process of the support-indices, but first the conditions for the validity of a candidate tie, for which a support-index will be created, are described.

Since the start-tie of an STI always precedes its finish-tie, a current pattern p cannot be extended by any finish-tie unless it already contains its complementing start-tie. Otherwise, the extended pattern could never be re-extended to form a valid TIRP, in which all the ties appear paired with their complementing ties. TIRPCLO enables the discovery of TIRPs that include multiple STIs which have the same symbol. Hence, a tie t (e.g., A^+) may occur multiple times within a discovered pattern p . To represent the number of occurrences of t within p , the notation $\#_p t$ is introduced.

Accordingly, the conditions for a tie cmd_t which is a valid candidate for the extension of p are the following:

1. If cmd_t is a start-tie, it is always a valid candidate for p 's extension.
2. If cmd_t is a finish-tie, then it is a valid candidate if $\#_p cmd_t < \#_p cmd_t^C$, which means that there are more occurrences of the complementing start-tie of cmd_t within p , than occurrences of cmd_t within p .

Example 4 The valid candidate ties for the extension of the pattern $q = \langle A^+ \rangle$ are all the start-ties (condition 1) and the single finish-tie A^- (condition 2). The valid candidate ties for the extension of $p = \langle A^+ B^+ \rangle$, however, include not only the valid candidates of q , but also the finish-tie B^- . That is since currently $0 = \#_p B^- < \#_p B^+ = 1$, which stands for the validity of B^- according to condition 2. For any other finish-tie which has a symbol $X \notin \{A, B\}$, it is still an invalid candidate for the extension of p since $0 = \#_p X^- = \#_p X^+ = 0$.

Lemma 1 Assume two patterns p and q , and a tiep l_p such that $p = q + l_p$, which means that the pattern p is the result of extending q with the tiep l_p . Then, for any tiep $t \notin \{l_p, l_p^C\}$ (i.e., t is neither l_p nor its complementing-tiep), t is a valid candidate for the extension of q if and only if it is a valid candidate for the extension of p .

Proof First, according to condition 1, a start-tiep t is always a valid candidate. Hence, suppose a finish-tiep $t \notin \{l_p, l_p^C\}$. According to condition 2, t is a valid candidate for the extension of the pattern q if and only if $\#_q t < \#_q t^C$. Since $t \notin \{l_p, l_p^C\}$ (under the assumption), $\#_q t = \#_p t$ and $\#_q t^C = \#_p t^C$. Therefore, $\#_q t < \#_q t^C \equiv \#_p t < \#_p t^C$ which is also equivalent to t being a valid candidate for the extension of p due to condition 2. \square

Based on Lemma 1 and the two conditions for the validity of a candidate tiep, the support-indices are created via the `GetSupportIndices` method (Algorithm 5). The method receives three parameters as input: a current pattern p , p 's projected sequences database sdb_p , and the previous candidate tieps' support-indices $prevSIs$, based on which the new candidate tieps' support-indices are created. Assume that $p = q + l_p$, i.e., the pattern p is the result of extending a pattern q with the tiep l_p , which is thus the last tiep of p . Then, according to Lemma 1, the set of previously valid candidate tieps (i.e., for the extension of q) is identical to the set of currently valid candidate tieps (i.e., for the extension of p). That is except for the tieps l_p and l_p^C , to which we will refer soon. Therefore, in Algorithm 5, the previous candidate tieps' support-indices are traversed for the purpose of creating the new candidate tieps' support-indices (line 5). Note that the tiep l_p was for sure a valid candidate previously, since q was extended by l_p forming the current pattern p . However, if l_p is a finish-tiep it might no longer be a valid candidate for the extension of p despite it has been a valid candidate previously. That is, subject to condition 2. In case condition 2 is violated, we make sure that a new support-index would not be created for l_p (lines 2–4).

New support-indices for the currently valid candidate tieps are created in lines 5–22. First, tieps that have been previously infrequent are ignored (lines 6–8). That is since they cannot extend p to form frequent TIRPs. For each valid candidate tiep that has been previously frequent, its new support-index is incrementally created based on the tiep's first instances within each record in the current projected sequences database sdb_p . These instances are collected by traversing the tiep's instances within each record through the tieps-index—from the first instance in the previous projected record (i.e., before the last projection) as indicated by the previous support-index, and until the first instance occurring within the current projected record (lines 9–12).

Algorithm 5 GetSupportIndices

Input: p – current frequent pattern
 sdb_p – p 's projected sequences database
 $prevSIs$ – set of previous support-indices

Output: $newSIs$ – set of new support-indices

```

1:  $newSIs \leftarrow \emptyset$ ,  $l_p \leftarrow p$ 's last tiep
2: if  $l_p.endType = FINISH \wedge \#_p l_p \geq \#_p l_p^C$  then
3:    $prevSIs \leftarrow prevSIs \setminus \{l_p\}$ 
4: end if
5: for  $\langle t, tSI \rangle \in prevSIs$  do
6:   if  $tSI.verticalSupport < minSup$  then
7:     continue
8:   end if
9:   for  $record \in sdb_p \cap tSI.instances$  do
10:     $firstInstance \leftarrow tSI.instances[record]$ 
11:     $eInstances \leftarrow TiesIndex[t][record.entityID]$ 
12:    for  $tInstance \in eInstances[firstInstance : ]$  do
13:      if  $t.endType = START \wedge \neg maxGapHolds(record.pi, tInstance)$  then
14:        break
15:      end if
16:      if  $tInstance \in record.ctseq$  then
17:         $newSIs.addInstance(t, record, tInstance)$ 
18:        break
19:      end if
20:    end for
21:  end for
22: end for
23: if  $l_p^C \notin prevSIs$  then
24:    $newSIs \leftarrow newSIs \cup \{GetSupportIndex(sdb_p, l_p^C)\}$ 
25: end if
26: return  $newSIs$ 

```

For a start-tie t , having one of its instances beyond the maximal gap (relatively to the last finish-tie in p), means that t 's following instances will be for sure also beyond it. Thus, a break stops the routine (lines 13–15). Otherwise, once the first instance of the tie is discovered within each record, it is added to the new support-index (lines 16–17). At last, in lines 23–25, a new-support index is also created for the complementing-tie of p 's last tie— l_p^C , which is for sure a currently valid candidate regardless of its end-type. That is, of course, only if a new support-index has not yet been created for it in lines 5–22. Eventually, the set of newly created support-indices is returned by the algorithm (line 26).

Additionally, note that when support-indices are created for a single-tie pattern $p = \langle l_p \rangle$, previous support-indices still do not exist. Hence, new support-indices are created from scratch, based on the ties-index, rather than being extended based on the previous support-indices. Since the initial sequences database is projected only by start-ties in Algorithm 1, the tie l_p must be a start-tie. Thus, according to the conditions for the validity of a candidate tie, initial support-indices are created for

all the frequent start-tieps (condition 1), as well as for the only finish-tiep l_p^C (condition 2).

3.4.1 Complexity

Assume an input dataset with S symbol types, and a current pattern $p = q + l_p$. Then, to create the support-index of a single valid candidate tiep cdn_t , the specific instances of the tiep are traversed within each record in p 's projected sequences database sdb_p —from the previous first instance indicated by the previous support-index, and until the first instance occurring within the current projected record (lines 12–20). For a worst case assessment, assume that cdn_t occurs within each record in sdb_p , and that all of its instances are always traversed until the last one in lines 12–20.

Then, as described in the previous subsection, the number of records within sdb_p equals to the number of instances of the current pattern p , which we denote by τ_p . In addition, note that the total number of instances of the tiep cdn_t across all records, counting from the first instances indicated by the previous support-index, equals to the number of instances of cdn_t within the previous projected sequences database—of the pattern q . As described, this is the number of instances of the pattern $q' = q + cnd_t$, which we denote by $\tau_{q'}$. Therefore, having a direct access to all the relevant instances of the tieps through the tieps-index in $O(1)$ time, a worst case assessment of the complexity of creating the support-index of a single candidate tiep cdn_t is $O(\tau_p + \tau_{q'})$. The maximal number of valid candidate tieps for the extension of p is $2S$, including the start-tieps and the finish-tieps of all the symbol types in the dataset. Thus, summing over all the $O(S)$ valid candidate tieps, an overall time complexity of $O(S \cdot (\tau_p + \max_{q'=q+cnd_t} \tau_{q'}))$ is assessed in the worst-case for TIRPCLO's candidate generation process.

3.5 Closure-checking

TIRPCLO's description in Algorithms 1–2 deals with the task of mining the entire set of frequent TIRPs. However, TIRPCLO is designed also for the complete mining of only the frequent closed TIRPs. For that, the algorithm utilizes a *closure-checking* scheme, which is used for the detection and possibly pruning of the unclosed TIRPs early during the mining process. Since TIRPCLO is a sequence-based TIRP mining algorithm, its closure-checking scheme is inspired by the bi-directional extension approach (Wang and Han 2004), that has been originally used for closed sequential pattern mining.

In TIRPCLO, the TIRPs are represented as sequences of their STIs' tieps. Thus, assume a TIRP which is represented in TIRPCLO by the pattern $p = \langle t_1, \dots, t_k \rangle$. Then, TIRPCLO's closure-checking scheme searches for an additional STI X whose both start-tiep and finish-tiep could extend p forming a super-TIRP which has the same vertical support. In such case, according to Definition 8, p is for sure not a closed TIRP. The tieps of such an STI X are referred to as *forward-extension* and *backward-extension* tieps, and they are formally defined as follows.

Definition 16 (*Forward-extension tiep*) A tiep t^* is a *forward-extension* tiep of a pattern $p = \langle t_1, \dots, t_k \rangle$ if $\text{vertical support}(p) = \text{vertical support}(p^{F_{t^*}})$, for p 's super-pattern $p^{F_{t^*}} = p + t^* = \langle t_1, \dots, t_k, t^* \rangle$. In this subsection, the notation $F(p, t^*)$ will be used to denote the set of records within the sequences database of p that support the super-pattern $p^{F_{t^*}}$.

Definition 17 (*Backward-extension tiep*) A tiep t^* is a *backward-extension* tiep of a pattern $p = \langle t_1, \dots, t_k \rangle$ if $\exists 1 \leq i \leq k$ such that $\text{vertical support}(p) = \text{vertical support}(p^{B_{i,t^*}})$, for p 's super-pattern $p^{B_{i,t^*}} = \langle t_1, \dots, t_{i-1}, t^*, t_i, \dots, t_k \rangle$. In this subsection, the notation $B^i(p, t^*)$ will be used to denote the set of records within the sequences database of p that support the super-pattern $p^{B_{i,t^*}}$.

Example 5 Given the sample dataset shown in Fig. 6, the tiep $t_1^* = B^-$ is a forward-extension tiep of the pattern $p = \langle B^+ \rangle$. That is since $\text{vertical support}(p) = \text{vertical support}(p^{F_{B^-}} = \langle B^+ B^- \rangle) = \frac{2}{2} = 1.0$. The tiep $t_2^* = A^+$, however, is a backward-extension tiep of the pattern p for $i = 1$, since $\text{vertical support}(p) = \text{vertical support}(p^{B_{1,A^+}} = \langle A^+ B^+ \rangle) = \frac{2}{2} = 1.0$.

Accordingly, let a current frequent pattern $p = \langle t_1, \dots, t_k \rangle$. TIRPCLO's closure-checking scheme is based on the following three claims:

Claim 1 Assume that 1) X^+ and X^- are backward-extension tieps of p for $1 \leq i \leq j \leq k$, and that 2) $\text{vertical support}(p) = |\{\text{record.entityID} \mid \text{record} \in B^i(p, X^+) \cap B^j(p, X^-)\}|$. Then p , as well as any future extension of it, are not closed TIRPs.

Proof Under the assumptions, each supporting entity of p has at least a single record within p 's sequences database that supports both of p 's super-patterns $p^{B_{i,X^+}}$ and $p^{B_{j,X^-}}$. Hence, the pattern $p^{B_{ij,X}} = \langle t_1, \dots, t_{i-1}, X^+, t_i, \dots, t_{j-1}, X^-, t_j, \dots, t_k \rangle$, which is a super-pattern of p as well, is supported by all of p 's supporting entities. Thus, $\text{vertical support}(p) = \text{vertical support}(p^{B_{ij,X}})$, and according to Definition 8, p is not a closed TIRP. Similarly, let an extended pattern of p : $q = p + \langle t_{k+1}, \dots, t_n \rangle$. Then $\text{vertical support}(q) = \text{vertical support}(q^{B_{ij,X}})$, for $q^{B_{ij,X}} = p^{B_{ij,X}} + \langle t_{k+1}, \dots, t_n \rangle$, which is a super-pattern of q . Therefore, p as well as any future extension of it q , are not closed TIRPs (Definition 8). \square

Claim 2 Assume that 1) X^+ is a backward-extension tiep of p for $1 \leq i \leq k$, 2) X^- is a forward-extension tiep of p , and that 3) $\text{vertical support}(p) = |\{\text{record.entityID} \mid \text{record} \in B^i(p, X^+) \cap F(p, X^-)\}|$. Then p is not a closed TIRP.

Proof Similar to Claim 1, $\text{vertical support}(p) = \text{vertical support}(p^{B_{i,F_X}})$ for p 's super-pattern $p^{B_{i,F_X}} = \langle t_1, \dots, t_{i-1}, X^+, t_i, \dots, t_k, X^- \rangle$. Therefore, p is not a closed TIRP according to Definition 8. \square

Claim 3 Assume that X^+ is a forward-extension tiep of p . Then p is not a closed TIRP.

Proof Under the assumption and according to Definition 16, $\text{vertical support}(p) = \text{vertical support}(p_{X^+}^F)$. Since the start-tiep of an STI always precedes its complementing finish-tiep, $\text{vertical support}(p) = \text{vertical support}(p_X^F)$ for $p_X^F = p_{X^+}^F + X^- = \langle t_1, \dots, t_k, X^+, X^- \rangle$. Since p_X^F is a super-pattern of p which has the same vertical support, p is not a closed TIRP according to Definition 8. \square

If the conditions for the first claim hold for p , it cannot be further extended to form any frequent closed TIRP. Therefore, p is safely pruned. Otherwise, if either the conditions for the second or third claims hold for p , it is not a closed TIRP. However, it may potentially be further extended to form frequent closed TIRPs. Thus, in such case, p is not added to the set of frequent closed TIRPs that are discovered by the algorithm, yet it is not pruned. Finally, for the complete mining of the frequent closed TIRPs in TIRPCl_0 , it is important to verify that backward-extension and forward-extension tieps which have the same symbol, indeed originate in the same specific STI. That is since TIRPCl_0 allows multiple occurrences of STIs having the same symbol within a single entity's STIs series. By following this closure-checking scheme and to the best of our knowledge, TIRPCl_0 is the *first algorithm that guarantees a complete discovery of frequent closed TIRPs*.

3.6 TIRPCl_0 completeness and complexity

In this subsection we provide proofs for TIRPCl_0 's completeness, which means that the algorithm's discovery of either the entire set of frequent TIRPs, or only the frequent closed TIRPs, is complete (Definitions 5, 9). In addition, an analytical complexity analysis of TIRPCl_0 is presented, which is a worst case assessment of the algorithm's runtime.

3.6.1 Completeness

Claim 4 TIRPCl_0 is a complete frequent TIRPs discovery algorithm.

Proof Let a frequent k -sized TIRP which is represented in TIRPCl_0 by the $2k$ -tieps pattern $T = \langle t_1, \dots, t_{2k} \rangle$. In order to prove that TIRPCl_0 discovers the pattern T , it is sufficient to show that for any $1 \leq i \leq 2k$, the pattern $T_i = \langle t_1, \dots, t_i \rangle$ is discovered and extended in TIRPCl_0 . We prove that by induction on i . (1) Since T is a frequent and valid TIRP, the tiep t_1 is for sure a frequent start-tiep. Therefore, the single-tiep pattern $T_1 = \langle t_1 \rangle$ is discovered and extended in TIRPCl_0 in lines 9–12 of Algorithm 1. (2) Assume that for some $1 < i < 2k$, T_i is discovered and extended in TIRPCl_0 . Then, as described in Sect. 3.3, since TIRPCl_0 's projection refers to all the re-occurring instances of the tieps, the projected sequences database of T_i includes a single projected record for each of T_i 's instances. Since T is frequent,

its prefix $T_{i+1} = T_i + t_{i+1}$ is frequent as well, and thus t_{i+1} is for sure frequent within the projected sequences database of T_i . In addition, t_{i+1} is a valid candidate for the extension of T_i (Sect. 3.4). Otherwise T is invalid, which contradicts the assumption. Therefore, T_{i+1} is extended in TIRPCLO (Algorithm 2 lines 7–9) as well, and due to (1) and (2) the pattern T is discovered by the algorithm. \square

Claim 5 TIRPCLO is a complete frequent closed TIRPs discovery algorithm.

Proof Let a frequent k -sized closed TIRP which is represented in TIRPCLO by the $2k$ -tieps pattern $T = \langle t_1, \dots, t_{2k} \rangle$. According to the previous claim, T is discovered in TIRPCLO when no pruning of unclosed TIRPs is performed. Assume that for some $0 < i < 2k$, $T_i = \langle t_1, \dots, t_i \rangle$ is pruned. Then, according to TIRPCLO's closure-checking scheme (Sect. 3.5), the conditions for claim 1 hold for T_i . Thus, T_i as well as any future extension of it, including T , are not closed TIRPs, which contradicts the assumption. \square

3.6.2 Complexity

As is often the case in pattern mining algorithms, it is almost impossible to quantify the input data, and as a consequence to provide an accurate analytical analysis of the time complexity of a frequent TIRP mining algorithm, which is therefore typically of lower interest compared to an empirical analysis. Thus, previous papers in the field of TIRP mining did not show a theoretical complexity analysis (Winarko and Roddick 2007; Wu and Chen 2007; Patel et al. 2008; Papapetrou et al. 2009; Chen et al. 2015; Moskovitch and Shahar 2015c; Chen et al. 2016; Sharma and Patel 2018; Lee et al. 2020), but rather solely conducted an empirical performance evaluation on multiple benchmark datasets. That is except for (Moskovitch and Shahar 2015b), in which a worst case assessment of the time complexity of the KarmaLego algorithm has been presented, which is in fact common for most time intervals-based TIRP mining methods and resembles the analysis provided in Mordvanyuk et al. (2021).

In this paper, we empirically evaluate the performance of the proposed TIRPCLO algorithm on a wide benchmark of over a dozen datasets compared to four state-of-the-art methods, which is in fact the widest benchmark to ever been used for runtime comparisons in a TIRP mining paper. However, for the readers' convenience, we also provide a theoretical complexity analysis of TIRPCLO, which is a simplified worst case assessment of the actual runtime and is somewhat common for any sequence-based TIRP mining method.

Assume:

S —number of symbol types

N —total number of STIs in the dataset

n —maximal number of STIs within an entities' STIs series

L —maximal number of STIs within a frequent TIRP

Initially, `TIRPCLO` transforms the input STIs series data into a tieps-based sequential representation through the `STIs2Seq` method (Algorithm 3). This input transformation process is conducted only once and has a time complexity of $O(n \cdot \log(n))$, as described in Sect. 3.2. Then, the recursive TIRP mining process is performed, following a DFS-like approach. In each step along the mining process a current frequent pattern q is extended by a tiep t to form a new, extended pattern $p = q + t$. Each such pattern extension step requires (1) to project the sequences database of q with respect to the tiep t , and then (2) to generate all the candidate tieps for the extension of the new pattern p . Based on the analysis presented in Sects. 3.3–3.4, the complexity of projection is $O(\tau_p)$, while the complexity of generating each specific candidate tiep cmd_t is $O(\tau_p + \tau_{q'})$ for $q' = q + cmd_t$. Since the maximal number of instances of a frequent TIRP in the input dataset is for sure smaller than N , a simplified worst case assessment of the complexity of any projection or a specific candidate's generation along the mining process is $O(N)$.

Note that as a sequence-based algorithm, to discover a k -sized TIRP in `TIRPCLO` it is necessary to perform $2k$ pattern extension steps—for the start-tiep and finish-tiep of each of the TIRP's k STIs. Hence, under the assumption of having at most L STIs within a frequent TIRP, the depth of the algorithm's search space is bounded by $2L$. In addition, as analyzed in Sect. 3.4, the maximal number of candidate tieps that are generated for the extension of any current pattern is $2S$. For a worst case assessment, assume that the search space constitutes a full tree. Thus, the time complexity of the entire frequent TIRP mining process in `TIRPCLO`, including the discovery of all of the specific instances of the discovered TIRPs, can be assessed by $O(N \cdot (2S)^{2L} + n \cdot \log(n))$ in the worst case. Nevertheless, we highlight that this is only a theoretical upper-bound for the complexity of the proposed algorithm, while the actual runtime is much shorter typically as will be shown in the empirical evaluation. For the readers' convenience, in Appendix C we also provide a concise analysis of the complexity of the more basic task of sequential pattern mining, where, unlike in TIRP mining, both the input data and the discovered patterns only consist of time point-based events, rather than STIs.

4 Experimental setup

This paper's experimental evaluation mainly focuses on comparing the performance of the proposed `TIRPCLO` algorithm to state-of-the-art methods, in terms of both runtime and memory consumption. Since there is no existing TIRP mining method designed for both the discovery of the entire set of frequent TIRPs and the discovery of only the frequent closed TIRPs, except for `TIRPCLO`, performance comparisons have been divided into the two TIRP mining tasks that the algorithm performs. The evaluation has been conducted based on an extensive benchmark of eleven real-world datasets, as well as four novel synthetic datasets, which is the widest benchmark to ever been used for runtime comparisons in a TIRP mining paper. In addition to the performance comparisons, we also conducted a rigorous quantitative analysis of the common completeness problem in frequent TIRP mining on real-world data, to supplement the theoretical discussion made in Sect. 2.4. Finally, `TIRPCLO` was

also run with different parameters to evaluate and exemplify the trade-off between its runtime duration and the number of discovered frequent TIRPs for different execution settings.

All the compared methods were implemented in Visual C# and the experiments were conducted on a Dell G5, having 16GB main memory, running Microsoft Windows 10. For the sake of conducting as fair comparison as possible, we verified that *the exact same set of frequent TIRPs were discovered by all the compared methods* in the respective experiments. In addition, input reading and output writing across all the compared methods were made using the exact same code, to make sure that runtime differences among the compared methods are solely due to the efficiency of the TIRP mining process. To allow the complete reproducibility of the experimental results, the evaluation datasets, as well as the code of TIRPCLO, were made publicly available on GitHub (a link provided in the Introduction Sect. 1).

4.1 Datasets

To evaluate the performance of the proposed TIRPCLO algorithm in each of the two TIRP mining tasks that the algorithm performs, we used a benchmark of eleven real-world datasets from various application domains (datasets 1–11 in Table 1). To the best of our knowledge, these datasets include all the publicly available time intervals datasets (Patel et al. 2008; Papapetrou et al. 2009; Mörchen and Fradkin 2010; Jakula and Cook 2011; Moskovitch and Shahar 2015c). In addition, to examine the performance of the compared methods in extreme scenarios which are not met by any of the real-world datasets (e.g., an extremely large number of STIs or symbol types), performance experiments were also conducted on two novel synthetic datasets (datasets 12–13 in Table 1). The synthetic datasets have been generated by our synthetic datasets generator, which is also provided as part of the online code repository. The generator requires the following seven input parameters:

- $\{minE, maxE\}$ —lower and upper bounds for the total number of entities' STIs series in the dataset, which is chosen uniformly at random between them.
- $\{minSTIs, maxSTIs\}$ —lower and upper bounds for the number of STIs within a single entity's STIs series. For each entity, its number of STIs is chosen uniformly at random between $minSTIs$ and $maxSTIs$.
- $maxDuration$ —the maximal time duration of a single STI.
- $maxTimestamp$ —the maximal timestamp at which an STI may start.
- $maxS$ —the maximal number of symbol types.

Using the synthetic datasets generator, two additional datasets were generated, to which arbitrary TIRPs of two to ten STIs with vertical support values of 10–70% were injected (datasets 14–15 in Table 1). These datasets were not used for the performance comparisons, but only for an advanced analysis of TIRPCLO's discovery of entire set of frequent TIRPs versus the discovery of only the frequent closed TIRPs, on which we will elaborate in the next subsection.

Table 1 Evaluation datasets

	Name	$ E $	$ STIs $	$\frac{ STIs }{ E }$	MHS	$ S $
1	ASL	65	2037	31.3	1.2	146
2	Diabetes	2038	80538	39.52	3.5	35
3	Smart-home	89	23213	260.8	8.8	95
4	ASL-BU	440	17961	40.82	1.44	154
5	ASL-GT	1751	72249	42.26	1.39	47
6	Auslan2	200	900	4.5	1.23	12
7	Blocks	210	1027	5.74	1.19	8
8	Context	240	12916	53.82	2.41	54
9	Pioneer	160	4883	30.51	1.1	92
10	Skating	530	18911	53.86	1.76	41
11	Hepatitis	498	48029	96.44	2.41	63
12	ST ₁	2746	163490	59.54	1.1	1299
13	ST ₂	1805	957451	530.5	1.3	1299
14	CT ₁	1060	45240	42.68	1.3	49
15	CT ₂	576	176916	307.1	6.29	64

Bold values indicate the extreme conditions of each dataset

Table 1 describes the fifteen datasets that have been used for the evaluation of TIRPClo considering five parameters: $|E|$ —the number of entities, $|STIs|$ —the total number of STIs in the dataset, $\frac{|STIs|}{|E|}$ —the mean number of STIs per entity, MHS —the mean horizontal support of a symbol within an entity, and $|S|$ —the number of symbol types. The extreme conditions of each dataset appear in bold in Table 1. A more detailed description of the real-world datasets is provided in Appendix A.

4.2 Experimental plan

The experimental evaluation is divided into five major experiments as follows:

- Runtime and memory consumption evaluation of TIRPClo in the discovery of the entire set of frequent TIRPs, compared to KarmaLego (Moskovitch and Shahar 2015c), ZMiner (Lee et al. 2020), and VertTIRP (Mordvanyuk et al. 2021).
- Runtime and memory consumption evaluation of TIRPClo in the discovery of only the frequent closed TIRP, compared to CCMiner (Chen et al. 2016).
- Quantitative analysis of the common completeness problem in frequent TIRP mining, in which the effect of the horizontal support discovery on the complete discovery of the frequent TIRPs is evaluated, either when mining the entire set of frequent TIRPs or only the frequent closed TIRPs.
- Comparison of TIRPClo's discovery of the entire set of frequent TIRPs versus the discovery of only the frequent closed TIRPs, analyzing major pros and cons of applying TIRPClo for each of the two TIRP mining tasks.
- Maximal gap analysis, in which the effect of the maximal gap parameter on TIRPClo's runtime duration and the total number of frequent TIRPs that are discovered is evaluated.

4.2.1 Experiment 1: entire frequent TIRP mining

In this experiment, we evaluated the runtime duration and the memory consumption of TIRPClo in the discovery of the entire set of frequent TIRPs, compared to (1) KarmaLego (Moskovitch and Shahar 2015c) with its enhanced index proposed in (Moskovitch et al. 2015), which was proved to be faster than ARMADA (Winarko and Roddick 2007), IEMiner (Patel et al. 2008), and H-DFS (Papapetrou et al. 2009; 2) ZMiner (Lee et al. 2020), which outperformed TPMiner (Chen et al. 2015) and STIPA (Sharma and Patel 2018); and (3) VertTIRP (Mordvanyuk et al. 2021). The comparisons were conducted on the full benchmark of eleven real-world datasets, as well as on the ST_1 and ST_2 synthetic datasets (datasets 1–13 in Table 1). On the majority of datasets, all methods were run with minimum vertical support thresholds of 10–70% and a fixed, common maximal gap value of 30. In several datasets, however, lower minimum vertical support thresholds and larger maximal gap values were required to discover a considerable number of frequent TIRPs, that allow a meaningful comparison. Thus, in these datasets (datasets 4–7, 10–12 in Table 1) the maximal gap value was set to infinity.

Note that in several TIRP mining papers in the literature, performance comparisons have often considered extremely low levels of minimum vertical support (even below 1%), which in fact usually does not show a meaningful difference. That is unlike a meaningful runtime duration difference already at higher levels of the minimum vertical support, when having dozens of percentages, as we show in this paper. Finally, in this experiment, as well as in the next experiment, we verified that *the exact same set of frequent TIRPs was discovered by all the compared methods* in all the datasets.

4.2.2 Experiment 2: frequent closed TIRP mining

The goal in this experiment was to evaluate `TIRPCLo`'s runtime duration and memory consumption in the discovery of only the frequent closed TIRPs, compared to `CCMiner` (Chen et al. 2016), which is the only current method designed for closed TIRP mining. Note that `CCMiner` discovers only the first instances of the TIRPs within each series of STIs. Hence, its projection was repeatedly applied in order to discover all the horizontally supporting instances of the TIRPs, which is essential for completeness. In addition, since `TIRPCLo` uses the maximal gap constraint, similar to `KarmaLego`, `ZMiner`, and `VertTIRP`; this parameter was also added to `CCMiner`, to conduct as fair comparison as possible between the methods. This experiment's settings are the same as described in experiment 1.

4.2.3 Experiment 3: completeness in frequent TIRP mining

To supplement the theoretical discussion regarding the incompleteness of the majority of existing methods for either the discovery of the entire set of frequent TIRPs or only the frequent closed TIRPs (Sect. 2.4), a quantitative analysis was designed. The goal of the analysis was to empirically substantiate the need in the discovery of all the horizontally supporting instances of the TIRPs (Definition 10) for completeness in frequent TIRP mining. That is, by showing the differences in both the number of discovered frequent TIRPs and in the number of their discovered instances, with and without the horizontal support discovery in real-world data. If such significant differences indeed persist, then it is not just a theoretical issue (which may be fine as well), but rather a fundamental aspect in frequent TIRP mining, which may ultimately be critical for a large variety of downstream tasks relying on the output of the frequent TIRPs discovery process.

For that purpose, `TIRPCLo` was run once without discovering the horizontally supporting instances of the TIRPs as previously made in (Winarko and Roddick 2007; Wu and Chen 2007; Patel et al. 2008; Papapetrou et al. 2009; Chen et al. 2015, 2016), and then including horizontal support discovery which is required for completeness. Taking advantage of `TIRPCLo`'s capability to discover all the specific instances of the frequent TIRPs, the evaluation metrics in this experiment include not only the comparison of (1) the number of frequent TIRPs that were discovered in each approach, but also (2) the mean number of discovered instances of a discovered frequent TIRP, and (3) the mean horizontal support value of a discovered frequent TIRP within an entity in the dataset.

These comparisons were conducted both in the discovery of the entire set of frequent TIRPs, as well as in the discovery of only the frequent closed TIRPs. For the former comparison we used the smart-home dataset, while the diabetes dataset was used for the latter. In both datasets, minimum vertical support thresholds of 20–70% were evaluated with a fixed, common maximal gap value of 20. These datasets were selected for this experiment due to their relatively high mean horizontal support value of a symbol within an entity among the real-world datasets (Table 1), and also due to the considerable number of frequent TIRPs that they include, given the respective range of minimum vertical support thresholds and the maximal gap value.

4.2.4 Experiment 4: entire frequent TIRP mining versus frequent closed TIRP mining using TIRPCLO

Since the entire set of frequent TIRPs can be revealed based on the more compact set of only the frequent closed TIRPs, the knowledge one can extract from the output of each of the two TIRP mining tasks that TIRPCLO performs is identical. Thus, in this experiment, we wanted to analyze the trade-off in applying TIRPCLO for each of them. That is, focusing on the runtime duration of TIRPCLO and on the reduction in the number of discovered frequent closed TIRPs compared to the entire set of frequent TIRPs. For that, TIRPCLO was run once for each of the two TIRP mining tasks—i.e., either for mining the entire set of frequent TIRPs or only the frequent closed TIRPs, on the CT₁ and CT₂ synthetic datasets (datasets 14–15 in Table 1). In both datasets minimum vertical support thresholds of 10–70% were used with a fixed maximal gap value of 100.

4.2.5 Experiment 5: maximal gap analysis

Despite the maximal gap parameter (Definition 15) was already used in Papapetrou et al. (2009), Moskovitch and Shahar (2015c), Lee et al. (2020), Mordvanyuk et al. (2021) to limit Allen's *before* temporal relation, its effects on the discovery of the frequent TIRPs were never evaluated empirically. In this experiment, the goal was to give a notion for these effects, focusing on the trade-off between the runtime duration of TIRPCLO and the number of discovered frequent TIRPs when changing the value of the maximal gap parameter. For that, TIRPCLO was run on the ASL, diabetes, smart-home, context, and pioneer datasets (datasets 1–3, 8–9 in Table 1), which are all the real-world datasets in which a considerable number of frequent TIRPs could be discovered for finite maximal gap values, as explained in experiment 1. In each of these datasets maximal gap values between 10 and 40 were evaluated, having the minimum vertical support value fixed on each of the following thresholds: 20%, 30%, and 40%.

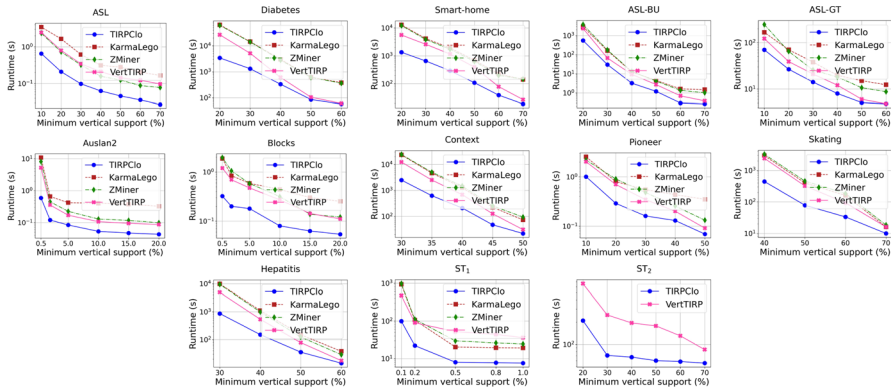


Fig. 8 Runtime comparison in the discovery of the entire set of frequent TIRPs. TIRPClo was faster than both KarmaLego, ZMiner, and VertTIRP in all the datasets and already at relatively high levels of minimum vertical support, reporting speed-ups by a factor of 1.5 up to more than 10

5 Results

5.1 Experiment 1: entire frequent TIRP mining

In this experiment, TIRPClo's runtime duration and memory consumption were compared to KarmaLego (Moskovitch and Shahar 2015c), ZMiner (Lee et al. 2020), and VertTIRP (Mordvanyuk et al. 2021) in the discovery of the entire set of frequent TIRPs. Figure 8 shows the runtime duration comparison when running the algorithms on the full benchmark of real-world datasets as well as on the two synthetic datasets ST_1 and ST_2 , using a logarithmic scale.

In Fig. 8 TIRPClo's superior performance compared to both KarmaLego, ZMiner, and VertTIRP is demonstrated in all the benchmark datasets, and already at considerably high levels of minimum vertical support. In the ASL, diabetes, smart-home, and ASL-BU datasets, for example, TIRPClo was 5 to 10 times faster than both KarmaLego and ZMiner, and 2–4 times faster than VertTIRP, already at 30–50% of minimum vertical support. At lower thresholds, e.g., 20%, the runtime duration differences between the compared methods were typically larger, reaching speed-ups by up to a factor of 18 compared to KarmaLego and ZMiner, and by up to a factor of 10 compared to VertTIRP. Encouragingly, even in the ASL-GT and pioneer datasets, in which the runtime differences between the methods were smaller, TIRPClo was still more than 1.5 times faster than each of the compared methods. Looking at Fig. 8, it is also quite clear to see that these differences typically increase as the minimum vertical support threshold decreases, and they are expected to widen even more at lower thresholds. Note that in the ST_2 dataset the size of the index of both KarmaLego and ZMiner exceeded the 16GB memory limit. Therefore, only the runtime duration results of TIRPClo and VertTIRP on this dataset are presented in Fig. 8.

The reported improvements in the runtime of TIRPClo have been mainly due to its projection-based mining approach and the effective candidate generation scheme,

which generates only valid and frequent candidates for pattern extensions directly from the data. That is unlike both KarmaLego, ZMiner, and VertTIRP, in which the generation of infrequent candidates cannot be prevented, but only limited based on search space reduction heuristics. The differences between the methods' runtime durations then typically depend on the quality and effectiveness of the respective heuristics.

In KarmaLego, for example, the transitivity property of Allen's temporal relations is used for that purpose, while in ZMiner transitivity is not utilized, but rather it is the indexed frequent two-sized TIRPs based on which new candidates are recursively generated. Looking at Fig. 8, it cannot be confidently determined whether one of these two heuristics is necessarily better than the other, as the runtimes of KarmaLego and ZMiner have been competitive in the majority of datasets. In VertTIRP, a pairing strategy is utilized for a more efficient candidate generation, which sorts the temporal relations to be assessed after employing transitivity, and seems to usually improve runtime compared to KarmaLego and ZMiner.

However, note that the effectiveness of VertTIRP's approach, similar to KarmaLego, is highly dependent on the distribution of temporal relations in the underlying data. That is since the transitivity table returns a varied number of possible relations for different input temporal relations. For example, the *before*, *meets*, and *equals* temporal relations typically return only a single possible relation from the transitivity table, and thus result in an effectively reduced search space. That is unlike the *contains* and *starts* temporal relations, for which multiple possible relations are almost always retrieved from the transitivity table, resulting in only a slightly reduced search space. In TIRPClo, on the other hand, the different temporal relations are seamlessly processed, since they are all implicitly embedded within the sequential representation. These insights may explain the somewhat smaller improvement factors in the runtime of TIRPClo compared to VertTIRP which have been recorded in the ASL-GT and pioneer datasets, in which the *before*, *meets*, and *equals* relations constitute 100% of the frequent pairwise temporal relations at 40–50% of minimum vertical support. In other datasets, e.g., the smart-home and diabetes datasets, the *contains* and *starts* relations typically constitute a considerable portion of about 20–40% of all the frequent pairwise temporal relations at the respective minimum support thresholds, resulting in larger runtime duration differences in favor of TIRPClo.

To further test for statistical significance in the differences between the evaluated methods' runtimes, we also conducted a pairwise post-hoc analysis comparing them. As suggested in Benavoli et al. (2016), comparisons were conducted using a Wilcoxon signed-rank test with Holm's alpha correction (Garcia and Herrera 2008), setting $\alpha = 5\%$. That is, searching for statistical significance in the rankings of the evaluated methods (based on their obtained runtimes) in each configuration of a dataset and a minimum vertical support threshold tested, over all of the tested configurations, which constitute a total sample size of over seventy configurations for significance testing. Thus, complementing the detailed comparisons provided in Fig. 8, which focus on the runtime differences between the methods in each configuration of a dataset and a minimum vertical support threshold, and not on their relative rankings over all the configurations.

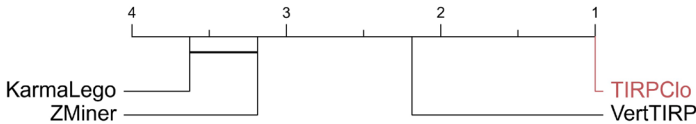


Fig. 9 Critical difference diagram comparing the obtained runtimes of TIRPClo, KarmaLego, ZMiner, and VertTIRP, over all the evaluation datasets in the discovery of the entire set of frequent TIRPs. While a statistically significant improvement is demonstrated in the runtime of TIRPClo compared to each of the three baseline methods, a non-significant difference is observed between ZMiner and KarmaLego

The results of significance testing are summarized in the critical difference diagram (Demšar 2006) shown in Fig. 9, in which the average rank of each method is shown, based on the obtained runtimes, and non-significantly different methods are connected by a thick horizontal line. In this figure, an indeed statistically significant improvement is demonstrated in the obtained runtimes of TIRPClo over each of the compared methods, while there is no significant difference in the runtimes of ZMiner and KarmaLego, which confirms the results presented in Fig. 8.

Figure 10 shows a memory consumption summary of TIRPClo, KarmaLego, ZMiner, and VertTIRP using a logarithmic scale. In this figure, it is demonstrated that TIRPClo's main memory requirements in the benchmark datasets were lower compared to evaluated baseline methods *by a factor of 2 up to more than 10*. That is except for the ST_2 dataset, in which TIRPClo and VertTIRP consumed similar amounts of memory. The observed differences may be explained by the relatively small size of TIRPClo's tieps-index, whose memory complexity can be bounded linearly in the number of STIs in the dataset, as theoretically analyzed in Sect. 3.2. That is in contrast to the more complex indices of KarmaLego and ZMiner, whose memory complexity is quadratic in the number of symbol types in the dataset as well as in the number of STIs per entity. Therefore, when having extreme values for these properties within an input dataset, which is the case in the ST_2 dataset, the memory consumption of their indices explodes, rapidly exceeding the 16GB memory limit even before the beginning of the recursive TIRP mining process.

5.2 Experiment 2: frequent closed TIRP mining

After showing TIRPClo's enhanced performance when mining the entire set of frequent TIRPs, in this experiment the runtime duration and memory consumption of TIRPClo were compared to CCMiner (Chen et al. 2016) in the discovery of only the frequent closed TIRPs. Figure 11 summarizes the runtime duration comparison of the algorithms using a logarithmic scale. In this figure, it is clearly demonstrated that TIRPClo has outperformed CCMiner in all the datasets and already at very high levels of minimum vertical support.

In the ASL, diabetes, smart-home, and ASL-BU datasets, for example, TIRPClo reported speed-ups *by at least a factor of 10 up to more than two orders of a magnitude* compared to CCMiner, already at 50% of minimum vertical support.

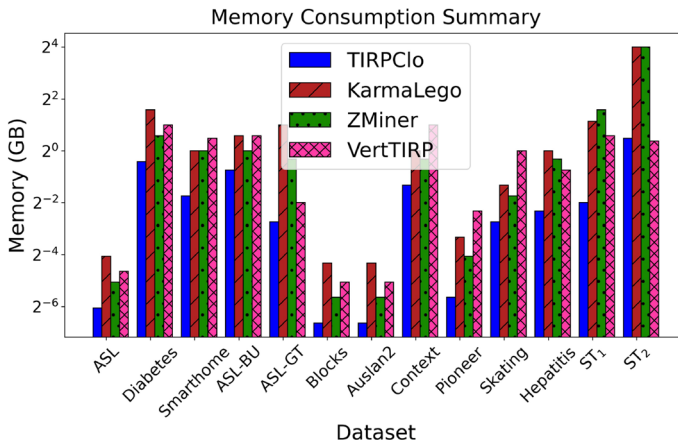


Fig. 10 Memory consumption comparison in the discovery of the entire set of frequent TIRPs. TIRPClo consumed up to 10 times less main memory compared to both KarmaLego, ZMiner, and VertTIRP

Similar improvements were also shown in the rest of the datasets, at relatively high and moderate minimum vertical support thresholds. We strongly hypothesize that these significant runtime speed-ups have mainly stemmed from TIRPClo's use of the tieps-index and the support-indices, due to which no repeated scanning nor copying of the input datasets' records are performed, unlike in CCMiner. In Fig. 12 a critical difference diagram is illustrated, comparing the obtained runtimes of TIRPClo and CCMiner over all the evaluation datasets, in which a statistically significant improvement is demonstrated in favor of TIRPClo.

At last, in Fig. 13 a memory consumption comparison of TIRPClo and CCMiner is presented in the discovery of only the frequent closed TIRPs. First, in all the datasets, both TIRPClo and CCMiner consumed reasonable amounts of main memory. Since CCMiner does not consist of an indexing phase, its memory consumption was expected to be somewhat lower compared to TIRPClo, which was indeed the case in the diabetes, ASL-GT, hepatitis, ST₁, and ST₂ datasets. In the smart-home, blocks, auslan2, and skating datasets, however, TIRPClo reported lower memory measurements, yet without showing a significant difference. That is due to an increased memory usage during the recursive TIRPs discovery process in CCMiner. Overall, looking at Fig. 13 it can be concluded that the memory requirements of the two methods are competitive.

5.3 Experiment 3: completeness in frequent TIRP mining

Beyond the performance evaluation, in this experiment we wanted to empirically substantiate the necessity of the horizontal support discovery for completeness in each of the two TIRP mining tasks addressed in the paper.

That is, both in the discovery of the entire set of frequent TIRPs, and in the discovery of only the frequent closed TIRPs. For that, we first compared the number of

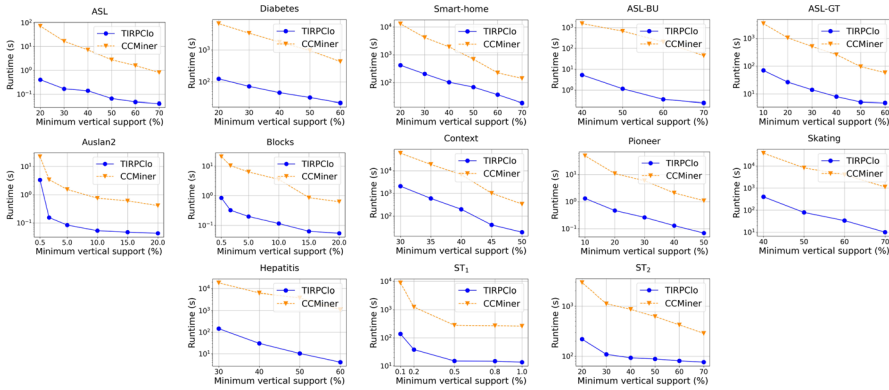


Fig. 11 Runtime comparison in the discovery of only the frequent closed TIRPs. TIRPClo was one to more than two orders of a magnitude faster than CCMiner in all the datasets

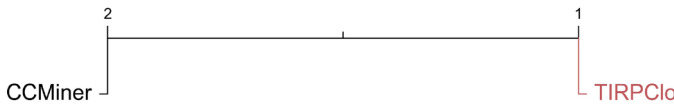


Fig. 12 Critical difference diagram comparing the obtained runtimes of TIRPClo and CCMiner over all the evaluation datasets in the discovery of only the frequent closed TIRPs, in which a statistically significant improvement is demonstrated in favor of TIRPClo

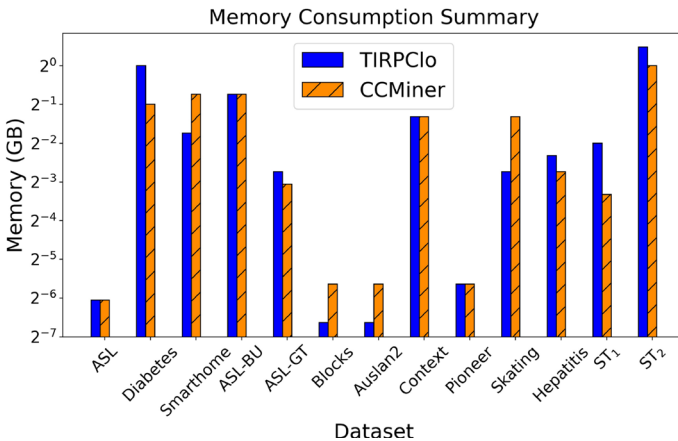


Fig. 13 Memory consumption comparison in the discovery of only the frequent closed TIRPs. TIRPClo's memory consumption was competitive with the memory consumption of CCMiner

frequent TIRPs that were discovered, either when discovering the horizontal support as made in TIRPClo, or without horizontal support discovery as previously made in Winarko and Roddick (2007), Wu and Chen (2007), Patel et al. (2008), Papapetrou et al. (2009), Chen et al. (2015).

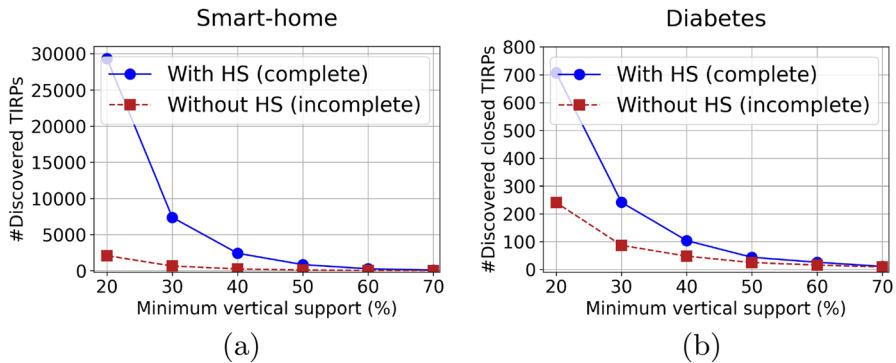


Fig. 14 Comparison of the number of discovered frequent TIRPs with and without the horizontal support (HS) discovery, either when mining the entire set of frequent TIRPs in the smart-home dataset (a), or only the frequent closed TIRPs in the diabetes dataset (b). In both (a) and (b) the complete mining of the frequent TIRPs, which includes horizontal support discovery, has resulted in a substantially larger number of discovered frequent TIRPs compared to the previous, incomplete approach, by factors of 3 up to more than 10

The results when mining the entire set of frequent TIRPs in the smart-home dataset are presented Fig. 14a, in which significant differences have been demonstrated between the two approaches already at considerably high levels of minimum vertical support. Specifically, for minimum vertical support thresholds between 20–40%, *less than 10%* of the frequent TIRPs which had been discovered by TIRPCLO were also discovered without the horizontal support discovery, as previously made. In Fig. 14b a comparison of the number of frequent closed TIRPs which have been discovered in the diabetes dataset is shown, either with or without the horizontal support discovery. In this comparison similar outcomes were observed, as even *less than one third* of the frequent closed TIRPs that had been discovered by TIRPCLO were also discovered without the horizontal support discovery, already at remarkably high levels of minimum vertical support (e.g., 50–60%). In addition, in both Fig. 14a, b a common, yet unsurprising trend was observed according to which the gap in the number of frequent that were discovered by the two approaches significantly widened as the minimum vertical support thresholds decreased.

Furthermore, taking advantage of TIRPCLO's capability to discover all the specific instances of the frequent TIRPs, Fig. 15 presents two additional comparisons of both the mean number of discovered instances of a discovered frequent TIRP (a–b), and the mean horizontal support value of a discovered frequent TIRP within an entity in the dataset (c–d). First, as expected, in Fig. 15a, b it is observed that the mean number of instances of a discovered frequent TIRP has typically increased with the minimum vertical support threshold, either when discovering the TIRPs' horizontal support or not.

However, without the horizontal support discovery, even when a frequent TIRP was indeed discovered, only 5–10% and 15–20% of its true total number of instances (which were discovered by TIRPCLO) had been typically discovered in the smart-home and diabetes datasets respectively. This shortage in the number of discovered

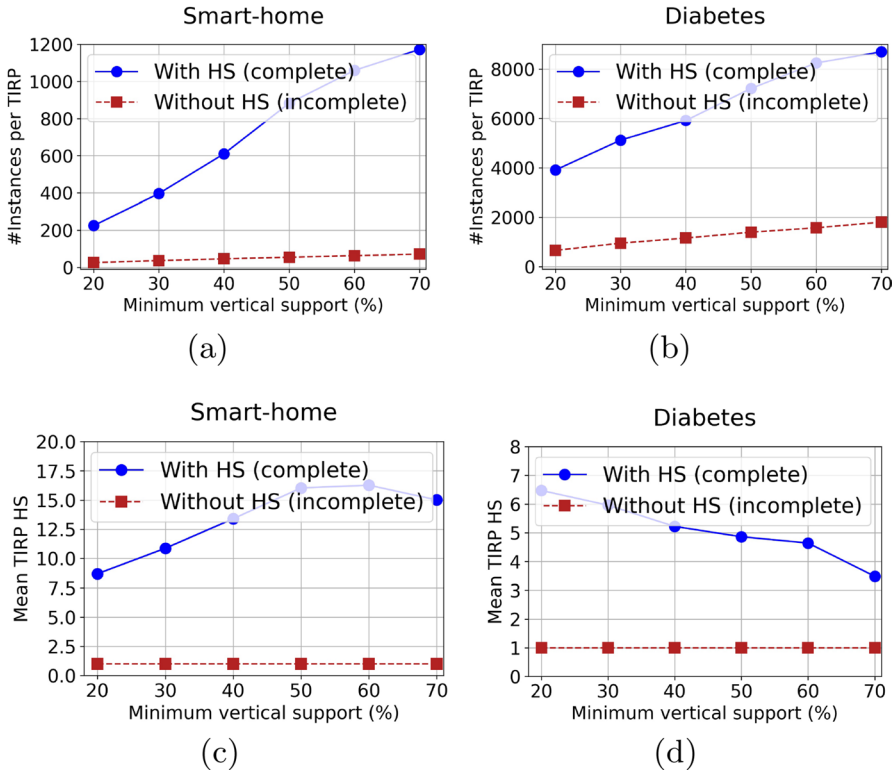


Fig. 15 Comparison of the mean number of discovered instances of a discovered frequent TIRP (a, b), and the mean horizontal support value of a discovered frequent TIRP within an entity in the dataset (c, d), with and without the horizontal support discovery in the smart-home and diabetes datasets. Not discovering the TIRPs’ horizontal support resulted in the discovery of only 5–20% of their true total number of instances, as well as only one out of typically 3–17 instances per entity on average

instances is directly reflected in missing either all the instances of a frequent TIRP, or at least some of them, within the STIs series of indeed supporting entities. The latter case is emphasized in Fig. 15c, d, in which the true mean horizontal support value of a discovered TIRP within an entity’s series of STIs is shown to vary from 8 to 17 in the smart-home dataset and 3–7 in the diabetes dataset; in correlation with the datasets’ mean horizontal support value of a symbol shown in Table 1. That is, while always having, by definition, only a single discovered instance of a TIRP within a supporting entity when the TIRPs’ horizontal support is not discovered. Note that as shown in Fig. 15c, d, the mean horizontal support of a discovered frequent TIRP may either increase or decrease when increasing the minimum vertical

support threshold. That is since the vertical support refers only to the number of supporting entities, and not to the number of instances of the TIRP within them.

Overall, the results obtained in this experiment demonstrate that without the discovery of the TIRPs' horizontal support, a substantial portion of both the frequent TIRPs and their instances are not discovered, which might eventually deteriorate the performance of downstream tasks relying on the output of the frequent TIRPs discovery process. For example, the classification of STIs series based on the frequent TIRPs that they include (Patel et al. 2008; Batal et al. 2009; Moskovitch et al. 2009; Moskovitch and Shahar 2015b; Rebane et al. 2021; Shknevsky et al. 2021). For such downstream task, not discovering the horizontal support is likely to result in misclassification of STIs series, due to either (1) completely missing the discovery of valuable frequent TIRPs, (2) missing the discovery of all the instances of frequent TIRPs within STIs series in which they indeed occur, or (3) discovering only one out of many instances of frequent TIRPs within each series of STIs, resulting in miscalculation of features such as the mean duration of a TIRP, which has been an important feature for correct classification in several previous works (Moskovitch and Shahar 2015b; Shknevsky et al. 2021). Further conducting a comprehensive qualitative evaluation of the discovered TIRPs in various such scenarios is out of the scope of this paper, while it is of course of interest for future work.

5.4 Experiment 4: entire frequent TIRP mining versus frequent closed TIRP mining using TIRPCLO

In this experiment we wanted to empirically evaluate major pros and cons of applying TIRPCLO for each of the two TIRP mining tasks that the algorithm performs, focusing on the runtime duration of TIRPCLO and on the reduction in the number of discovered frequent closed TIRPs compared to the entire set of frequent TIRPs. The results of running TIRPCLO for each of the two TIRP mining tasks on the CT₁ and CT₂ datasets are summarized in Fig. 16. First, in both datasets it is observed that the discovery of only the frequent closed TIRPs has typically resulted in a much more compact output of frequent TIRPs compared to the discovery of the entire set of frequent TIRPs, showing at least a 40% reduction (a–b). In addition, larger differences in the number of discovered frequent TIRPs were reported for lower minimum vertical support thresholds. That is primarily due to long closed TIRPs of various frequencies, which contain an exponential number (in the closed TIRPs' lengths) of sub-TIRPs that are not closed.

In terms of the runtime duration of TIRPCLO, however, a different behavior is observed in Fig. 16c–d) when running the algorithm on each of the two datasets in hand. While in the CT₁ dataset closed TIRPs discovery was faster at higher levels of minimum vertical support but somewhat slower at lower thresholds, in the CT₂ dataset, on the other hand, the discovery of the entire set of frequent TIRPs was faster for all the minimum vertical support thresholds assessed. This outcome, especially in the CT₂ dataset, might seem surprising given the reduction in the number of discovered closed TIRPs compared to the entire set of frequent TIRPs shown in Fig. 16b. However, it could be indeed explained by the additional computational cost of the

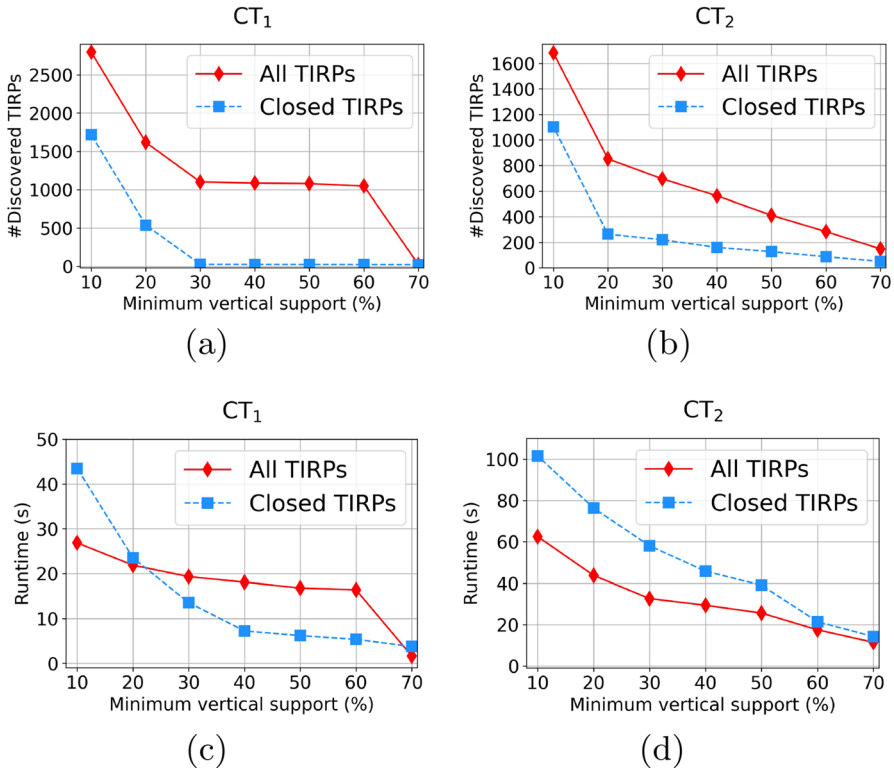


Fig. 16 Entire frequent TIRP mining versus frequent closed TIRP mining using TIRPCLO. Runtimes in the CT₁ and CT₂ datasets are shown at the bottom, while the numbers of frequent TIRPs that have been discovered in each dataset are shown at the top. In both datasets closed TIRPs discovery resulted in a much smaller number of discovered frequent TIRPs, showing at least a 40% reduction. However, in the CT₂ dataset the discovery of all the frequent TIRPs was still faster due to the additional computational cost of TIRPCLO’s closure-checking employed only for closed TIRP mining, while in the CT₁ dataset closed TIRPs discovery was faster at high and moderate minimum vertical support thresholds, due to more effective pruning of TIRPs that are not closed early during the mining process

closure-checking performed in TIRPCLO only when applied for closed TIRP mining, which, in this dataset, outclassed the gains from the early pruning of the TIRPs that are not closed during the mining process.

5.5 Experiment 5: maximal gap analysis

In this experiment, we evaluated the trade-off between TIRPCLO’s runtime duration and the number of discovered frequent TIRPs when changing the value of the maximal gap parameter. The results of running the algorithm on the ASL, diabetes, smart-home, context, and pioneer datasets are summarized in Fig. 17. At the top of the figure the number of frequent TIRPs discovered by TIRPCLO is shown, while at the bottom the respective runtimes are presented, depending on the value of the maximal gap for several minimum vertical support thresholds.

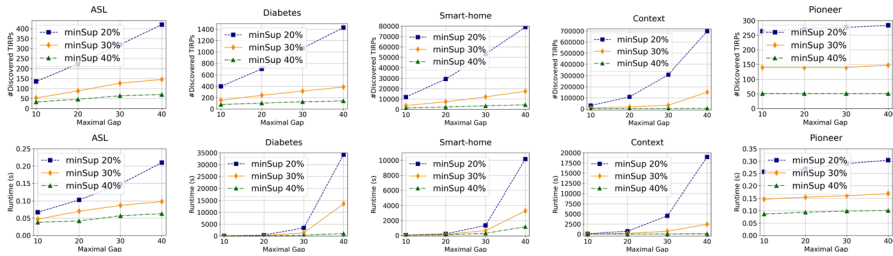


Fig. 17 Maximal gap analysis. At the top the total numbers of discovered frequent TIRPs are presented, while at the bottom the respective runtimes of TIRPCLO are shown, depending on the value of the maximal gap for several minimum vertical support thresholds. For a fixed value of minimum vertical support, larger maximal gap values typically led not only to an increased number of discovered frequent TIRPs, but also to much longer runtimes, demonstrating the trade-off between the two aspects

First, as expected, given a fixed minimum vertical support threshold, increasing the value of the maximal gap parameter typically resulted in a larger number of discovered frequent TIRPs, as well as in longer runtimes. However, while in the majority of datasets increasing the maximal gap value by 10 time-units resulted in several tens of percentages of additionally discovered frequent TIRPs, in the pioneer dataset only a slight change has been recorded. This could be explained by the distribution of time durations of the *before* temporal relation in each dataset, i.e., the time gaps between non-overlapping pairs of STIs, depicted in Fig. 18 in Appendix B. In the pioneer dataset about 80% of these gaps were even below the initial maximal gap of 10. Thus, not showing a significant difference in neither the number of discovered frequent TIRPs nor in runtime when increasing the maximal gap value beyond it. That is unlike the rest of the datasets, in which the 80th percentile was always beyond 20.

In these datasets, it is also important to highlight that increasing the value of the maximal gap typically resulted not only in an increased number of discovered frequent TIRPs, but also in much longer runtimes. For example, in the smart-home dataset, due to increasing the maximal gap value by 10 time-units, up to 150% more frequent TIRPs were discovered in the cost of 2–7 times longer runtimes. These outcomes demonstrate the trade-off between the number of discovered frequent TIRPs (and consequently, their potential degree of expressiveness of the relevant temporal information in the underlying data) and the computational aspect, which may play a key role when determining the desirable value of the maximal gap parameter as we elaborate in the Discussion Sect. 6.

Finally, note that even if we keep on increasing the maximal gap to much larger values, we might still discover more and more frequent TIRPs. That is although they are potentially meaningless, due to the extremely large time gaps between their STIs. In future work, we plan on conducting a qualitative analysis of the discovered TIRPs depending on the maximal gap, aiming at minimizing the computational cost of the TIRP mining process while preserving the majority of temporal information of the discovered TIRPs, through controlling the value of the maximal gap.

6 Discussion

In this work we introduced `TIRPClo`—an efficient algorithm for the complete mining of either the entire set of frequent TIRPs or only the frequent closed TIRPs. `TIRPClo` proposes a novel non-ambiguous transformation method from STIs series into sequential representation, a memory-efficient index, and a new complete projection method; due to which it is the first complete algorithm for frequent closed TIRP mining, and the first sequence-based complete algorithm for mining the entire set of frequent TIRPs. The performance evaluation benchmarked `TIRPClo` on over a dozen datasets against the best performing methods in each of the two TIRP mining tasks that the algorithm performs, demonstrating significant runtime speed-ups while at the same time maintaining lower memory measurements.

Beyond performance, the paper also provided an in-depth analysis of the common completeness problem in the majority of previous methods for frequent TIRP mining, due to not entirely discovering the TIRPs' horizontal support. The analysis included a theoretical demonstration as well as a quantitative assessment on real-world data showing that the horizontal support discovery is crucial for both the complete discovery of the entire set of frequent TIRPs, and the complete discovery of only the frequent closed TIRPs. Otherwise, a non-negligible number of frequent TIRPs, and their instances, are missed, which is much likely to deteriorate the performance of downstream tasks relying on the output of the frequent TIRPs discovery process, e.g., clustering or classification of STIs series.

In addition, we compared `TIRPClo`'s discovery of the entire set of frequent TIRPs to the discovery of only the frequent closed TIRPs. Since the complete set of frequent TIRPs can be revealed based on the more compact set of only the frequent closed TIRPs, the knowledge one can extract from the output of each of the two TIRP mining tasks is identical. In this paper's evaluation, although quite a considerable number of frequent TIRPs that are not closed had been pruned during the frequent closed TIRPs discovery process, the discovery of all the frequent TIRPs was still usually faster. This outcome demonstrates the potential in the discovery of the frequent closed TIRPs, which significantly shrinks the output of the algorithm, alongside the possibly higher computational costs due to the applied closure-checking.

At last, we also demonstrated the trade-off between `TIRPClo`'s runtime duration and the total number of discovered frequent TIRPs when changing the value of the maximal gap parameter, showing that increasing the maximal gap typically results not only in a larger number of discovered frequent TIRPs, but also in much longer runtimes. Therefore, when determining the desirable value of the maximal gap, there are several aspects to consider.

First, is the maximal relevant time duration among two STIs, which is domain-dependent and can sometimes be estimated by a domain expert. Second, is the computational aspect. Since the complexity of the TIRPs discovery process increases with the value of the maximal gap, it may be desirable to limit its value according to the available computing time and resources. However, one should also bear in mind that setting a too small maximal gap value is likely to result in much fewer frequent

TIRPs, which might fail capturing all the temporal information in the underlying data relevant for a specific downstream task.

Finally, as demonstrated in experiments 4–5, both the output and the performance of TIRPCLO heavily depend on its execution configuration (i.e., the desired TIRP mining task and the set of parameters values—e.g., minimum vertical support, maximal gap). Thus, for future work, we would like to learn an execution configuration, or a small set of configurations, which fit best for a given real-world TIRP mining application. That is to supply guidelines for application-customized execution configurations. In addition, we would also like to design TIRPCLO to enable parallel processing within one machine or distributed processing over multiple machines, for the efficient processing of significantly higher data volumes.

Appendix A: Real-world datasets

Detailed information is provided on the real-world datasets which have been used to evaluate the proposed TIRPCLO algorithm (datasets 1–11 in Table 1).

- *American Sign Language (ASL)* The dataset was created by the National Center for Sign Language and Gesture Resources at Boston University (Papapetrou et al. 2009). It consists of a collection of 884 utterances, in which each utterance associates a segment of video with a detailed transcription and contains several ASL gestures and grammatical fields (e.g., eyebrow raised, head tilted forward) occurring over a time interval.
- *Diabetes* The dataset was provided by Moskovitch and Shahar (2015c) as part of a collaboration with Clalit Health Services (Israel's largest HMO). It contains data on 2038 patients having type II diabetes, collected monthly from 2002 to 2007. The dataset contains six variables recorded over time for each patient: hemoglobin-A1c values, blood glucose levels, cholesterol values, and several medications the patients purchased: diabetic (insulin-based) medications, cholesterol reducing statins, and beta blockers.
- *MavLab Smart-home* The dataset was provided by Jakkula and Cook (2011). It contains data from the readings of ninety-nine sensors installed in a computerized apartment, describing the activity of people and various appliances scattered around the apartment.
- *ASL-BU* (Mörchen and Fradkin 2010) The dataset contains STIs which are transcriptions from videos of American Sign Language expressions. An entity's series of STIs represents a single sentence.
- *ASL-GT* (Mörchen and Fradkin 2010) STIs were derived from sixteen dimensional numerical time series with features extracted from videos of American Sign Language expressions. The dataset includes a larger number of entities' STIs series and a smaller number of symbol types compared to the previous dataset.
- *Auslan2* (Mörchen and Fradkin 2010) STIs were derived from the publicly available Australian Sign Language dataset in the UCI repository. An entity's series of STIs represents a single word.

- *Blocks* (Mörchen and Fradkin 2010) Contains STIs which represent visual primitives drawn from videos of a human hand stacking colored blocks. An entity's STIs series represents one of eight different scenarios including either atomic actions (e.g., *move-right*) or complete scenarios (e.g., *assemble*).
- *Context* (Mörchen and Fradkin 2010) STIs were derived from categorical and numerical data that describe the context of a mobile device carried by humans in different situations. An entity's series of STIs represents one of five scenarios (e.g., *meeting* or *street*).
- *Pioneer* (Mörchen and Fradkin 2010) STIs were derived from the Pioneer-1 dataset in the UCI repository, which contains data collected from sensor readings of the Pioneer-1 mobile robot. An entity's STIs series describes one of the robot's three moving scenarios—either *gripper*, *move* or *turn*.
- *Skating* (Mörchen and Fradkin 2010) The dataset contains STIs derived from fourteen dimensional numerical time series, which describe the muscle activity and leg position of six professional In-Line Speed Skaters during controlled tests. An entity's series of STIs describes a complete movement cycle.
- *Hepatitis* (Patel et al. 2008) The dataset contains STIs describing tests conducted to patients suffering from either Hepatitis B or C over a time period of 10 years. An entity's STIs series represents the tests conducted to a single patient.

Appendix B: Distribution of time gaps between non-overlapping STIs

Figure 18 shows the distribution of time durations of the *before* temporal relation, i.e., the time gaps between non-overlapping pairs of STIs, in the ASL, diabetes, smart-home, context, and pioneer datasets which have been used for the maximal gap analysis in experiment 5.

Appendix C: Worst-case complexity analysis of sequential pattern mining

In Sect. 3.6.2, a simplified worst-case assessment of the complexity of the proposed TIRPCLO algorithm, as a representative of sequence-based TIRP mining methods, was provided. In this appendix, we follow similar notations to those introduced in Sect. 3.6.2, and also concisely assess the complexity of the more basic task of sequential pattern mining.

Assume:

S —number of event-types

N —total number of events in the dataset

n —maximal number of events within an entities' event-sequence

L —maximal number of events within a frequent sequential pattern

In sequential pattern mining, both the input data and the discovered patterns only consist of time point-based events. Thus, the discovery of a k -sized sequential

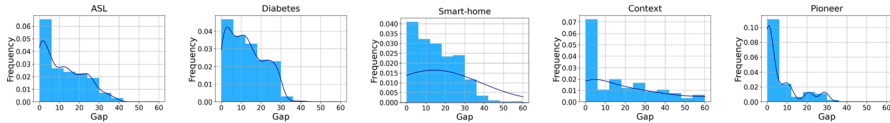


Fig. 18 Histograms showing the distribution of time durations of the *before* temporal relation, i.e., the time gaps between non-overlapping pairs of STIs in the datasets used for the maximal gap analysis in experiment 5

pattern requires k pattern extension steps, i.e., by a single event at a time, while having at most S candidates generated in each step, given a total of S event-types or symbols. That is unlike in sequence-based TIRP mining, where the input STIs data are broken into their start and finish tieps, which consequently doubles the number of pattern extension steps required to discover a k -sized TIRP as well as the maximal number of generated candidates, as described in Sect. 3.6.2. Hence, including the initial sorting of entities' event-sequences, the overall time complexity of the more basic task of sequential pattern mining can be typically assessed by $O(N \cdot S^L + n \cdot \log(n))$ in the worst-case.

Acknowledgements The authors wish to thank Prof. Panagiotis Papapetrou and Prof. Diane J Cook for providing datasets for the evaluation. This research was partially funded by a grant of the Israeli Ministry of Science and Technology (Grant 8760441). Omer Harel was funded also by the Darom-Lachish scholarship of Kreitman School of Advanced Graduate Studies at Ben Gurion University (No. 1955129).

Funding This research was partially funded by a grant of the Israeli Ministry of Science and Technology (grant 8760441). Omer Harel was also funded by the Darom-Lachish scholarship of Kreitman School of Advanced Graduate Studies at Ben Gurion University (No. 1955129).

Data availability To allow the complete reproducibility of our experimental results and contribute to future research in the field of frequent TIRP mining, all the real-world and synthetic datasets used in the paper, as well as our synthetic datasets generator, were made publicly available through the online repository referenced in the Introduction Sect. 1.

Code availability The source code of the TIRPCLO algorithm was also made publicly available through the online repository referenced in the Introduction Sect. 1.

Declarations

Conflicts of interest The authors have no relevant financial or non-financial interests to disclose.

References

- Allen JF (1983) Maintaining knowledge about temporal intervals. *Commun ACM* 26(11):832–843. <https://doi.org/10.1145/182.358434>
- Ayres J, Flannick J, Gehrke J, et al (2002) Sequential pattern mining using a bitmap representation. In: Proceedings of the Eighth ACM SIGKDD international conference on knowledge discovery and data mining. Association for Computing Machinery, New York, NY, USA, KDD '02, pp 429–435, <https://doi.org/10.1145/775047.775109>
- Batal I, Sacchi L, Bellazzi R, et al (2009) A temporal abstraction framework for classifying clinical temporal data. In: AMIA Annual Symposium Proceedings, vol 2009. American Medical Informatics Association, Rockville, MD, p 29

- Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? *J Mach Learn Res* 17(1):152–161
- Chang L, Wang T, Yang D, et al (2008) Seqstream: mining closed sequential patterns over stream sliding windows. In: 2008 Eighth IEEE International Conference on Data Mining. IEEE Computer Society, Washington, DC, USA, ICDM '08, pp 83–92. <https://doi.org/10.1109/ICDM.2008.36>
- Chen YC, Peng WC, Lee SY (2015) Mining temporal patterns in time interval-based data. *IEEE Trans Knowl Data Eng* 27(12):3318–3331. <https://doi.org/10.1109/TKDE.2015.2454515>
- Chen YC, Weng JTY, Hui L (2016) A novel algorithm for mining closed temporal patterns from interval-based data. *Knowl Inf Syst* 46(1):151–183. <https://doi.org/10.1007/s10115-014-0815-2>
- Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res* 7:1–30
- Ezeife CI, Lu Y, Liu Y (2005) Plwap sequential mining: open source code. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations. Association for Computing Machinery, New York, NY, USA, OSDM '05, pp 26–35. <https://doi.org/10.1145/1133905.1133910>
- Fournier-Viger P, Lin JCW, Kiran RU et al (2017) A survey of sequential pattern mining. *Data Sci Pattern Recogn* 1(1):54–77
- Fumarola F, Lanotte PF, Ceci M et al (2016) Clofast: closed sequential pattern mining using sparse and vertical id-lists. *Knowl Inf Syst* 48(2):429–463. <https://doi.org/10.1007/s10115-015-0884-x>
- García S, Herrera F (2008) An extension on " statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *J Mach Learn Res* 9(12):2677
- Gomariz A, Campos M, Marin R, et al (2013) Clasp: An efficient algorithm for mining frequent closed sequences. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Berlin, pp 50–61. https://doi.org/10.1007/978-3-642-37453-1_5
- Han J, Pei J, Mortazavi-Asl B, et al (2000) Freespan: Frequent pattern-projected sequential pattern mining. In: Proceedings of the Sixth ACM SIGKDD international conference on knowledge discovery and data mining. Association for Computing Machinery, New York, NY, USA, KDD '00, pp 355–359. <https://doi.org/10.1145/347090.347167>
- Han J, Pei J, Mortazavi-Asl B, et al (2001) Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: proceedings of the 17th international conference on data engineering. IEEE Computer Society, Washington, DC, USA, pp 215–224
- Harel OD, Moskovitch R (2021) Complete closed time intervals-related patterns mining. In: proceedings of the 35th AAAI conference on artificial intelligence. AAAI Press, Palo Alto, CA
- Höppner F (2001) Learning temporal rules from state sequences. In: IJCAI Workshop on Learning from Temporal and Spatial Data. Citeseer
- Höppner F (2002) Time series abstraction methods: a survey. *Informatik bewegt: Informatik 2002–32 Jahrestagung der Gesellschaft für Informatik ev (GI)*
- Huang JW, Jaysawal BP, Chen KY et al (2019) Mining frequent and top-k high utility time interval-based events with duration patterns. *Knowl Inf Syst* 61(3):1331–1359. <https://doi.org/10.1007/s10115-019-01333-6>
- Huang KY, Chang CH, Tung JH, et al (2006) Cobra: Closed sequential pattern mining using bi-phase reduction approach. In: International Conference on data warehousing and knowledge discovery. Springer, Berlin, pp 280–291. https://doi.org/10.1007/11823728_27
- Hui L, Chen YC, Weng JTY et al (2016) Incremental mining of temporal patterns in interval-based database. *Knowl Inf Syst* 46(2):423–448. <https://doi.org/10.1007/s10115-015-0828-5>
- Itzhak N, Jaroszewicz S, Moskovitch R (2023) Continuously predicting a time intervals based pattern completion towards event prediction. *PAKDD*, Osaka, Japan
- Jakkula VR, Cook DJ (2011) Detecting anomalous sensor events in smart home data for enhancing the living experience. *Artif Intell Smart Living* 11(201):1
- Kam PS, Fu AWC (2000) Discovering temporal patterns for interval-based events. In: International conference on data warehousing and knowledge discovery. Springer, Berlin, pp 317–326. https://doi.org/10.1007/3-540-44466-1_32
- Kostakis A, Gionis A (2017) On mining temporal patterns in dynamic graphs, and other unrelated problems. In: International conference on complex networks and their applications. Springer, Berlin, pp 516–527. https://doi.org/10.1007/978-3-319-72150-7_42
- Kostakis O, Papapetrou P, Hollmén J (2011) Artemis: Assessing the similarity of event-interval sequences. In: Joint European conference on machine learning and knowledge discovery in databases, Springer, pp 229–244. https://doi.org/10.1007/978-3-642-23783-6_15

- Kotsifakos A, Papapetrou P, Athitsos V (2013) ibsm: interval-based sequence matching. in: proceedings of the 2013 siam International Conference on Data Mining, SIAM, pp 596–604, <https://doi.org/10.1137/1.9781611972832.66>
- Lavrac N, Keravnou E, Zupan B (2000) Intelligent data analysis in medicine. *Encycl Comput Sci Technol* 42(9):113–157
- Lee Z, Lindgren T, Papapetrou P (2020) Z-miner: An efficient method for mining frequent arrangements of event intervals. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining. Association for Computing Machinery, New York, NY, USA, KDD '20, pp 524–534, <https://doi.org/10.1145/3394486.3403095>
- Lin J, Keogh E, Lonardi S, et al (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery. Association for Computing Machinery, New York, NY, USA, DMKD '03, pp 2–11, <https://doi.org/10.1145/882082.882086>
- Lin MY, Lee SY (2002) Fast discovery of sequential patterns by memory indexing. In: International conference on data warehousing and knowledge discovery. Springer, Berlin, pp 150–160, https://doi.org/10.1007/3-540-46145-0_15
- Mabroukeh NR, Ezeife CI (2010) A taxonomy of sequential pattern mining algorithms. *ACM Comput Surv* 43(1):1–41. <https://doi.org/10.1145/1824795.1824798>
- Mirbagheri SM, Hamilton HJ (2020a) High-utility interval-based sequences. In: International Conference on big data analytics and knowledge discovery, Springer, pp 107–121, https://doi.org/10.1007/978-3-030-59065-9_9
- Mirbagheri SM, Hamilton HJ (2020b) Similarity matching of temporal event-interval sequences. In: Canadian conference on artificial intelligence, Springer, pp 420–425, https://doi.org/10.1007/978-3-030-47358-7_43
- Mirbagheri SM, Hamilton HJ (2021) Mining high utility patterns in interval-based event sequences. *Data Knowl Eng* 135(101):924. <https://doi.org/10.1016/j.datak.2021.101924>
- Mörchen F, Fradkin D (2010) Robust mining of time intervals with semi-interval partial order patterns. In: Proceedings of the 2010 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp 315–326, <https://doi.org/10.1137/1.9781611972801.28>
- Mörchen F, Ultsch A (2005) Optimizing time series discretization for knowledge discovery. In: Proceedings of the Eleventh ACM SIGKDD international conference on knowledge discovery in data mining. Association for Computing Machinery, New York, NY, USA, KDD '05, pp 660–665, <https://doi.org/10.1145/1081870.1081953>
- Mordvanyuk N, López B, Bifet A (2021) verttirp: Robust and efficient vertical frequent time interval-related pattern mining. *Expert Syst Appl* 168(114):276. <https://doi.org/10.1016/j.eswa.2020.114276>
- Mordvanyuk N, López B, Bifet A (2022) Ta4l: efficient temporal abstraction of multivariate time series. *Knowl-Based Syst* 244(108):554. <https://doi.org/10.1016/j.knosys.2022.108554>
- Moskovitch R, Shahar Y (2015a) Classification-driven temporal discretization of multivariate time series. *Data Min Knowl Disc* 29(4):871–913. <https://doi.org/10.1007/s10618-014-0380-z>
- Moskovitch R, Shahar Y (2015b) Classification of multivariate time series via temporal abstraction and time intervals mining. *Knowl Inf Syst* 45(1):35–74. <https://doi.org/10.1007/s10115-014-0784-5>
- Moskovitch R, Shahar Y (2015) Fast time intervals mining using the transitivity of temporal relations. *Knowl Inf Syst* 42(1):21–48. <https://doi.org/10.1007/s10115-013-0707-x>
- Moskovitch R, Peek N, Shahar Y (2009) Classification of ICU patients via temporal abstraction and temporal patterns mining. Notes of the intelligent data analysis in medicine and pharmacology (IDAMAP 2009) Workshop. American Medical Informatics Association, Verona, Italy, pp 35–40
- Moskovitch R, Walsh C, Wang F, et al (2015) Outcomes prediction via time intervals related patterns. In: 2015 IEEE international conference on data mining. IEEE Computer Society, Washington, DC, USA, pp 919–924, <https://doi.org/10.1109/ICDM.2015.143>
- Novitski P, Cohen CM, Karasik A, et al (2020) All-cause mortality prediction in t2d patients. In: International conference on artificial intelligence in medicine. Springer, Berlin, pp 3–13, https://doi.org/10.1007/978-3-030-59137-3_1
- Papapetrou P, Kollios G, Sclaroff S et al (2009) Mining frequent arrangements of temporal intervals. *Knowl Inf Syst* 21(2):133. <https://doi.org/10.1007/s10115-009-0196-0>

- Patel D, Hsu W, Lee ML (2008) Mining relationships among interval-based events for classification. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. Association for Computing Machinery, New York, NY, USA, SIGMOD '08, pp 393–404, <https://doi.org/10.1145/1376616.1376658>
- Pei J, Han J, Mortazavi-Asl B, et al (2000) Mining access patterns efficiently from web logs. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, pp 396–407, https://doi.org/10.1007/3-540-45571-X_47
- Rebane J, Karlsson I, Bornemann L et al (2021) Smile: a feature-based temporal abstraction framework for event-interval sequence classification. *Data Min Knowl Disc* 35(1):372–399. <https://doi.org/10.1007/s10618-020-00719-3>
- Sacchi L, Larizza C, Combi C et al (2007) Data mining with temporal abstractions: Learning rules from time series. *Data Min Knowl Disc* 15(2):217–247. <https://doi.org/10.1007/s10618-007-0077-7>
- Shahar Y (1997) A framework for knowledge-based temporal abstraction. *Artif Intell* 90(1–2):79–133. [https://doi.org/10.1016/S0004-3702\(96\)00025-2](https://doi.org/10.1016/S0004-3702(96)00025-2)
- Sharma AK, Patel D (2018) Stipa: A memory efficient technique for interval pattern discovery. In: 2018 IEEE International conference on big data (Big Data). IEEE Computer Society, Washington, DC, USA, pp 1767–1776, <https://doi.org/10.1109/BigData.2018.8622421>
- Shknevisky A, Shahar Y, Moskovitch R (2021) The semantic adjacency criterion in time intervals mining. arXiv preprint [arXiv:2101.03842](https://arxiv.org/abs/2101.03842)
- Srikant R, Agrawal R (1996) Mining sequential patterns: Generalizations and performance improvements. In: International conference on extending database technology. Springer, Berlin, pp 1–17, <https://doi.org/10.1007/BFb0014140>
- Tzvetkov P, Yan X, Han J (2005) Tsp: mining top-k closed sequential patterns. *Knowl Inf Syst* 7(4):438–457. <https://doi.org/10.1007/s10115-004-0175-4>
- Villafane R, Hua KA, Tran D et al (2000) Knowledge discovery from series of interval events. *J Intell Inf Syst* 15(1):71–89. <https://doi.org/10.1023/A:1008781812242>
- Wang J, Han J (2004) Bide: Efficient mining of frequent closed sequences. In: Proceedings. 20th international conference on data engineering. IEEE Computer Society, Washington, DC, USA, pp 79–90, <https://doi.org/10.1109/ICDE.2004.1319986>
- Winarko E, Roddick JF (2007) Armada: an algorithm for discovering richer relative temporal association rules from interval-based data. *Data Knowl Eng* 63(1):76–90. <https://doi.org/10.1016/j.datak.2006.10.009>
- Wu SY, Chen YL (2007) Mining nonambiguous temporal patterns for interval-based events. *IEEE Trans Knowl Data Eng* 19(6):742–758. <https://doi.org/10.1109/TKDE.2007.190613>
- Yan X, Han J, Afshar R (2003) Clospan: mining: closed sequential patterns in large datasets. In: Proceedings of the 2003 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp 166–177, <https://doi.org/10.1137/1.9781611972733.15>
- Yang CW, Jaysawal BP, Huang JW (2017) Subsequence search considering duration and relations of events in time interval-based events sequences. In: 2017 IEEE International conference on data science and advanced analytics (DSAA), IEEE, pp 293–302, <https://doi.org/10.1109/DSAA.2017.47>
- Yang Z, Wang Y, Kitsuregawa M (2007) Lapin: effective sequential pattern mining algorithms by last position induction for dense databases. In: International conference on database systems for advanced applications. Springer, Berlin, pp 1020–1023, https://doi.org/10.1007/978-3-540-71703-4_95
- Zaki MJ (2001) Spade: an efficient algorithm for mining frequent sequences. *Mach Learn* 42(1–2):31–60. <https://doi.org/10.1023/A:1007652502315>
- Zhang J, Wang Y, Yang D (2015) Ccspan: mining closed contiguous sequential patterns. *Knowl-Based Syst* 89:1–13. <https://doi.org/10.1016/j.knosys.2015.06.014>
- Zhao Q, Bhowmick SS (2003) Sequential pattern mining: a survey. ITechnical Report CAIS Nayang Technological University Singapore 1(26):135

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Omer Harel¹  · Robert Moskovitch^{1,2}

✉ Omer Harel
omerdavi@post.bgu.ac.il

Robert Moskovitch
robertmo@bgu.ac.il

¹ Software and Information Systems Engineering, Ben Gurion University of the Negev,
Be'er Sheva, Israel

² Population Health Science and Policy, Icahn School of Medicine at Mount Sinai, New York,
NY, USA