Frequent Pattern Mining in Continuous-Time Temporal Networks

Ali Jazayeri and Christopher C. Yang

***Abstract*—Networks are used as highly expressive tools in different disciplines. In recent years, the analysis and mining of temporal networks have attracted substantial attention. Frequent pattern mining is considered an essential task in the network science literature. In addition to the numerous applications, the investigation of frequent pattern mining in networks directly impacts other analytical approaches, such as clustering, quasi-clique and clique mining, and link prediction. In nearly all the algorithms proposed for frequent pattern mining in temporal networks, the networks are represented as sequences of static networks. Then, the inter- or intra-network patterns are mined. This type of representation imposes a computation-expressiveness trade-off to the mining problem. In this paper, we propose a novel representation that can preserve the temporal aspects of the network losslessly. Then, we introduce the concept of constrained interval graphs (CIGs). Next, we develop a series of algorithms for mining the complete set of frequent temporal patterns in a temporal network data set. We also consider four different definitions of isomorphism for accommodating minor variations in temporal data of networks. Implementing the algorithm for three real-world data sets proves the practicality of the proposed approach and its capability to discover unknown patterns in various settings.**

***Index Terms*—Continuous-time networks, frequent subgraphs, pattern mining, temporal networks.**

Nomenclature

$\|A\|$	Cardinality of a given set A
ϵ	User-defined support threshold.
$\mathbb{W}$	Time window of temporal network.
$\mathscr{I}$	Time interval.
$\mathscr{L}$	List of edges of temporal network.
$\mathscr{M}$	Map of vertices in V_N to their associated interval trees.
agg_w	Aggregation window.
CIG	Constrained Interval Graph.
cl	Canonical labeling.
DAG	Directed Acyclic Graph.
DS	Data set of temporal networks.
DS'	Data set of CIGs associated with temopral networks.
E_i	Set of edges in network i
$freq(s)$	Frequency of subgraph s
IG	Interval Graph.
IT	Interval Tree.
M	Map of vertices in V_{CIG} to pair of vertices in V_N
N	Temporal network.
s	Subgraph.
V_i	Set of vertices in network i

Manuscript received 21 May 2021; revised 16 August 2023; accepted 5 October 2023. Date of publication 16 October 2023; date of current version 5 December 2023. This work was supported in part by the National Science Foundation under Grants IIS-1741306 and IIS-2235548, and in part by the Department of Defense under Grants W81XWH-22-1-0581 and W81XWH-22-1-0582. This material is based upon work supported by (while serving at) the National Science Foundation. Recommended for acceptance by L. Li. *(Corresponding author: Ali Jazayeri.)*

The authors are with the College of Computing and Informatics, Drexel University, Philadelphia, PA 19104 USA (e-mail: ali.jazayeri@drexel.edu; chris.yang@drexel.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2023.3324799, provided by the authors.

Digital Object Identifier 10.1109/TPAMI.2023.3324799

I. Introduction

NETWORKS have been extensively adopted for modeling systems where in addition to the systems' components, the inter-component interactions may provide deeper insights into the systems' behavior. Networks, with a long history of applications [1], [2], are used as highly expressive tools for system modeling in different domains [3].

Among different analytical and mining techniques proposed for network research, the mining of frequent network patterns has an essential place [4]. The underlying idea behind this problem is that the recurring patterns observed more frequently may represent essential characteristics of the system that networks represent [5], [6]. However, the implementation of network mining for identifying frequent patterns is a non-trivial and computationally costly task. The main reason is the requirement to verify the graph and subgraph isomorphism in different iterations of the frequent pattern mining process.

Furthermore, in many applications, the temporality of the systems should be included in the modeling effort. It is shown that when the time scale of the changes in the system is comparable, using dynamic and time-varying network models can inform the identification of important components more effectively [7], [8]. One approach is to represent the temporal aspects of the system as attributes of the vertices and edges of the corresponding network. However, this approach might obscure some of the temporal information [9], [10]. Besides, some of the well-defined metrics and concepts in static networks, such as distance, diameter, centralities, and connectivity, have been differently defined and interpreted for temporal networks. Therefore, aggregating and representing temporal aspects of

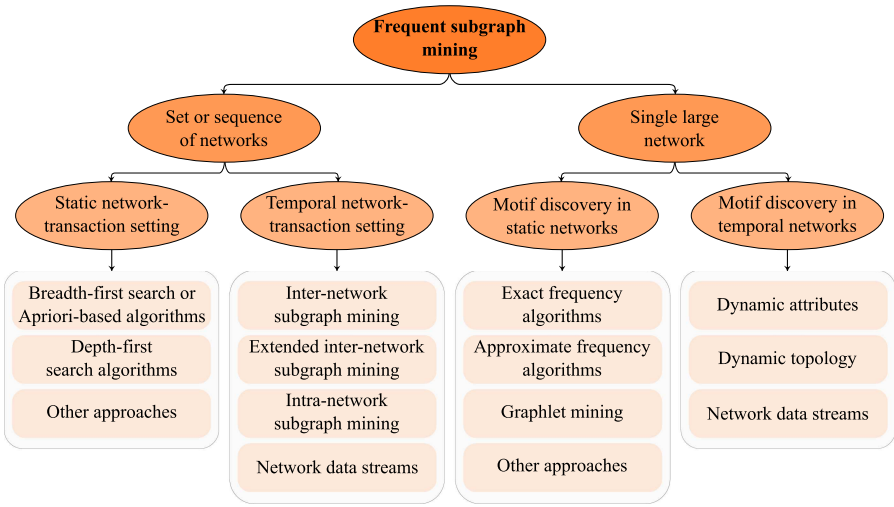

Fig. 1. Taxonomy of frequent subgraph mining algorithms proposed in the literature.

the system as static networks' characteristics might adversely impact the derived insight [11], [12], [13], [14], [15].

A. Background

Frequent subgraph mining problem has attracted substantial attention in domains where the data can be represented as networks, such as in chemo-informatics [16], [17], [18], health informatics [19], [20], [21], [22], [23], public health [24], [25], [26], bioinformatics [27], [28], [29], social network analysis [30], [31], [32], computer vision [33], [34], [35], [36], [37], [38], and security [39], [40], [41], [42], [43]. The frequent subgraph mining in these discplines are either applied to a data set of small networks [44] or a data set of one large network [45]. These tasks are traditionally called *network-transaction setting* and *motif discovery*, respectively. Also, the output of the mining process is called *frequent subgraphs* (similar to *frequent itemsets* in the *frequent itemset mining* literature) for the former setting and *motifs* after a study by Milo et al. [46] for the latter setting.

Fig. 1 provides a taxonomy of the algorithms in the frequent subgraph mining literature. These algorithms can be categorized based on the network data available, either a single network or a set or sequence of networks. These algorithms are then can be categorized based on the temporality of the data. In cases where the data set is composed of a set of static networks, the algorithms can be classified based on the adopted approach for graph traversal and pattern search strategy [17], [18], [47], [48]. In the temporal network case, the algorithms can be classified based on the patterns being mined; either each network in the sequence is mined (inter-network subgraph mining) [49], [50], [51] or the changes occurring between each pair of consecutive networks in the sequence (intra-network subgraph mining) [52], [53]. In some algorithms, the inter-network subgraph mining approach is generalized to multiple sequences (extended inter-network subgraph mining) [54], [55]. Besides, in some applications, the networks are added to the sequence in real-time, which creates a separate category of algorithms [56], [57]. For further details and a discussion of algorithms in each subcategory, refer to [44]. The algorithms can be classified based on the adopted approach for frequency computations in single static networks in the motif discovery problem [58], [59]. For motif discovery in a large temporal network, the algorithms can be classified based on the temporal changes occurring in the network data, such as in the network's attributes, network topology, or when the network data is provided in real-time [60], [61], [62].

B. Our Contribution

One common approach for mining frequent subgraphs in temporal networks is representing the temporal network as a sequence of static networks. This type of representation of temporal networks has attracted some popularity as it can capture the system's temporal aspects to some extent. However, as will be discussed in the following section, adopting this modeling approach creates a computation-expressiveness trade-off. In other words, increasing the expressiveness of the network representation increases computational costs. For reducing the computational cost, we need to sacrifice some of the system's temporal aspects. Due to this fact, when the duration of interactions between system's entities is not identical for all the interactions, the equal-width temporal aggregation approach might over-represent some of the interactions.

C. Problem Formulation and Paper Organization

Unlike other approaches proposed in the literature, our objective is to mine all frequent patterns in a data set of continuous-time temporal networks while retaining their actual temporal information. Therefore, this paper is dedicated to addressing the following problem.

Problem definition: Given a data set $DS = \{N_1, N_2, \ldots N_n\}$ of n continuous-time temporal networks and a user-defined support threshold $\epsilon \in [0, 1]$, the problem of frequent pattern mining in continuous-time temporal networks aims to identify

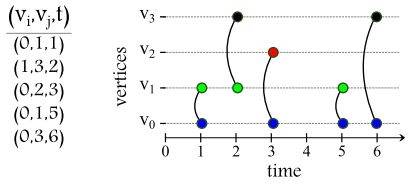

Fig. 2. Example of a contact sequence. The list of edges on the left is visualized on the right with an explicit temporal dimension.

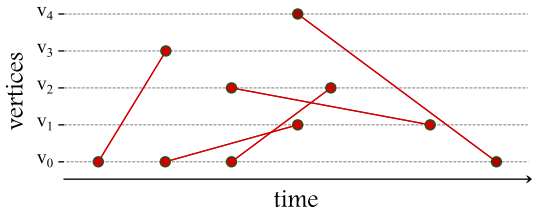

Fig. 3. Continuous-time temporal network with vertices and edges active over some period of time.

all patterns that appear in at least $\epsilon \times n$ of the networks in DS while preserving their actual temporal information.

To solve this problem, we first introduce a novel concept to represent temporal networks in DS. In some cases, we might be interested in mining structurally identical structures while accepting minor variations in temporal information. To make this possible, we propose a series of isomorphism definitions in the context of temporal networks, adding some levels of tolerance in temporal variations to find more generalizable patterns. Next, we explain the algorithm developed for mining frequent patterns in continuous-time temporal networks, called *tempowork*. We discuss the performance of the proposed algorithms analytically and evaluate them experimentally using three real-world data sets from different disciplines. To accomplish all these tasks, we need to introduce several novel concepts and heuristics and provide definitions for some other available concepts.

II. Temporal Network Representation

Networks are considered temporal if their components, vertices and edges, or their associated attributes, change over time. We define temporal networks as follows:

Temporal Network: A temporal network N is defined over a range of $\mathbb{W}$ as an ordered pair, $N = (V, E)$, of two sets, $V = \{v_1, v_2, \ldots, v_n\}$ which is the set of vertices of the network and referred to as V_N, and $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$, which is the set of temporal edges of the network and referred to as E_N. An edge e_k is represented as $e_k = \{v_i, v_j, a_i, l_k, a_j, s_k, \delta_k\}$ where:

- v_i and v_j: identifiers of the edge's two end-points.
- a_i and a_j: attributes of v_i and v_j, respectively. These attributes might be different between the same pair of vertices in various interactions. Also, the same vertex might take different attributes in its interaction with other vertices in overlapping intervals.
- l_k: edge attribute, which might differ between the same or different pairs of vertices in various interactions.
- s_k: the starting point of the interaction window in which e_k is active.
- δ_k: length of the interaction window in which e_k is active.

In some literature, the temporal networks are represented as either a contact sequence or an interval network [13], [14], [15]. The contact sequence representation is composed of edges in the form of $\{v_i, v_j, t\}$, where v_i and v_j are identifiers of the edge's two end-points and t is the point in time that these vertex pairs are connected. Fig. 2 shows an example of a list of edges in a contact sequence and the corresponding visualization.

In applications where the interactions are instantaneous, adopting a contact sequence representation is preferable (for example, in email correspondence where send and receive events happen in a fraction of seconds). However, in other applications, the duration of interactions is not negligible, for example, in face-to-face interactions, transportation networks, or some of the applications of proximity networks. Therefore, edges are shown as $\{v_i, v_j, s, \delta\}$ (or $\{v_i, v_j, s, f\}$), where s is the start time of the interaction and δ (f) is the duration (finish time) of the interaction. In this case, the contact sequences are a special case in which $\delta = 0$ ($s = f$). We adopt this latter representation in this paper and generalize that to both attributed and unattributed networks. Fig. 3 visualizes an unattributed temporal network of this type. This network is composed of five pairs of vertices' interactions over some period of time. Here, for example, we have v_0 at the starting time of interaction connected to v_3 at the ending time of interaction, meaning that v_0 and v_3 interact for the entire duration where they are connected. In Fig. 3, an undirected network is shown. Therefore, we can change the starting point andending point to be v_3 and v_0, respectively. To establish a standard visualization approach for directed networks, we can opt to draw the edges from the tail vertices to the head vertices.

In the literature of frequent subgraph mining, on the other hand, the common approach toward temporal network representation and analysis is converting the temporal dimension to a sequence of intervals and representing the continuous network as a sequence of aggregated static networks [44]. In this representation, for the range of temporal network, $\mathbb{W}$, and aggregation window, agg_w, the number of aggregation windows or static networks would be $|seq| = \lceil \mathbb{W}/agg_w \rceil$. For each aggregation window, the relationship between each pair of vertices is aggregated. In other words, for each pair of vertices, a connecting edge is assumed if at least there is one connection between the pair of vertices in that aggregation window, independently from the duration of the connection. It implies that by increasing the agg_w, the probability of having a connection between every two vertices in each aggregation window increases. On the other hand, the number of static networks, $|seq|$, decreases. This representation of temporal network is shown in Fig. 4(a) for multiple agg_w as a sequence of static networks related to the temporal network in Fig. 4(b).

When the $agg_w = \infty$, we consider an edge between each pair of vertices if there is at least one connection between these two vertices at some point in $\mathbb{W}$. When the $agg_w = \infty$, all the network's dynamic aspect is overlooked, and the continuous network is represented as a single static network. By decreasing

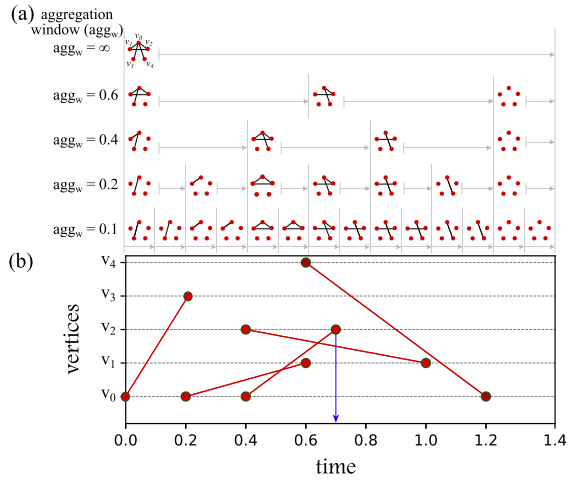

Fig. 4. Temporal network representation in sequential and continuous-time forms. 4a: Sequences of static networks at different aggregation times representing a temporal network. 4b: Continuous-time representation of the same temporal network.

the agg_w, more continuity characteristics of the network are captured. This is also shown in Fig. 4(a) as we move from $agg_t = \infty$ to $agg_w = 0.1$. However, there are some downsides to this type of representation. For example, in Fig. 4(a), considering $agg_w = 0.2$, the representation can capture most of the dynamics associated with the temporal network except for $t \in [0.6, 0.8]$. In this period, the corresponding static network shows a connection between v_0 and v_2 in [0.6,0.8], which is not correct, as this interaction ends at $t = 0.7$. Therefore, the $agg_w = 0.2$ over-represents the edge between these two vertices from [0.6,0.7] to [0.6,0.8]. This over-representation can be modified by reducing the agg_w to 0.1. In this case, all the temporal network's dynamic characteristics are correctly captured by static networks at regular time-stamps. However, many duplicate static networks are generated at consecutive time-stamps. These duplications negatively impact both the memory requirements and the processing resources needed for frequent pattern mining. Furthermore, considering that the actual duration of some of the interactions might be more than the agg_w, some post-processing might be necessary to evaluate the relationships between edges mined in the sequence of static networks and their corresponding interactions in the original network. To overcome these challenges, we adopt an interval network representation (examples are shown in Figs. 3 and 4(b)). Besides, because there might be multiple edges between each pair of vertices with different attributes, a starting-point sorted edge-based representation is utilized for each pair of vertices in the form of:

$$v_i = \{v_j : [(a_i^1, e_l^1, a_j^1, s_i^1, \delta_i^1), (a_i^2, e_l^2, a_j^2, s_i^2, \delta_i^2), \ldots, (a_i^k, e_l^k, a_j^k, s_i^k, \delta_i^k)]\} \quad (1)$$

Then, the network can be written as follows (and potentially as adjacency lists with extra dimensions for labels and interaction windows):

$$N = \{v_i = \{v_j : edge\_list_j\}\} \quad (2)$$

where $edge\_list_j$ is the list of edges between vertices v_i and v_j in the form of (1).

Note that there might be multiple edges between each pair of vertices appearing at different intervals in $\mathbb{W}$. The vertices might appear or disappear over the continuous dimension. Besides, the attribute of vertices, a_i and a_j, and edges, l_k, might change in different interactions, even between the same pair of vertices interacting at different intervals in $\mathbb{W}$.

Some other network representations would be special cases of the proposed representation. For example, one can use identical attributes for the network components; then, it is considered an unlabeled or unattributed temporal network. Or, by considering starting point for e_i as constant and $\delta_i = \infty$ for $i \in \{1, \ldots, m\}$, the network would represent a static network.

III. Preliminary Concepts and Algorithms

Based on the proposed representation, the next step is defining and introducing the basic concepts of frequent subgraph mining and concepts needed to adopt or develop for mining frequent patterns in temporal networks. The typical approach adopted by different algorithms for mining frequent subgraphs is composed of multiple steps recursively repeated. First, the frequent single vertices or edges in the network database are identified by comparing these simple subgraphs' frequencies with a user-defined threshold. In the subsequent step, some algorithms in the existing literature employ an Apriori-like approach to generate a set of candidates from already identified frequent subgraphs to detect larger frequent subgraphs [17], [18]. Conversely, others aim to eliminate candidate generation and false positive pruning adopting an optimized pattern growth strategy [47], [48]. We have chosen the latter approach; however, incorporating temporality into the data set of temporal networks introduces an additional layer of complexity into the problem of frequent pattern mining. These complexities arise due to the inclusion of interaction duration and the delay between the initiation of interactions. To effectively utilize the established practices from the frequent pattern mining literature, such as efficient pruning strategies and frequency computation, it is necessary to transform temporal networks into well-established concepts within the domain of graph theory. These concepts must be customized in a manner that not only preserves the structural data of the networks but also enables the lossless retrieval of associated temporal information when required. Furthermore, given the computational complexities inherent in the frequent pattern mining problem, it is crucial to minimize other computational costs by adopting efficient data structures for storing and retrieving network data. Therefore, in addition to the fundamental concepts of frequent pattern mining in network data sets, these considerations have guided us to adopt, customize, and design a series of concepts as outlined in the subsequent section. To understand both the process of frequent pattern mining and complexities associated with this problem, one must familiarize themselves with various essential concepts. These include the graph and subgraph isomorphism problems, common pruning strategies, and canonical labeling. Detailed explanations of these concepts can be found in Appendix A, available online.

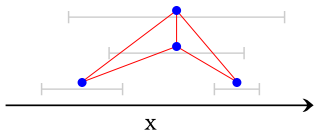

Fig. 5. Interval graph created from four intervals.

A. Constrained Interval Graph

Compared to the previous approaches proposed in the literature in which the temporal dimension of networks are transformed into equal-width intervals, we are interested in preserving the actual duration of and delays between interactions. Considering the uniqueness of this approach, we needed to design a novel concept, *Constrained Interval Graph* (CIG), to preserve both structural and temporal information of networks. It also makes it possible to traverse the temporal network as a crucial step in frequent pattern mining. The constrained interval graph concept is developed on top of the definition of interval graphs. Interval graphs are a special class of intersection graphs with a rich history of analysis [63], [64]. Given a collection of nonempty sets, an intersection graph is composed of vertices representing sets in the collection and edges as connections between vertices if and only if their corresponding sets' intersection is nonempty. In cases where the objects of the collection are intervals, and edges represent interval intersections, the resulting graph is called interval graph. In the following, after providing the formal definition of intervals and interval graphs, the constrained interval graphs are explained in detail.

An interval I denoted by $[\underline{x}, \overline{x}]$ as a subset of the real line is defined as follows:

$$I = [\underline{x}, \overline{x}] = \{z \in \mathbb{R} | \underline{x} \leq z \leq \overline{x}\} \tag{3}$$

We consider an interval closed if it includes both end-points of the interval [65]. Then, given two intervals, $I = [\underline{x}, \overline{x}]$ and $I' = [\underline{y}, \overline{y}]$, the intersection of two intervals would be considered empty if $\overline{y} < \underline{x}$ or $\overline{x} < \underline{y}$, otherwise it is defined as follows [66]:

$$\begin{aligned} I \cap I' &= \{z | z \in I \wedge z \in I'\} \\ &= [max\{\underline{x}, \underline{y}\}, min\{\overline{x}, \overline{y}\}] \end{aligned}$$

Interval Graph: Given a set of intervals, $S(I)$, an interval graph IG is defined as a network composed of:

- V_{IG}, in which each vertex of IG is associated with an interval in $S(I)$, and
- E_{IG}, in which an edge represents two connected vertices if and only if the intersection of their corresponding intervals in $S(I)$ is non-empty [67].

Fig. 5 shows an interval graph constructed from four intervals. The vertices of intersecting intervals are connected with an edge.

Constrained Interval Graph: For any given edge e_i in a temporal network N, the edge and its corresponding end-points are associated with an interval, $\mathscr{I}_i = [s_i, s_i + \delta_i]$. We can transform temporal networks into a particular type of interval graph by using the intervals associated with edges and their corresponding end-points. For each edge in the temporal network N, we add one node to the interval graph. Then, for each pair of overlapping edges e_i and e_j in N, we connect their corresponding nodes in the interval graph if they have end-points representing the same vertex. In other words, using the definition of interval graphs, the constrained interval graph CIG is defined as a network composed of a set of vertices, V_{CIG}, which are the edges in the temporal network N and an edge set, E_{CIG}, composed of edges connecting pairs of overlapping edges' in the temporal network if they have a vertex in common. Fig. 6 shows how a temporal network composed of two edges is transformed to a CIG, as the two edges e_i and e_j overlap, and they have one vertex in common.

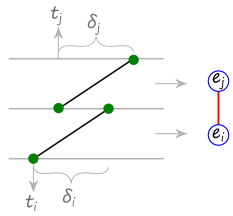

Fig. 6. Constrained interval graph created for a temporal network composed of two overlapping edges, e_i and e_j with one vertex in common shown with two blue vertices and a red edge.

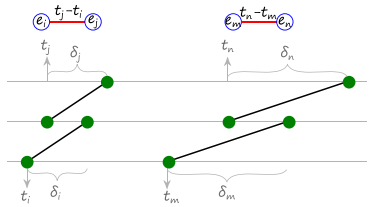

Fig. 7. Constrained interval graph created from two temporal networks, each is composed of two overlapping edges with one vertex in common.

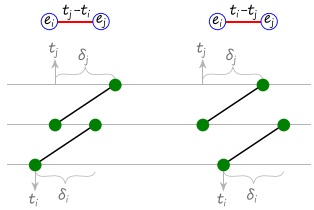

Fig. 8. Associated constrained interval graphs are identical for two different temporal networks.

Furthermore, we utilize edges in E_{CIG} to capture the differences between starting points of edges in N. For this purpose, an attribute is added to each edge of E_{CIG} computed as the difference between the starting points of the corresponding temporal edges in N connected by an edge in E_{CIG}, i.e., $max(s_i, s_j) - min(s_i, s_j)$. Fig. 7 shows examples of two temporal networks and their associated CIGs composed of one edge with an attribute representing the differences between starting points of the two temporal edges.

Although this definition of constrained interval graph creates a deterministic representation of the corresponding temporal network, it suffers from the edges' delay's symmetric nature. Fig. 8 visualizes this problem. In this figure, two edges e_i and e_j are shown in two configurations wherein the left configuration,

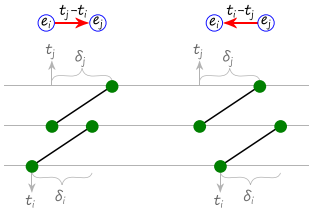

Fig. 9. Constrained interval graphs are created with directed edges to differentiate two temporal networks.

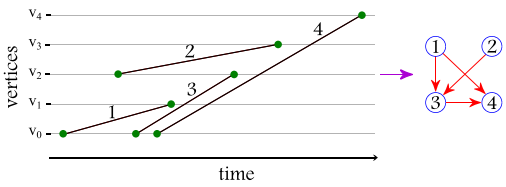

Fig. 10. Right network shows the CIG created from the continuous-time temporal network shown on the left.

e_i appears before e_j, and in the right configuration e_j appears first. In both cases, the edge attribute would be identical, as the difference between starting points of the two edges is the same. In other words, given a CIG without the information related to the starting point of edges and vertices, it is impossible to reconstruct the original temporal network from the constrained interval graph because none of the current labels and attributes of CIG expresses which edge appears first.

To solve this problem, we make the edges in CIG directed, e.g., connecting the vertex related to the edge with the smaller starting point to the vertex pertaining to the edge appearing later in the temporal network by a directed edge. As shown in Fig. 9, now the two different temporal networks are uniquely represented by two constrained interval graphs.

With the above solution, we can now preserve and retrieve the precise temporal information of networks. However, the proposed design involves summarizing the information of each edge in the temporal network, including the two end points, into one vertex in the corresponding CIG. This aggregation makes it challenging to extract the structural information of the temporal network directly without prior knowledge. To address this issue, edges in CIG will be assigned a second label (in addition to the delay label), which takes the form of "xy" & $x, y \in \{1, 2\}$. As the edges in CIG are directed, the label xy indicates that the end point located in position x of the tail vertex is connected to the end point located in position y of the head vertex. To maintain the order of end points associated with each vertex in CIG when referred to with xy labels, we will use an auxiliary map $M = \{\mathbb{v} : (v_i, v_j)\}$, where each vertex $\mathbb{v}$ in CIG is mapped to the corresponding temporal edge's end points.

Fig. 10 shows the CIG associated with a continuous-time temporal network. Each vertex of the CIG represents one edge of the temporal network. The direction of edges in CIG shows the relative delay between starting points of each pair of edges in the temporal network. The edges are attributed with these delays and the labels associated with the structural information of the corresponding temporal edges. However, for the sake of clarity, these attributes are not shown in this figure.

We assume that the temporal network data available as a list of temporal edges, $\mathscr{L}$, in which the edges are sorted based on their starting time. Therefore, to read the data and construct the corresponding CIG, we need to iterate over $\mathscr{L}$ edge by edge. For each temporal edge e_i in $\mathscr{L}$, we need to perform two checks:

- whether any of the edges appeared earlier in $\mathscr{L}$ has a vertex in common with one of the two end-points of e_i, and if yes,
- whether any of those edges appeared earlier overlap with the interval associated with e_i.

Considering that there might be multiple other edges passing these checks for each edge, we need to create an efficient data structure to keep track of each vertex and its associated intervals. Once we read a new edge, the data structure should be updated with the new edge information. The data structure adopted for this purpose is the interval tree. Therefore, to efficiently accomplish reading and updating the interval trees, we use a map of vertex *id*s to interval trees, $\mathscr{M} = \{id : IT_{id}\}$, where the keys of $\mathscr{M}$ represents the identifier of vertices in N, and the values of $\mathscr{M}$ are interval trees keeping track of intervals associated with edges having vertex id as one of their end-points. When we read a new edge e_i with two vertices v_m and v_n, we update the interval trees mapped to v_m and v_n with the interval of e_i, $[s_i, s_i + \delta_i]$. In the following, the characteristics of interval trees are described. Then, we explain the CIG construction step in detail.

B. Interval Tree

There are multiple data structures developed for operations associated with intervals and ranges, such as interval trees, range trees, and segment trees [68], [69]. Among them, the interval tree is typically known as the most efficient data structure for storing and querying continuous intervals. For the construction of interval trees for the temporal network's vertices, we follow the approach proposed in [70]. The interval tree data structure can be considered an augmented data structure constructed on top of red-black trees. The red-black trees are (approximately) balanced binary search trees with specific properties. The properties should hold for interval trees to make them efficient data structure for implementing different types of operations, including insertion, deletion, and interval search, on sets of intervals.

In our application, we focus on interval insertion and search operations. The interval insertion is used when we read a new edge from $\mathscr{L}$, and we want to update the interval trees associated with the two end-points of the edge in $\mathscr{M}$. On the other hand, the interval search is used to connect the end-points of the newly added edge to the end-points of edges already inserted. Therefore, we need to know in which of the earlier appearances of the desired vertices, their corresponding intervals overlap with the new edge's interval. The interval search and insertion operations, based on their applications in this paper, are described in more detail in Appendix B, available online.

Algorithm 1: CIG Construction Algorithm.

procedure CONSTRUCT_CIG($\mathscr{L}$)
  Initialize $CIG = (V, E)$, $V_{CIG} = \emptyset$, $E_{CIG} = \emptyset$
  Initialize $\mathscr{M} = \{id : IT\}$
  while !End of $\mathscr{L}$ **do**
    Read edge e_i from $\mathscr{L}$ ▷ $e_i = \{v_m, v_n, a_m, l_i, a_n, t_i, \delta_i\}$
    Define $\mathscr{I} = [t_i, t_i + \delta_i]$
    Add one vertex to CIG representing e_i with an associated attribute, "a_i-e_l-a_j-δ_{ij}"
    $v_m\_neighbors \leftarrow IntervalTree\_Search(IT_m, \mathscr{I})$ ▷ $IT_m = \mathscr{M}[v_m]$
    for $v \in v_m\_neighbors$ **do**
      Retrieve interval $\mathscr{I}_v = [t_v, t_v + \delta_v]$ associated with v
      Compute edges' delay $d = t_i - t_v$
      Connect vertex including v in CIG to the vertex in CIG
        representing e_i with a directed edge e
      Label e with d
    $v_n\_neighbors \leftarrow IntervalTree\_Search(IT_n, \mathscr{I})$ ▷ $IT_n = \mathscr{M}[v_n]$
    for $v \in v_n\_neighbors$ **do**
      Retrieve interval $\mathscr{I}_v = [t_v, t_v + \delta_v]$ associated with v
      Compute edges' delay $d = t_i - t_v$
      Connect vertex including v in CIG to the vertex in CIG
        representing e_i with a directed edge e
      Label e with d
    Call $IntervalTree\_Insert(IT_m, \mathscr{I})$
    Call $IntervalTree\_Insert(IT_n, \mathscr{I})$
  return CIG & $\mathscr{M}$

This loop is similar to the previous For Loop, with $m \rightarrow n$

C. Constrained Interval Graph Construction

Once we have operational functions for insertion and search of intervals in interval trees, we are ready to construct the constrained interval graph, CIG, representing the temporal network N. For doing that, first, we create an empty map associating vertex identifiers to the interval trees, $\mathscr{M} = \{id : IT_{id}\}$ and initialize CIG with empty sets for vertices and edges. Besides, we assume that the temporal network data is provided as an edge list $\mathscr{L}$ sorted based on the starting points of edges. Then, we iterate over $\mathscr{L}$ edge by edge. For each edge e_i, the corresponding interval $\mathscr{I} = [s_i, s_i + \delta_i]$ is created. The edge e_i has two end points, v_m and v_n. Corresponding to the edge e_i, a new vertex, $\mathbb{v}_i$, is added to the vertex set of CIG. In the next step, we search the $\mathscr{M}$ to see if the interval trees associated with v_m and v_n, namely IT_m and IT_n, have any intervals overlapping with $\mathscr{I}$. Each of IT_m and IT_n might have none, one, or more than one intervals overlapping with $\mathscr{I}$ representing different edges in N. Then, we connect vertices representing these edges of N in CIG to $\mathbb{v}_i$ with a directed edge. Each vertex in CIG is labeled with the data of the temporal edge it represents. The labels of vertices in CIG are composed of the labels of the two end-points, a_i and a_j, the edge's label, l_k, and the length of the interval, δ_k of the corresponding edge in the temporal network. Each edge in CIG is attributed with the difference in starting points of the associated edges with the vertices (as shown in Fig. 9). Next, we update IT_m and IT_n interval trees with the new interval $\mathscr{I}$. After reading all the edges in the edge list, we have the CIG representing the temporal network ready for downstream analysis. Algorithm 1 provides the pseudo-code for the construction of CIG.

D. Temporal Network Reconstruction

In some of the applications, we need to convert subgraphs of a CIG to the corresponding temporal networks that they represent. It can be easily shown that the relationships between CIGs and temporal networks are not one-to-one. In other words, although Algorithm 1 always constructs a unique CIG for any temporal network given, there might be multiple subgraphs of CIG representing the same temporal network. The mapping of CIG to the corresponding temporal network is accomplished using the attributes of vertices and edges and edge directions. The vertices' attributes in CIG provide the attributes of edges in the associated temporal network. The edges' attributes and directions in CIG are to infer the magnitudes of overlaps and delays between pairs of edges in the temporal network.

Taking a more in-depth look into the constrained interval graph and the temporal relationships of edges represented by directed edges in this graph, it can be deduced that the constrained interval graph is a directed acyclic graph (as we assume that time has a given direction).

For reconstructing a temporal network from a CIG, we always start with the vertices having the smallest identifiers and proceed in the vertex set. It is because the edge list $\mathscr{L}$ that the CIG is created from in the first place has been sorted based on the starting points of the edges. Therefore the vertices with the smallest identifiers represent the edges that appear earlier in $\mathscr{L}$ and have a smaller starting time. Besides, we assume that the first vertex in CIG has a starting point of zero (or any other arbitrary value). The starting points of other edges are derived from relative delays to the other edges in CIG.

Algorithm 2: Temporal Network Reconstruction Algorithm.

procedure RECONSTRUCT_TEMPORALNETWORK(CIG, *support*) ▷ *support* is one of DS networks supporting CIG
  Initialize $\mathscr{L}$ as an empty list ▷ This list is used for recording edges of temporal network
  for all v in V_{CIG} **do**
    $v.visited = False$
  Using the first node read from CIG, extract $\{a_1, e_l, a_2, \delta_{12}\}$
  Using the *support*, identify the two nodes representing v_1 and v_2 ▷ Note: each $v \in CIG$ represent two vertices v_i and v_j of the temporal network, and their connecting edge
  Define $temp\_edge = \{v_1, v_2, a_1, e_l, a_2, 0, \delta_{12}\}$
  $\mathscr{L}.push\_back(temp\_edge)$
  Create an empty stack, S
  $S.push(v_1)$
  Create an empty map for recording start times of edges, $STs = \{\}$
  $STs = \{v_1 : 0\}$ ▷ We consider the starting time of v_1 as zero
  while $!S.empty()$ **do**
    $v = S.pop()$
    if $!v.visited$ **then**
      $v.visited = True$
      for all u in $v.neighbors()$ **do**
        if $!u.visited$ **then** ▷ ⟶ Extra code relative to DFS starts here!
          Extract temporal edge attributes from u, $\{a_i, e_l, a_j, \delta_{ij}\}$
          Using the *support*, identify the two nodes representing v_i and v_j
          $STs[u] = STs[v] + e^l_{uv}$ ▷ e^l_{uv} is the label of the edge connecting vertices u and v in CIG
          Define $temp\_edge = \{v_i, v_j, a_i, e_l, a_j, STs[u], \delta_{ij}\}$
          $\mathscr{L}.push\_back(temp\_edge)$ ▷ ⟶ Extra code relative to DFS ends here!
          $S.push(u)$
  return $\mathscr{L}$

The reconstruction of a temporal network from a CIG or a subgraph of CIG is performed as follows. We start with the vertex with the smallest identifier in the (subgraph of) CIG, $\mathbb{v}_1$. Then we traverse the CIG with one of the directed acyclic graph traversal strategies (such as breadth-first search or depth-first search). We consider the starting point of the edge represented by $\mathbb{v}_1$ as zero. Therefore, we create the first edge of the temporal network, $e_1 = \{v_1, v_2, a_1, l_1, a_2, 0, \delta_1\}$. Using the attributes of the edges originating from $\mathbb{v}_1$, we can find the starting time of the neighbor vertices (representing edges in the temporal network). We traverse the CIG vertex by vertex to generate the temporal network's edges using this strategy. Algorithm 2 provides the pseudo-code for reconstructing temporal networks from their associated CIGs.

IV. THE *TEMPOWORK* ALGORITHM

After being able to construct constrained interval graphs from temporal networks and reconstruct temporal networks from (any subgraphs of) constrained interval graphs, we can discuss the *tempowork* algorithm for the identification of frequent patterns in temporal networks. Specifically, given a data set of n temporal networks, $DS = \{N_1, N_2, \ldots, N_n\}$ and support threshold $\epsilon \in [0, 1]$, we would like to identify all the patterns appearing in at least $\epsilon * n$ networks of the DS. The format of the input data would be a list composed of network *id*s and edges for each network, where the edges are sorted based on their starting times.

$$
\begin{aligned}
&N\,\#\,t-1\\
&\vdots\\
&N\,\#\,t\\
&e\;v_i^1\;v_j^1\;a_i^1\;e_l^1\;a_j^1\;s^1\;\delta^1\\
&\vdots\\
&e\;v_i^m\;v_j^m\;a_i^m\;e_l^m\;a_j^m\;s^m\;\delta^m\\
&N\,\#\,t+1\\
&\vdots
\end{aligned}
$$

In this format, N and e at the begining of each line inform the algorithm whether the line is associated with a network or an edge. The line $N\,\#\,t$ specifies the *id*=t of the network. The lines starting with an e represent the list of edges for each network. In each edge line, the vertices $(v_i, v_j) \in V$ are the two end points of each edge, (a_i, a_j) are the labels associated with the vertices, e_l is the label of the edge, and s and δ are the starting point and duration of the interaction between the two vertices, all space-separated. The algorithm should produce an output with the exactly similar format. However, the list of edges provided represent a frequent pattern appearing at least in $\epsilon * n$ networks of DS. Also, the lines starting with an N

denote the occurrence list of patterns, i.e., the space-seperated *id*s of the temporal networks in DS including the pattern. Note that the sequence of edges considered as frequent patterns might be separated with infrequent edges in different networks of DS.

To accomplish these goals, first, we convert the $N_i \in DS$ for $i \in \{1, n\}$ to their associated CIG_i and create a secondary data set of constrained interval graphs, DS'. The DS' can be considered as a data set of static networks. However, CIGs has multiple characteristics which should be noted in the mining process. Furthermore, we need to adopt a series of strategies for pattern growth, subgraph enumeration, graph isomorphism, and subgraph isomorphism checks and verification. It would be beneficial to adopt strategies such that no duplicates subgraphs are generated, and any brute-force implementation of graph and subgraph isomorphism problems are avoided.

During frequent pattern mining, the verification of graph isomorphism is carried out to confirm that the current candidate has not been examined earlier in the process and prevent any redundant exploration of patterns. The algorithm should either avoid generating duplicate candidates or identify identical candidates as soon as possible in the mining process. On the other hand, the subgraph isomorphism is used when we want to verify if any of the networks in the data set contain the candidates being enumerated. In the worst-case scenario, we need to iterate over the networks in the data set one by one and verify if any of them contain the candidate of interest. Both of these tasks are computationally very expensive. We adopt a set of heuristics to avoid both of them as much as we can.

We start the mining process with the identification of frequent vertices in DS'. These vertices represent frequent temporal edges in the DS. Besides, we identify frequent edges in DS'. Each frequent edge in DS' represents a two-edge pattern in DS. The frequent edges in DS' can be reordered based on their frequencies. We use these frequent edges (as seeds and in combination with other frequent patterns already detected) to generate new candidates in each iteration of the mining process.

As we discussed, the CIGs of temporal networks are directed acyclic graphs, DAGs. However, in addition to the characteristics that CIGs inherit from DAGs, CIGs have an essential feature; for mining the CIGs, we do not need to mine all the edges. All the frequent subgraphs of temporal networks can be represented with subtrees of the corresponding CIGs. This characteristic results from the temporal relationships between vertices in CIG (representing temporal edges in temporal networks). In other words, if some of the temporal relationships between vertices of CIG are known, we might be able to deduce the temporal relationships between other vertices of CIG.

Therefore, we start mining by adding one edge (with only their delay attributes) at a time to known frequent subgraphs. The first set of frequent subgraphs are frequent edges. We add an edge to a frequent subgraph to create a new candidate; we make sure that this new edge does not create any cycle in the candidate's undirected version. Also, to minimize the number of duplicates generated, we will be using the lexicographic ordering introduced in gSpan [47], [48]. For the sake of space, we do not repeat the principles of and pattern growth strategies based on the lexicographic ordering, as they are perfectly discussed in gSpan papers. However, there are some changes we need to apply to these principles based on the characteristics of DAGs and CIGs that will be addressed in the following.

The CIGs are directed networks. Therefore, we need to consider the direction of edges when we are generating new candidates. We only create new candidates by adding forward-edges, as we are interested in mining non-cyclic patterns. The edges that their addition to the candidates create cyclic subgraphs in undirected versions of CIGs result in duplicate candidates. As discussed, each edge e of the CIGs carries the delay information between temporal edges that the two end-points of the e represent. If these two end-points are already connected through a different path, the delay that e represents can be deduced from the path that connects these two end-points. In other words, we do not need to create candidates using backward edges. This approach eliminates a large number of duplicate candidates. Although adopting the lexicographic ordering and forward growth of candidates prevents the generation of many duplicate candidates, the graph isomorphism problem can not be avoided entirely. In fact, it is even worse than the case where we mine a data set of static networks. It is because two completely acyclic candidates might still represent the same temporal network. Furthermore, the addition of a single edge can create different valid patterns based on the structural information connected to the edge's vertices. To ensure the correct identification of both duplicate candidates and patterns with distinct structures, we begin by converting candidates into their temporal networks (using Algorithm 2) and recreating their CIGs (using Algorithm 1). If candidates represent identical temporal networks, their corresponding CIGs should be isomorphic, considering the structural information of edges within CIGs. Furthermore, upon adding a new edge to a parent pattern, we examine various frequent canonical labelings that the parent pattern might represent. This validation step is vital for both pattern growth and frequency computation. Thus, by recording the canonical labeling of the CIGs, we ensure the prevention of generating duplicate candidates for pattern growth and identify all frequent and structurally distinct patterns. The canonical labeling can be created using the same lexicographic ordering principles or any other known to be efficient approaches proposed in the literature, such as nauty and Traces [71], or bliss [72]. The computational complexity of this process is discussed in Section V.

We create an embedding list for each frequent subgraph for frequency computation. When we add a forward edge to a frequent parent subgraph to create a new child candidate, we only check the embeddings that support the parent subgraph (instead of searching all the networks in the data set). Suppose the new edge added to a parent subgraph creates a valid child subgraph. In that case, we check whether the embeddings supporting the parent subgraphs can be extended in their associated networks with the new edge. If the list of networks that support the child subgraph meets the user-defined support threshold, we consider the child frequent, which can be used for candidate generation in the next iterations. Otherwise, neither the child nor the candidates created from this child (based on the downward-closure property) would be frequent and can be eliminated. We can avoid the subgraph isomorphism problem altogether by adopting

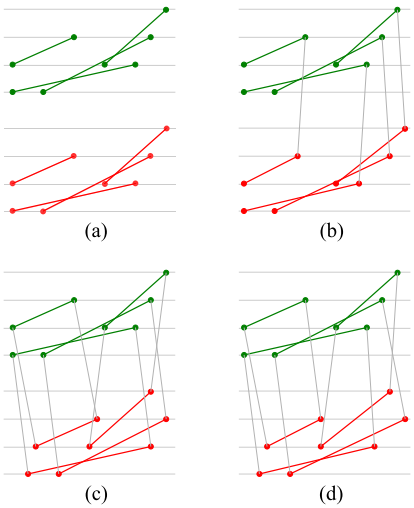

Fig. 11. Schematic representation of the four different isomorphism definitions considered for the temporal networks. In the exact formulation, (a) all the temporal edges' durations and the inter-edge delays are identical between the two networks. In the inexact-time version, (b) some user-defined variations in edge durations are permitted. However, the magnitudes of delays between edge pairs in the two isomorphic networks are exactly identical. In the exact-time sequence-preserved formulation, (c) the edges mapped to one another have exactly identical durations. However, two networks are considered isomorphic if the edges' appearances follow the same sequences in both networks. In the inexact-time sequence-preserved formulation, (d) the constraint of duration equality of corresponding edges is removed based on a user-defined variation tolerance mechanism.

this strategy for subgraph enumeration and frequency computation. Algorithm 3 provides the pseudo-code for the *tempowork* algorithm.

The temporal information of the temporal networks is recorded in the CIGs. The duration of the temporal edges (δs) are recorded in the vertices of CIGs, and the delays between temporal edges (Δs) are recorded in the edges of CIGs. In certain scenarios, we may regard two interactions or the delays between interactions as temporally identical if their variations fall within an acceptable tolerance range. Therefore, when we create the embedding lists, we can relax the temporal constraints of isomorphisms at different levels to accommodate minor variations in interactions' durations and initiation delays. In the following, we define four types of graph isomorphism based on the level of tolerance we consider two networks isomorphic. A visual representation of these definitions is provided in Fig. 11.

Exact-time graph isomorphism: Two networks, $N_1 = (V_1, E_1)$ and $N_2 = (V_2, E_2)$, are exact-time isomorphic ($N_1 \overset{e}{\simeq} N_2$) if there is a bijective function I^e such that in addition to the constraints defined for graph isomorphism, for any $\{v_i, v_j\} \in E_1$ mapped to $\{I^e(v_i), I^e(v_j)\} \in E_2$ and any pair of edges $e_i, e_j \in E_1$ mapped to $I^e(e_i), I^e(e_j) \in E_2$:

$$\begin{aligned} \delta\{v_i, v_j\} &= \delta\{I^e(v_i), I^e(v_j)\} \\ \Delta(e_i, e_j) &= \Delta(I^e(e_i), I^e(e_j)) \end{aligned} \tag{4}$$

Where $\delta\{v, v'\}$ represents the duration of the edge connecting v and v' and $\Delta(e, e')$ denotes the delay between starting points of any pair of edges e, e'.

Inexact-time graph isomorphism: Two networks, $N_1 = (V_1, E_1)$ and $N_2 = (V_2, E_2)$, are inexact-time isomorphic ($N_1 \overset{i}{\simeq} N_2$) if there is a bijective function I^i such that in addition to the constraints defined for graph isomorphism, for any $\{v_i, v_j\} \in E_1$ mapped to $\{I^i(v_i), I^i(v_j)\} \in E_2$ and any pair of edges $e_i, e_j \in E_1$ mapped to $I^i(e_i), I^i(e_j) \in E_2$:

$$\begin{aligned} \delta\{v_i, v_j\} &\simeq \delta\{I^i(v_i), I^i(v_j)\} \\ \Delta(e_i, e_j) &= \Delta(I^i(e_i), I^i(e_j)) \end{aligned} \tag{5}$$

Exact-time sequence-preserved graph isomorphism: Two networks, $N_1 = (V_1, E_1)$ and $N_2 = (V_2, E_2)$, are exact-time sequence-preserved isomorphic ($N_1 \overset{es}{\simeq} N_2$) if there is a bijective function I^{es} such that in addition to the constraints defined for graph isomorphism, for any $\{v_i, v_j\} \in E_1$ mapped to $\{I^{es}(v_i), I^{es}(v_j)\} \in E_2$ and any pair of edges $e_i, e_j, e_k \in E_1$ mapped to $I^{es}(e_i), I^{es}(e_j), I^{es}(e_k) \in E_2$:

$$\begin{aligned} &\delta\{v_i, v_j\} = \delta\{I^{es}(v_i), I^{es}(v_j)\} \\ &\Delta(e_i, e_j) < \Delta(e_j, e_k) \iff \\ &\Delta(I^{es}(v_i), I^{es}(v_j)) < \Delta(I^{es}(v_j), I^{es}(v_k)) \\ &\Diamond(\Delta(e_i, e_j) \neq \Delta(I^{es}(v_i), I^{es}(v_j))) \wedge \\ &\Diamond(\Delta(e_j, e_k) \neq \Delta(I^{es}(v_j), I^{es}(v_k)) \end{aligned} \tag{6}$$

Here, the expressions starting with $\Diamond$ means that the magnitude of delays can differ between corresponding isomorphisms. The sequence of appearance of edges is identical among the two networks.

Inexact-time sequence-preserved graph isomorphism: Two networks, $N_1 = (V_1, E_1)$ and $N_2 = (V_2, E_2)$, are inexact-time sequence-preserved isomorphic ($N_1 \overset{is}{\simeq} N_2$) if there is a bijective function I^{is} such that in addition to the constraints defined for graph isomorphism, for any $\{v_i, v_j\} \in E_1$ mapped to $\{I^{is}(v_i), I^{is}(v_j)\} \in E_2$ and any pair of edges $e_i, e_j, e_k \in E_1$ mapped to $I^{is}(e_i), I^{is}(e_j), I^{is}(e_k) \in E_2$:

$$\begin{aligned} &\delta\{v_i, v_j\} \simeq \delta\{I^{is}(v_i), I^{is}(v_j)\} \\ &\Delta(e_i, e_j) < \Delta(e_j, e_k) \iff \\ &\Delta(I^{is}(v_i), I^{is}(v_j)) < \Delta(I^{is}(v_j), I^{is}(v_k)) \\ &\Diamond(\Delta(e_i, e_j) \neq \Delta(I^{is}(v_i), I^{is}(v_j))) \wedge \\ &\Diamond(\Delta(e_j, e_k) \neq \Delta(I^{is}(v_j), I^{is}(v_k)) \end{aligned} \tag{7}$$

In these definitions, we need to provide user-defined thresholds to show how much temporal variations are tolerable for identifying inexact time equivalences. Also, we can discretize the duration of edges. The discretization problem has its own rich literature [73]. Different characteristics of the networks (such as attribute of vertices and edges and classes of networks) can be used to perform discretization in supervised or unsupervised modes. To preserve the sequences, we can assume that all the edges in the CIGs have the same attribute and let the differences between the duration of edges and their starting points create the sequences of edge appearances.

Algorithm 3: *Tempowork* Algorithm.

1: **procedure** PREPROCESSING (DS, min_supp, ISO_type)
2: Initialize DS' ▷ This is the CIG data set
3: Initialize 1_Node_map ▷ This is a dictionary: {node_labels: location of their embeddings in DS'}
4: Initialize 1_Edge_map ▷ This is a dictionary: {edge_labels: location of their embeddings in DS'}
5: **for all** N in DS **do**
6: $CIG_T = Construct\_CIG(T)$ ▷ If $ISO\_type \in \{i, es, is\}$, relabel δs and/or Δs accordingly
7: $DS'.push(CIG_T)$
8: Order node labels ("a_i-e_l-a_j-δ_{ij}"s) and edge labels (Δs) descendingly based on their frequencies in CIGs of DS'
9: Remove node and edge labels in CIGs with frequencies $< min\_supp$
10: Relabel lexicographically node and edge labels in CIGs with frequencies $\geq min\_supp$
11: Update 1_Node_map and 1_Edge_map dictionaries with the location of their embeddings in DS'
12: Return 1_Node_map, 1_Edge_map
13: **procedure** *tempowork*DS, min_supp, ISO_type
14: Initialize $frequent\_patterns$ ▷ This is a dictionary: {frequent_patterns: location of their embeddings in DS'}
15: Initialize cl_list ▷ It is a list for recording list of canonical labeling of CIG already mined
16: 1_Node_map, $1\_Edge\_map = Preprocessing(DS, min\_supp, ISO\_type)$
17: Update $frequent\_patterns$ with 1_Node_map and 1_Edge_map
18: $temp\_1\_Edge\_map = 1\_Edge\_map$ ▷ We create a copy of the 1_Edge_map
19: **procedure** CIG_MINING ($pattern$, $pattern\_support$)
20: **for all** $edge$ in $temp\_1\_Edge\_map$ **do**
21: $new\_pattern =$ add $edge$ to $pattern$ as forward edge
22: **if** $new\_pattern$ has the minimum lexicographical form **then**
23: $temp\_tw = Reconstruct\_TemporalNetwork(new\_pattern, support)$ ▷ $support$ is one of DS networks supporting $new\_pattern$ created from the function input argument $pattern\_support$
24: $temp\_cig = Construct\_CIG(temp\_tw)$
25: $cl = canonical\_labeling(temp\_cig)$ ▷ This line computes the canonical labeling of $temp\_cig$
26: **if** cl not already in cl_list **then** ▷ The cl is constructed for this child of all the
27: $cl\_list.push(cl)$ ▷ frequent but structurally different parents.
28: $frequent\_patterns[new\_pattern] = new\_pattern\_support$
29: Call CIG_Mining($new\_pattern$, $frequent\_patterns[new\_pattern]$)
30: **for all** $edge$, $frequent\_patterns[edge]$ in 1_Edge_map **do**
31: Call CIG_Mining($edge$, $frequent\_patterns[edge]$)
32: Remove $edge$ from $temp\_1\_Edge\_map$
33: Return $frequent\_patterns$

V. Computational Complexity Analysis

The frequent pattern mining process primarily drives the computational complexity of the proposed approach. However, for the sake of completeness, we provide the analysis of the computational complexity for all steps. Also, the worst-case computational complexity is discussed, including an explanation of how often it might occur. The algorithm processes a data set of n temporal networks, transforming each temporal network $N_i \in DS$ to a corresponding CIG_i to create a secondary data set DS'. For each network in DS, the algorithm reads each edge, searches the two interval trees associated with the edge's end points, and then updates these interval trees with the new edge using two insertion operations. Assuming the worst-case scenario, where all edges overlap with one another and share one common vertex, the interval search at iteration $i+1$ has a computational complexity of $\mathcal{O}(i)$. The insertion operation's computational complexity is $\mathcal{O}(\log i)$ [70], which is less expensive than the interval search operation in the corresponding iteration. As the algorithm iterates over each edge and performs the interval search for each edge, the total computational complexity for network N with the set of edges E_N amounts to $\mathcal{O}(|E_N|^2)$. This complexity, associated with Algorithm 1, should be performed once for all the networks in DS to generate DS' in lines 5–7 of Algorithm 3 and also repeated for all the candidate patterns reaching to line 24 of the same Algorithm.

The computational complexity of reconstructing the temporal network associated with a given (subgraph of) CIG is similar to that of Depth-First Search (DFS), as we traverse the CIG to reconstruct the temporal network. The complexity of DFS depends on the data structure used to represent the network. If an adjacency matrix representation is used, it is in the order of $\mathcal{O}(|V_{CIG}|^2)$. However, using an adjacency list representation can reduce the complexity to $\mathcal{O}(|V_{CIG}| + |E_{CIG}|)$ [70], where $|V_{CIG}|$ is the number of vertices and $|E_{CIG}|$ is the number of edges in the CIG. This complexity is associated with Algorithm 2 called in line 23 for all the candidate patterns reaching to this line in Algorithm 3.

Mining frequent patterns in DS' requires its own analysis. The proposed approach mitigates the issue of subgraph isomorphism by creating embedding lists for frequent patterns. However, for a subgraph s, we might still need to perform graph isomorphism verification after reconstructing the temporal network associated with s and constructing its CIG to ensure that it has not been already evaluated (lines 25 and 26 of Algorithm 3). A breakthrough study by Babai in [74], [75] shows that graph isomorphism can be solved in quasipolynomial time ($exp((\log n)^{\mathscr{O}(1)})$), where n is the number of vertices in s, which would be identical to the number of edges in the corresponding temporal network. He later proposed an approach for canonical labeling of graphs within the same time bound [76]. Therefore, in the worst-case scenario a canonical labeling should be constructed for each pattern being mined. In [47], it was demonstrated that the computational complexity of mining frequent patterns from a data set of static networks is $\mathscr{O}(kFn + rF)$, where k is the maximum number of subgraph isomorphism verifications needed between a subgraph and networks in DS', F is the number of frequent patterns, and r is the maximum number of duplicate labels generated (related to line 22 in Algorithm 3 and for support enumeration in line 28 of the same algorithm).

As noted, mining frequent patterns in networks is inherently computationally expensive due to the verifications required for graph and subgraph isomorphism. Adding the temporal layer exacerbates the problem's complexity due to the CIG construction and reconstruction. However, there are several considerations that can significantly improve performance. First, mining labeled network data sets, prevalent in many real-world applications, is less costly. Moreover, the variations in interaction durations and delays act as implicit labels, reducing the complexities associated with graph and subgraph isomorphism considerably. Additionally, adopting established pruning strategies, such as those discussed in [47], can significantly enhance the mining algorithms' performance. The degree of contribution of each of these factors is highly application-dependent, determined by the structural attributes of the networks being investigated, the number and distribution of labels attributed to vertices and edges, as well as the temporal properties of the networks. The following section provides an experimental analysis for real-world scenarios.

VI. Experiments

To the best of our knowledge, this is the first algorithm proposed for mining frequent patterns in continuous-time temporal networks. We evaluated the proposed algorithm's performance using three real-world data sets with different numbers of vertices, edges, and time windows:

- Hospital Ward Proximity Networks: A data set of proximity networks representing interactions between patients and healthcare providers in a hospital ward in Lyon, France [77], [78].
- High school Contact Networks: This data set includes seven days of temporal interactions among high school students of five classes [78], [79].

TABLE I
Characteristics of the Data Sets Used for the Experiments, Including the Number of Networks in Each Data Set and the Average Number of Vertices $\overline{|V|}$ and Edges $\overline{|E|}$ of Networks in Continuous-Time Representations of the Data Sets

data set	$\lvert N\rvert$	$\overline{\lvert V\rvert}$	$\overline{\lvert E\rvert}$
Hospital ward proximity network	5	48	2806
High school contact networks	7	151	2824
Sepsis EHR data set	13,229	5	24

- Sepsis EHR Data Set: This data set includes retrospectively collected EHR data related to cellular and physiological responses of sepsis patients in 13,229 visits [80].

A comprehensive depiction of these data sets is available in Appendix C.1, available online, along with a detailed explanation of the preprocessing conducted on these data sets for the proposed approach in Appendix C.2, available online. Table I summarizes these three data sets' characteristics.

The proposed algorithm was applied to the three network data sets described above using multiple frequency thresholds. Also, we implemented the algorithm for different definitions of isomorphism with different discretization parameters to evaluate the impact of temporal variations allowed on the number of frequent pattern detected by the algorithm. The results are shown in Figs. 12, 13, and 14. The numerical values of results are provided in Appendix D-Table 2, available online. The findings are discussed in the following subsection. The Python implementation of the algorithm used in this study is publicly available from the PyPI repository and can be installed under "*tempowork*". The experiments were conducted on a personal computer with an Intel Core i7-8700 3.20 GHz CPU processor with 16.0 GB installed RAM.

A. Discussion

The proposed algorithm was applied to the pre-processed data sets. We used the four different types of isomorphisms, different support values, and different discretization parameters. The results show that increasing the support thresholds decreases the number of frequent patterns in all three data sets consistently (Fig. 12).

The results of the implementation for the high school contact networks when the *id*s are used as labels showed that most of the frequent patterns are composed of either one temporal edge (the interactions between two specific students for a particular period) or two temporal edges (the interactions among three specific students in different arrangements) (Fig. 12-(b)). This can be attributed to the unique labels considered in this implementation for each student. Also, the algorithm did not find many frequent subgraphs for the sepsis EHR data set when it is implemented in the exact-time or exact-time sequence-preserved modes (Fig. 12-(d)). It can be attributed to the nature of this data set. The lab measurements are recorded at times with a precision of fractional portions of a second. At this precision, it is very rare to find patterns with identical durations among patients. However, when we implement the algorithm for the high school contact networks with classes as labels (Fig. 12-(c))

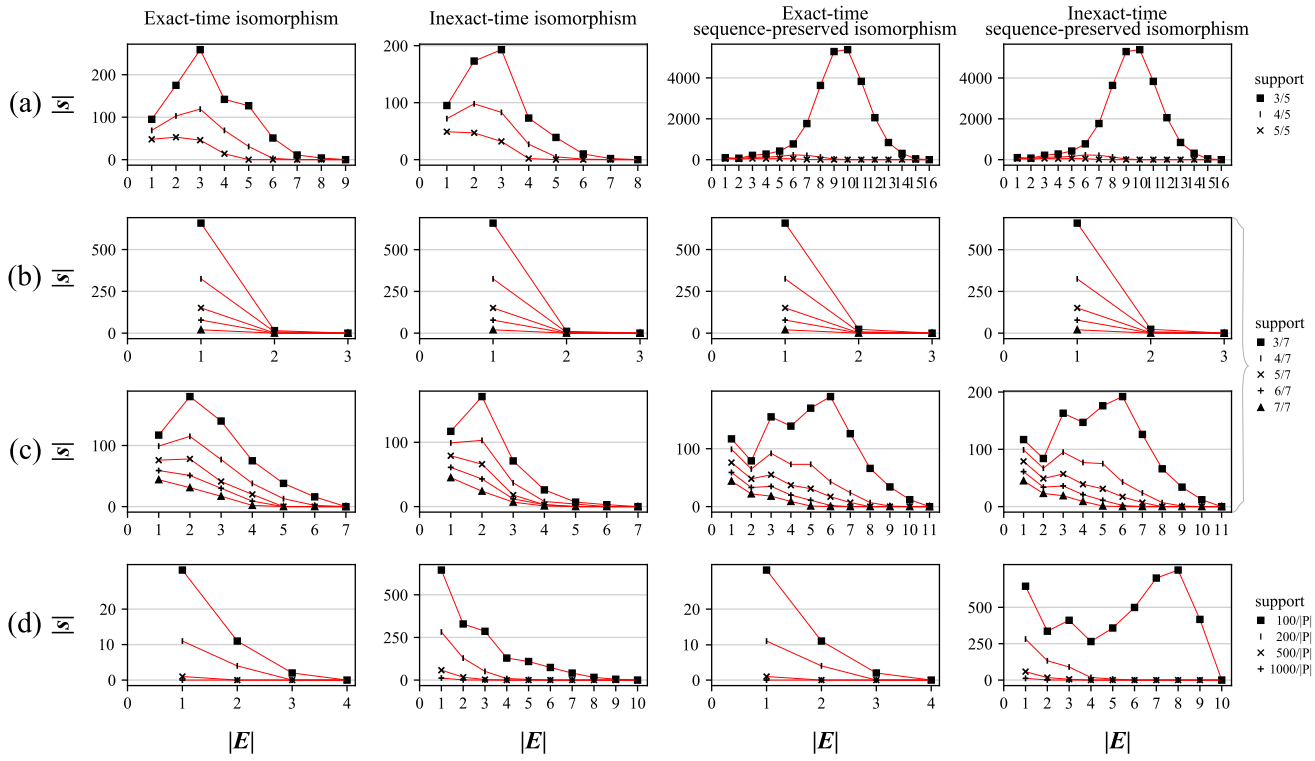

Fig. 12. Number of frequent patterns detected by the algorithm using different definitions of isomorphism from (a) hospital ward proximity, (b) high school contact networks with *id*s as labels, (c) high school contact networks with classes as labels, and (d) sepsis EHR data sets at different support thresholds ($|E|$: number of edges in the frequent subgraphs, $|s|$: number of frequent subgraphs detected, $|p|$: total number of patients in the sepsis EHR data set, 13,229).

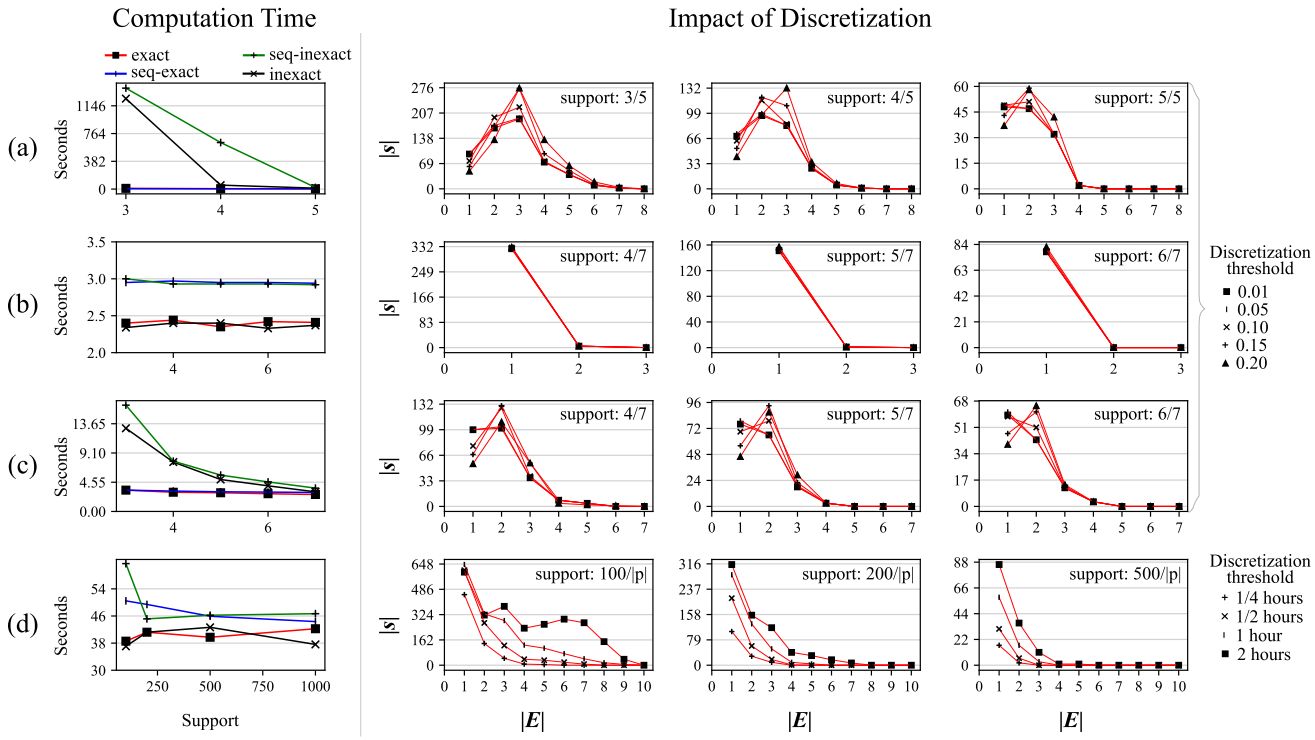

Fig. 13. Computation time (left) and impact of different discretization parameters (right) on the number of frequent subgraphs detected at different support thresholds for (a) hospital ward proximity networks, (b) high school contact networks with *id*s as labels, (c) high school contact networks with classes as labels, and (d) sepsis EHR data sets. For discretization parameters, the inexact version of isomorphism is used. ($|E|$: number of edges in the frequent subgraphs, $|s|$: number of frequent subgraphs detected, $|p|$: total number of patients in the sepsis EHR data set, 13,229).

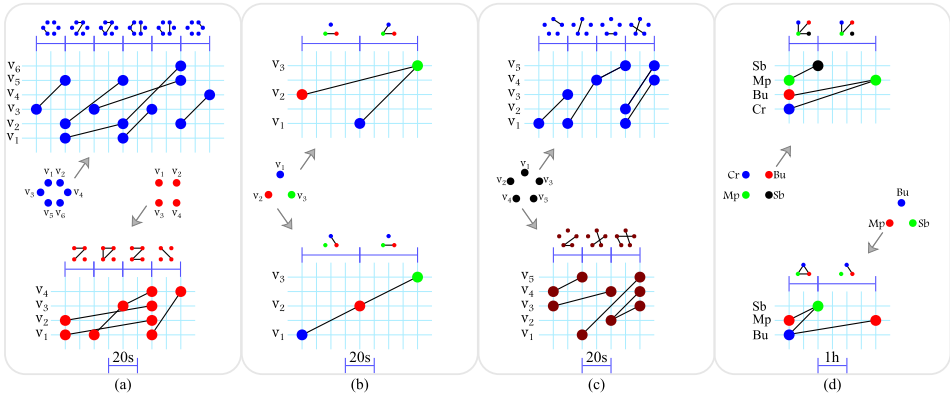

Fig. 14. Some of the frequent patterns detected in (a) hospital ward proximity networks, (b) high school contact networks with *id*s as labels, (c) high school contact networks with classes as labels, and (d) sepsis EHR data sets. Vertices with the same labels are shown with identical colors. Each row represents one vertex. The interactions of pairs of vertices over time are shown as edges spanning from left to right. Also, for each of the continuous-time temporal patterns detected, a typical network representation is provided above the pattern. Note that when these patterns are frequent, any subgraph of these patterns is frequent as well, based on the downward closure property (s: seconds, h: hour, Bu: Blood urea nitrogen, Cr: Creatinine, Mp: Mean arterial pressure, Sb: Systolic blood pressure).

or for sepsis EHR data set for other two modes of isomorphism (Fig. 12-(d)), the algorithm could find many frequent patterns at different support thresholds.

It should be noted that although adoption of inexact-time isomorphism increased the number of frequent patterns detected, it might decrease the total number of frequent patterns in some cases. In fact, inexact-time isomorphism has two opposing effects. It may group some infrequent patterns into one frequent pattern (if they are isomorphic based on the inexact-time isomorphism definition). In this case, the number of frequent subgraphs increases. Simultaneously, some of the different frequent subgraphs detected based on the exact-time isomorphism might be considered one subgraph based on the inexact-time isomorphism definition, consequently decreasing the total number of frequent subgraphs.

In all the data sets, we observed that using the sequence-preserved definitions almost always results in a higher number of frequent patterns compared to their exact-time and inexact-time counterparts, respectively. It is because we are relaxing the constraints related to the overlaps between edges, as far as they follow the same sequence of edge appearances.

Implementing the algorithm for different discretization parameters shows that these variations impact data sets differently (Fig. 13). In the first and second data sets, using different discretization parameters does not significantly change the number of frequent patterns. However, due to the nature of the sepsis EHR data set and the temporal precision in data collection, we do not have many frequent exactly identical patterns common among patients. However, as we increase the discretization parameters and allow higher tolerance of temporal variations, we observe many frequent patterns among the patients. Also, it should be noted that the opposing effect of inexact-time isomorphism discussed earlier can affect the number of frequent subgraphs detected for various discretization parameters differently as well.

The three data sets used in this study have some significant differences that directly impact the frequent subgraph mining computation time. The first data set is composed of five networks. The average number of vertices and edges in this data set is 48 and 2,806, respectively, and each vertex is labeled with one of the four possible labels. On the other hand, although the second data set comprises seven networks (about the same number of networks as the first data set), it is composed of a larger number of vertices (about three times more than the first data set) with almost the same number of edges (and nearly the same number of vertex labels). This difference between the number of vertices in these two data sets results in the lower density in the second data set and consequently decreases the computation time (Fig. 13-(a) versus Fig. 13-(c)). When the vertices in the second data set are labeled with unique *id*s of students, it results in a lower number of frequent patterns and, accordingly, significantly lower computation time (Fig. 13-(b) versus Fig. 13-(c)). In the sepsis EHR data set, we have a maximum of 19 vertices in each of 13,229 networks. However, all the vertices are uniquely labeled with one of the associated cellular and physiological responses or biomarkers they represent. Also, not all of them are necessarily present in the patients' hospitalization records. The labeling approach significantly reduces the computation time required for performing the graph isomorphism (Fig. 13-(d)).

Fig. 14 shows two sample frequent subgraphs detected in the four implementations of the algorithm on the three data sets. Nearly all the frequent subgraphs detected in the first data set were composed of paramedical staff interacting with each other (Fig. 14-(a)-top). However, the algorithm also detected a few frequent patterns among medical doctors 14-(a)-bottom). The exact-time instances of these patterns are observed in at least three (out of 5) networks of the data set. Fig. 14-(b) and (c) show four frequent temporal patterns observed in at least 4 (out of 7) networks of the second data set in the *id*s as labels and class as labels implementations, respectively. The algorithm detected a few frequent patterns in the second data set with *id*s as labels with more than one temporal edge. These patterns are among three students interacting in different formations, such as three specific students interact for some particular time at the same time, one student joins the other two students somewhere in between of the first two students' interaction 14-(b)-top), or two

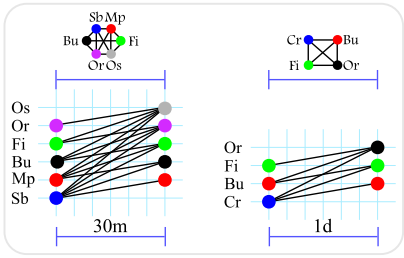

Fig. 15. Co-failure of multiple cellular and physiological responses in patients with sepsis over the same period of time (m: minutes, d: day, Bu: Blood urea nitrogen, Cr: Creatinine, Fi: Fraction of inspired oxygen (FiO_2), Or: SpO_2/FiO_2, Os: Oxygen (O_2) source, Mp: Mean arterial pressure, Sb: Systolic blood pressure).

students are interacting for some time, and one of them ends this interaction and start another interaction with a different student for a while 14-(b)-bottom. When the students are labeled with classes, there are many frequent patterns with a larger number of temporal edges. However, most of the patterns are among students of the same class 14-(c).

Finally, Fig. 14-(d) shows two patterns identified as frequent patterns in at least 100 sepsis patients by running the algorithm in the inexact-time isomorphism mode. Although these patterns show some changes in the patterns' edges over time, most of the frequent patterns detected in the sepsis EHR data set are related to the simultaneous failures of multiple cellular and physiological responses over some period of time (Fig. 15). It can be attributed to the nature of laboratory test measurements performed almost simultaneously in pre-defined intervals. These patterns can also be considered complete networks, as all the vertices are connected with one another. Both patterns are identified using the algorithm in the inexact-time sequence-preserved setting, one with 15 minutes duration deviation tolerance (left) and the other one with one-hour duration deviation tolerance (right).

Given the novel nature of the proposed method in extracting patterns unattainable by existing approaches, conventional benchmarking methods are not applicable for performance evaluation. Nevertheless, the performance of the proposed approach can be assessed through alternative methods. One crucial aspect of evaluation involves exploring the significance of patterns detected by the proposed approach in critical real-world scenarios. For instance, in a medical context, we investigated the importance of considering these patterns in predicting the outcomes of sepsis patients. The results demonstrated the potential of integrating these discovered patterns with traditional features used for patient outcome prediction, leading to significant performance improvements and early identification of deteriorating patient conditions [81]. Furthermore, evaluating the efficiency of the proposed approach is essential for practical applications. One way to achieve this is by measuring the algorithm's execution time under various scenarios and data set sizes. In our research, we provided an analysis of the approach's computational complexities and experimental execution times to better understand its scalability and resource requirements.

VII. Conclusion

This study proposed a novel approach for mining the complete set of frequent patterns in continuous-time temporal networks. A novel representation of temporal networks for frequent pattern mining is described. We developed an original method for continuous-time temporal network traversal and considered four types of isomorphism to detect frequent subgraphs in a temporal network data set.

Additionally, we tested the proposed algorithms on three real-world data sets, and the results revealed considerable variations in interactions' durations in many of these patterns (e.g., refer to Fig. 14). This suggests that the transformation of continuous-time temporal networks into sequences of networks might hinder the detection of these patterns. Moreover, this transformation necessitates prior knowledge of the appropriate interval width, which may not be feasible in numerous real-world applications.

One avenue for future research would be mining frequent patterns in network data streams. Most of the previous work defines network streams as updating batches of network components at discrete intervals [56], [82]. Investigating the frequent pattern mining problem, including the data structure requirements, for temporal networks with continuous updates of network components would be a challenging future direction with a wide variety of applications. Furthermore, in our future work, we want to use the outcome of the proposed algorithm for mining the evolution among frequent patterns, where patterns might emerge, merge, shrink, or grow.

In conclusion, our approach represents a novel contribution to the field of continuous-time temporal networks, offering promising results in mining frequent patterns. There remain opportunities for further enhancements that could significantly amplify its effectiveness. Future research endeavors could focus on refining and optimizing various steps within our proposed framework, including the construction and reconstruction of CIGs, along with the formulation of pruning techniques to circumvent graph isomorphism verifications, ultimately leading to even more remarkable performance gains.

Acknowledgment

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] L. V. Bertalanffy, *General System Theory: Foundations, Development, Applications*. New York, NY, USA: G. Braziller, 1972.
[2] M. E. J. Newman, "The structure and function of complex networks," *SIAM Rev.*, vol. 45, no. 2, pp. 167–256, 2003.
[3] L. da Fontoura Costa et al., "Analyzing and modeling real-world phenomena with complex networks: A survey of applications," *Adv. Phys.*, vol. 60, no. 3, pp. 329–412, 2011.
[4] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *Knowl. Eng. Rev.*, vol. 28, no. 1, pp. 75–105, 2012.
[5] K. Yoshida, H. Motoda, and N. Indurkhya, "Graph-based induction as a unified learning framework," *Appl. Intell.*, vol. 4, no. 3, pp. 297–316, 1994.
[6] C. Borgelt and M. R. Berthold, "Mining molecular fragments: Finding relevant substructures of molecules," in *Proc. IEEE Int. Conf. Data Mining*, 2002, pp. 51–58.

[7] J. Tang, M. Musolesi, C. Mascolo, V. Latora, and V. Nicosia, "Analysing information flows and key mediators through temporal centrality metrics," in *Proc. 3rd Workshop Social Netw. Syst.*, 2010, pp. 1–6. [Online]. Available: https://doi.org/10.1145/1852658.1852661

[8] V. Kostakos, "Temporal graphs," *Physica A, Stat. Mechanics Appl.*, vol. 388, no. 6, pp. 1007–1023, 2009.

[9] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, "Temporal motifs in time-dependent networks," *J. Stat. Mechanics, Theory Experiment*, vol. 2011, no. 11, Nov. 2011, Art. no. P11005. [Online]. Available: https://doi.org/10.1088%2F1742--5468%2F2011%2F11%2Fp11005

[10] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal networks*. Berlin, Germany: Springer, 2013, pp. 15–40.

[11] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, "Characterising temporal distance and reachability in mobile and online social networks," Jan. 2010. [Online]. Available: https://doi.org/10.1145/1672308.1672329

[12] D. Kempe, J. Kleinberg, and A. Kumar, "Connectivity and inference problems for temporal networks," *J. Comput. Syst. Sci.*, vol. 64, no. 4, pp. 820–842, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000002918295

[13] R. K. Pan and J. Saramäki, "Path lengths, correlations, and centrality in temporal networks," *Phys. Rev. E*, vol. 84, Jul. 2011, Art. no. 016105. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.84.016105

[14] P. Holme and J. Saramäki, "Temporal networks," *Phys. Rep.*, vol. 519, no. 3, pp. 97–125, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0370157312000841

[15] P. Holme, "Modern temporal network theory: A colloquium," *Eur. Phys. J. B*, vol. 88, no. 9, Sep. 2015, Art. no. 234. [Online]. Available: https://doi.org/10.1140/epjb/e2015--60657-4

[16] L. Dehaspe, H. Toivonen, and R. D. King, "Finding frequent substructures in chemical compounds," in *Proc. Int. Conf. Knowl. Discov. Data Mining*, 1998, pp. 30–36.

[17] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Principles of Data Mining and Knowledge Discovery*, D. A. Zighed, J. Komorowski, and J. Żytkow Eds., Berlin, Germany: Springer, 2000, pp. 13–23.

[18] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proc. IEEE Int. Conf. Data Mining*, 2001, pp. 313–320.

[19] Y. Ren, K. Zhang, and Y. Shi, "Survival prediction from longitudinal health insurance data using graph pattern mining," in *Proc. IEEE Int. Conf. Bioinf. Biomed.*, 2019, pp. 1104–1108.

[20] C. Sun, Q. Li, L. Cui, H. Li, and Y. Shi, "Heterogeneous network-based chronic disease progression mining," *Big Data Mining Analytics*, vol. 2, no. 1, pp. 25–34, 2019.

[21] X. Cui et al., "Classification of Alzheimer's disease, mild cognitive impairment, and normal controls with subnetwork selection and graph kernel principal component analysis based on minimum spanning tree brain functional network," *Front. Comput. Neurosci.*, vol. 12, 2018, Art. no. 31. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2018.00031

[22] J. Du, L. Wang, B. Jie, and D. Zhang, "Network-based classification of ADHD patients using discriminative subnetwork selection and graph kernel PCA," *Computerized Med. Imag. Graph.*, vol. 52, pp. 82–88, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0895611116300350

[23] D. Zhang, L. Tu, L.-J. Zhang, B. Jie, and G.-M. Lu, "Subnetwork mining on functional connectivity network for classification of minimal hepatic encephalopathy," *Brain Imag. Behav.*, vol. 12, no. 3, pp. 901–911, 2018.

[24] R. Kavuluru, M. Ramos-Morales, T. Holaday, A. G. Williams, L. Haye, and J. Cerel, "Classification of helpful comments on online suicide watch forums," in *Proc. 7th ACM Int. Conf. Bioinf. Comput. Biol. Health Inform.*, 2016, pp. 32–40. [Online]. Available: https://doi.org/10.1145/2975167.2975170

[25] Z. Deng et al., "AirVis: Visual analytics of air pollution propagation," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 1, pp. 800–810, Jan. 2020.

[26] X. Li, Y. Cheng, G. Cong, and L. Chen, "Discovering pollution sources and propagation patterns in urban area," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1863–1872.

[27] A. Mrzic et al., "Grasping frequent subgraph mining for bioinformatics applications," *BioData Mining*, vol. 11, no. 1, pp. 20–24, 2018.

[28] Q. Chen, C. Lan, B. Chen, L. Wang, J. Li, and C. Zhang, "Exploring consensus RNA substructural patterns using subgraph mining," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 5, pp. 1134–1146, 2017. [Online]. Available: https://doi.org/10.1109/TCBB.2016.2645202

[29] F. C. Queiroz, A. M. Vargas, M. G. Oliveira, G. V. Comarela, and S. A. Silveira, "ppiGReMLIN: A graph mining based detection of conserved structural arrangements in protein-protein interfaces," *BMC Bioinf.*, vol. 21, no. 1, pp. 1–25, 2020.

[30] J. Wu and L. Zhou, "DOBNet: Exploiting the discourse of deception behaviour to uncover online deception strategies," *Behav. Inf. Technol.*, vol. 34, no. 9, pp. 936–948, 2015.

[31] G. A. Wang, H. J. Wang, J. Li, A. S. Abrahams, and W. Fan, "An analytical framework for understanding knowledge-sharing processes in online Q&A communities," *ACM Trans. Manage. Inf. Syst.*, vol. 5, no. 4, pp. 1–31, Dec. 2014. [Online]. Available: https://doi.org/10.1145/2629445

[32] G. Bachi, M. Coscia, A. Monreale, and F. Giannotti, "Classifying trust/distrust relationships in online social networks," in *Proc. Int. Conf. Privacy Secur. Risk Trust Int. Conf. Social Comput.*, 2012, pp. 552–557.

[33] N. Acosta-Mendoza, A. Gago-Alonso, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. E. Medina-Pagola, "Improving graph-based image classification by using emerging patterns as attributes," *Eng. Appl. Artif. Intell.*, vol. 50, pp. 215–225, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197616000348

[34] M. Dammak, M. Mejdoub, and C. B. Amar, "Histogram of dense subgraphs for image representation," *IET Image Process.*, vol. 9, no. 3, pp. 184–191, 2014.

[35] N. Acosta-Mendoza, A. Gago-Alonso, and J. E. Medina-Pagola, "Frequent approximate subgraphs as features for graph-based image classification," *Knowl.-Based Syst.*, vol. 27, pp. 381–392, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705111002668

[36] N. B. Aoun, M. Mejdoub, and C. B. Amar, "Graph-based approach for human action recognition using spatio-temporal features," *J. Vis. Commun. Image Representation*, vol. 25, no. 2, pp. 329–338, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1047320313001910

[37] N. B. Aoun, H. Elghazel, and C. B. Amar, "Graph modeling based video event detection," in *Proc. Int. Conf. Innov. Inf. Technol.*, 2011, pp. 114–117.

[38] N. B. Aoun, M. Mejdoub, and C. B. Amar, "Bag of sub-graphs for video event recognition," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2014, pp. 1547–1551.

[39] A. Abusnaina, H. Alasmary, M. Abuhamad, S. Salem, D. Nyang, and A. Mohaisen, "Subgraph-based adversarial examples against graph-based IoT malware detection systems," in *Computational Data and Social Networks*, A. Tagarelli and H. Tong Eds., Berlin, Germany: Springer, 2019, pp. 268–281.

[40] N. Asrafi, "Comparing performances of graph mining algorithms to detect malware," in *Proc. ACM Southeast Conf.*, 2019, pp. 268–269. [Online]. Available: https://doi.org/10.1145/3299815.3314485

[41] V. Herrera-Semenets and A. Gago-Alonso, "A novel rule generator for intrusion detection based on frequent subgraph mining," *Ingeniare. Revista chilena de ingeniería*, vol. 25, no. 2, pp. 226–234, 2017.

[42] V. Herrera-Semenets, N. Acosta-Mendoza, and A. Gago-Alonso, "A framework for intrusion detection based on frequent subgraph mining," in *Proc. 2nd SDM Workshop Mining Netw. Graphs*, 2015.

[43] T. Wüchner, A. Cisłak, M. Ochoa, and A. Pretschner, "Leveraging compression-based graph mining for behavior-based malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 99–112, Jan./Feb. 2019.

[44] A. Jazayeri and C. C. Yang, "Frequent subgraph mining algorithms in static and temporal graph-transaction settings: A survey," *IEEE Trans. Big Data*, vol. 8, no. 6, pp. 1443–1462, Dec. 2022.

[45] A. Jazayeri and C. C. Yang, "Motif discovery algorithms in static and temporal networks: A survey," *J. Complex Netw.*, vol. 8, no. 4, 2020, Art. no. cnaa031.

[46] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002. [Online]. Available: https://science.sciencemag.org/content/298/5594/824

[47] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proc. IEEE Int. Conf. Data Mining*, 2002, pp. 721–724.

[48] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," University of Illinois at Urbana-Champaign, Tech. Rep. UIUCDCS-R-2002–2296, 2002.

[49] M. Lahiri and T. Y. Berger-Wolf, "Structure prediction in temporal networks using frequent subgraphs," in *Proc. IEEE Symp. Comput. Intell. Data Mining*, 2007, pp. 35–42.

[50] M. Lahiri and T. Y. Berger-Wolf, "Mining periodic behavior in dynamic social networks," in *Proc. IEEE 8th Int. Conf. Data Mining*, 2008, pp. 373–382.
[51] C. H. You, L. B. Holder, and D. J. Cook, "Graph-based data mining in dynamic networks: Empirical comparison of compression-based and frequency-based subgraph mining," in *Proc. IEEE Int. Conf. Data Mining Workshops*, 2008, pp. 929–938.
[52] C. Robardet, "Constraint-based pattern mining in dynamic graphs," in *Proc. IEEE 9th Int. Conf. Data Mining*, 2009, pp. 950–955.
[53] B. Wackersreuther, P. Wackersreuther, A. Oswald, C. Böhm, and K. M. Borgwardt, "Frequent subgraph discovery in dynamic networks," in *Proc. 8th Workshop Mining Learn. Graphs*, 2010, pp. 155–162. [Online]. Available: https://doi.org/10.1145/1830252.1830272
[54] A. Inokuchi and T. Washio, "A fast method to mine frequent subsequences from graph sequence data," in *Proc. IEEE 8th Int. Conf. Data Mining*, 2008, pp. 303–312.
[55] A. Inokuchi and T. Washio, "Mining frequent graph sequence patterns induced by vertices," in *Proc. SIAM Int. Conf. Data Mining*, 2010, pp. 466–477. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972801.41
[56] A. Cuzzocrea, Z. Han, F. Jiang, C. K. Leung, and H. Zhang, "Edge-based mining of frequent subgraphs from graph streams," *Procedia Comput. Sci.*, vol. 60, pp. 573–582, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091502311X
[57] A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà, "Mining frequent closed graphs on evolving data streams," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2011, pp. 591–599. [Online]. Available: https://doi.org/10.1145/2020408.2020501
[58] D. J. Cook and L. B. Holder, "Substructure discovery using minimum description length and background knowledge," *J. Artif. Intell. Res.*, vol. 1, no. 1, pp. 231–255, Feb. 1994.
[59] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph*," *Data Mining Knowl. Discov.*, vol. 11, no. 3, pp. 243–271, Nov. 2005. [Online]. Available: https://doi.org/10.1007/s10618--005-0003-9
[60] R. Jin, S. McCallen, and E. Almaas, "Trend motif: A graph mining approach for analysis of dynamic complex networks," in *Proc. IEEE 7th Int. Conf. Data Mining*, 2007, pp. 541–546.
[61] K. M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther, "Pattern mining in frequent dynamic subgraphs," in *Proc. 6th Int. Conf. Data Mining*, 2006, pp. 818–822.
[62] A. Ray, L. B. Holder, and S. Choudhury, "Frequent subgraph discovery in large attributed streaming graphs," in *Proc. 3rd Int. Conf. Big Data Streams Heterogeneous Source Mining Algorithms Syst. Program. Models Appl.*, 2014, pp. 166–181.
[63] T. A. McKee and F. R. McMorris, *Topics in Intersection Graph Theory*. Philadelphia, PA, USA: SIAM, 1999.
[64] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. New York, NY, USA: Academic Press, 1980.
[65] D. Zwillinger, *CRC Standard Mathematical Tables and Formulae*. Boca Raton, FL, USA: CRC, 2002.
[66] R. E. Moore, *Interval Analysis*, vol. 4. Englewood Cliffs, NJ, USA: Prentice-Hall, 1966.
[67] D. R. Fulkerson and O. A. Gross, "Incidence matrices and interval graphs," *Pacific J. Math.*, vol. 15, no. 3, pp. 835–855, 1965. [Online]. Available: https://projecteuclid.org:443/euclid.pjm/1102995572
[68] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Berlin, Germany: Springer, 2012.
[69] D. Lee and H.-I. Yu, "Interval, segment, range, and priority search trees," in *Handbook of Data Structures and Applications*. London, U.K.: Chapman and Hall/CRC, 2018, pp. 291–307.
[70] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[71] B. D. McKay and A. Piperno, "Practical graph isomorphism, II," *J. Symbolic Comput.*, vol. 60, pp. 94–112, 2014.
[72] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *Proc. Meeting Algorithm Eng. Expermiments*, 2007, pp. 135–149.
[73] S. Ramírez-Gallego et al., "Data discretization: Taxonomy and Big Data challenge," *WIREs Data Mining Knowl. Discov.*, vol. 6, no. 1, pp. 5–21, 2016.
[74] L. Babai, "Graph isomorphism in quasipolynomial time [extended abstract]," in *Proc. 48th Annu. ACM Symp. Theory Comput.*, 2016, pp. 684–697. [Online]. Available: https://doi.org/10.1145/2897518.2897542
[75] L. Babai, "Fixing the UPCC case of split-or-Johnson," 2017. [Online]. Available: https://people.cs.uchicago.edu/laci/
[76] L. Babai, "Canonical form for graphs in quasipolynomial time: Preliminary report," in *Proc. 51st Annu. ACM SIGACT Symp. Theory Comput.*, 2019, pp. 1237–1246.
[77] P. Vanhems et al., "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PLoS One*, vol. 8, no. 9, 2013, Art. no. e73970.
[78] SocioPatterns collaboration. Accessed: Apr. 01, 2021. [Online]. Available: http://www.sociopatterns.org/
[79] J. Fournet and A. Barrat, "Contact patterns among high school students," *PLoS One*, vol. 9, no. 9, 2014, Art. no. e107878. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0107878
[80] A. Jazayeri, M. Capan, C. Yang, F. Khoshnevisan, M. Chi, and R. Arnold, "Network-based modeling of sepsis: Quantification and evaluation of simultaneity of organ dysfunctions," in *Proc. 10th ACM Int. Conf. Bioinf. Comput. Biol. Health Inform.*, 2019, pp. 87–96. [Online]. Available: https://doi.org/10.1145/3307339.3342160
[81] A. Jazayeri, C. C. Yang, and M. Capan, "Frequent temporal patterns of physiological and biological biomarkers and their evolution in sepsis," *Artif. Intell. Med.*, vol. 143, 2023, Art. no. 102576. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0933365723000908
[82] P. Braun, J. J. Cameron, A. Cuzzocrea, F. Jiang, and C. K. Leung, "Effectively and efficiently mining frequent patterns from dense graph streams on disk," *Procedia Comput. Sci.*, vol. 35, pp. 338–347, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914010795

Ali Jazayeri received the BS degree in materials science and engineering from Amirkabir University of Technology, in 2006, followed by an MS degree in socio-economic systems engineering and sociology from Sharif University of Technology and the University of Tehran in 2010 and 2016, respectively, and the PhD degree from the College of Computing and Informatics (CCI), Drexel University, with his dissertation focusing on mining frequent substructures and their evolution in temporal networks. He has more than 20 publications in Artificial Intelligence in Medicine, *IEEE Transactions on Big Data*, *Journal of Complex Networks*, IEEE International Conference on Healthcare Informatics, ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, and more.

Christopher C. Yang is a professor in the College of Computing and Informatics with Drexel University. He also has a courtesy appointment with the School of Biomedical Engineering, Science, and Health Systems. He is serving as the program director in the Division of Intelligent and Information Systems (IIS), Computer and Information Science and Engineering (CISE), National Science Foundation in 2022 to 2024. He is the director of the Healthcare Informatics Research Lab. He was the Founding Director of Data Science Programs and the Program Director of MS in Health Informatics. His research interest includes data science, artificial intelligence, machine learning, fairness AI, explainable AI, healthcare informatics, social media analytics, electronic commerce, and intelligence and security informatics. He has more than 360 publications in top-tier journals, conferences, and books, such as *ACM Transactions on Intelligent Systems and Technology*, *ACM Transaction on Management Information Systems*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Computational Social Systems*, *PLOS One*, *Journal of Medical Internet Research*, *Artificial Intelligence in Medicine*, and more. He has received over $10M research fundings from NSF, NIH, DoD, PCORI, HK RGC, etc. He is the editor-in-chief of Journal of Healthcare Informatics Research and Electronic Commerce Research and Application. He is the editor of the CRC book series on Healthcare Informatics and the founding steering committee chair of the IEEE International Conference on Healthcare Informatics. He has been the general chair of more than five conferences and program chairs of more than ten conferences.