Information Sciences 733 (2026) 122930



Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins



Segmentation of temporal graphs

Raffaele Giancotti[a], Francesco Gullo[b,*], Pietro H. Guzzi[a], Edoardo Serra[c], Pierangelo Veltri[d]

[a] 'Magna Graecia' University, Catanzaro, Italy
[b] University of L'Aquila, L'Aquila, Italy
[c] Boise State University, Boise, ID, USA
[d] University of Calabria, Rende, Italy

ARTICLE INFO

Keywords:
Graph data management
Temporal graphs
Scalable data science
Data compression
Sequence segmentation

ABSTRACT

We study the problem of *segmentation of temporal graphs* (TGraphSeg): given a temporal graph – i.e., a sequence of snapshots depicting the relationships (temporal edges) occurring among entities (vertices) of interest at specific timestamps – aggregate similar consecutive snapshots and replace every aggregated snapshot set with a single, well-representative (newly computed) snapshot, to optimize a tradeoff between data reduction ratio and closeness between segmented and original graphs. This novel fundamental problem produces more compact versions of temporal graphs, thus making them more amenable to resource-efficient and responsible downstream processing, without any need for changing the technology already in place to handle them in their raw form.

The proposed TGraphSeg is formulated as an instance of *Sequence Segmentation* (SeqSeg), a well-established combinatorial optimization problem for time-series data reduction. The contextualization of SeqSeg to the temporal-graph setting implies major technical challenges, including defining a proper distance function among snapshots and a methodology to compute representative snapshots from the aggregated ones. Additionally, non-trivial implementation challenges are faced to efficiently adapt popular SeqSeg algorithms to the context at hand. Effective solutions are provided to address all these challenges. Based on these, we devise a principled formulation of TGraphSeg, along with two algorithms: an exact, more accurate one and a heuristic, faster one. Our contributions are complemented by an extensive experimental evaluation, which attests to the high performance of the proposed algorithms and their superiority over baselines on a variety of real data and the downstream tasks of vertex similarity search and temporal community detection.

Reproducibility: *Source code is available at* https://github.com/rafgia/temporal-graph-segmentation.

1. Introduction

Graphs, which are sets of entities (vertices) linked to one another (via edges), have become a ubiquitous real-world data representation model [1]. Very often, the structure of a graph – in terms of vertices and/or edges – is not fixed, rather it changes over time. This gives rise to the notion of *temporal graph*, which is a sequence of *snapshots* representing the vertices and edges at specific timestamps (see left part of Fig. 1). A static graph is a graph where vertices and edges remain fixed. Unlike static graphs, temporal graphs capture time-evolving interactions among the entities corresponding to their vertices. Specifically, a temporal graph is mainly characterized

* Corresponding author.
Email addresses: raffaele.giancotti@unicz.it (R. Giancotti), gullof@acm.org (F. Gullo), hguzzi@unicz.it (P.H. Guzzi), edoardoserra@boisestate.edu (E. Serra), pierangelo.veltri@unical.it (P. Veltri).

https://doi.org/10.1016/j.ins.2025.122930
Received 13 August 2024; Received in revised form 15 August 2025; Accepted 25 November 2025
Available online 6 December 2025
0020-0255/© 2025 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

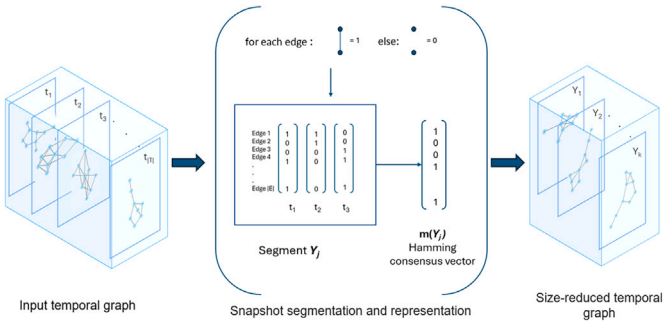


Fig. 1. Illustration of the problem we tackle in this work: segmentation of temporal graphs (TGraphSeg). Left: the input of TGraphSeg is a *temporal graph*, that is a sequence of *snapshots* (i.e., static graphs assigned to timestamps). Middle: consecutive snapshots of the input temporal graph are aggregated into segments. Right: the output of TGraphSeg is a new temporal graph, corresponding to a size-reduced version of the input temporal graph, where every segment of aggregated snapshots is replaced with a single, representative snapshot (computed as the *Hamming consensus vector*, see Section 3).

by a set $\mathcal{E} = \{(u_i, v_i, t_i)\}_{i=1}^{|\mathcal{E}|}$ of *temporal edges*. Every temporal edge $(u_i, v_i, t_i) \in \mathcal{E}$ denotes that an interaction between vertices u_i and v_i has occurred at timestamp t_i. Temporal graphs represent and analyze numerous systems/phenomena, where the timing and sequence of events are crucial, e.g., social networks, communication networks, transportation networks, and epidemic spread phenomena. For example, on social networks, friendships are a phenomenon that naturally changes over time. Thus, tracking friendship is a process that necessarily has to be carried out considering the temporal dimension. Similarly, it is crucial for epidemiological studies to incorporate the temporal dimension in order to effectively analyze the underlying disease spreading process. Yet, analysis and processing of transportation networks are typically heavily affected by phenomena such as traffic or road conditions, which are phenomena that vary over time. Temporal graphs have received considerable attention within a variety of scientific communities [2], and find application in a plethora of problems and tasks [3–7].

Motivation. In many applications, temporal graphs consist of billions (or more) of vertices and edges and hundreds of thousands (or more) of timestamps. Thus, the problem of processing them efficiently, starting from their raw representation, arises. This calls for general methodologies that can reduce the size of any input temporal graph to make its processing less resource-demanding (in terms of running time and/or memory usage). A major requirement for the problem of temporal graph reduction is that the ultimate output is still a temporal graph. This is highly desirable as it allows for retaining the same data structures, algorithms, and technology (e.g., graph databases/platforms/software libraries) already in place and used for processing temporal graphs in their raw form.

Contributions. We contribute to the general task of temporal-graph reduction by introducing the novel problem of *segmentation of temporal graphs* (TGraphSeg), and devising algorithms for it. The main general design principle of TGraphSeg consists of reducing the size of a given temporal graph by (*i*) aggregating consecutive snapshots that are similar to each other and (*ii*) replacing every such set of aggregated snapshots with a single, representative snapshot that effectively summarizes the aggregated ones. This yields a more compact version of the input temporal graph, with a resulting number of snapshots (much) fewer than the original ones, thus making it more amenable to resource-efficient and responsible processing. This process is illustrated in Fig. 1. The main objective in TGraphSeg is to perform such a snapshot aggregation and representation to achieve the best trade-off between the reduction ratio and the proximity of the reduced graph to the original. To the best of our knowledge, the TGraphSeg problem has never been studied so far.

We map the TGraphSeg problem to the well-known *Sequence Segmentation* (SeqSeg) problem, a long-established problem in time-series data reduction [8]. The problem SeqSeg consists of partitioning a sequence of numerical values ordered in time into a user-defined number of target segments (that is, groups of contiguous values). Such partitioning is pursued to minimize the distances between the overall time-series data input and the values representing the target segments (called segment-representative values), according to a user-defined distance function (e.g., Euclidean distance). The set of all segment-representative values constitutes the output-reduced version of the input time series.

We leverage the SeqSeg techniques to address our TGraphSeg problem by treating the evolving relationships in temporal graphs similarly to the patterns found in time-series data. By applying SeqSeg methods, it is possible to identify significant transitions and stable periods in temporal graphs, allowing us to segment the graph based on changes in connectivity over time. Specifically, exploiting SeqSeg for temporal-graph segmentation the way we do in our approach allows for identifying cohesive segments of highly similar snapshots. Thus, by identifying segments this way, the approach allows capturing the most important temporal transitions and stable periods in the underlying temporal graph. As such, the resulting temporal-graph reduction is expected to be well-representative of the main original temporal patterns, and not to negatively affect downstream tasks in any significant way. This is demonstrated by our experiments (see Section 6).

Adapting SeqSeg to our context is particularly challenging as we manage (time-ordered) sequences of snapshots (i.e., graphs), rather than numerical values. Our main idea to properly tackle this challenging setting is to interpret the snapshot of any timestamp as a binary vector whose entries conceptually correspond to edges. Entries of such a vector equal to one correspond to the edges of that snapshot. This is a simple design choice, principled and effective, for two main reasons. The first one is that it enables the formulation of the TGraphSeg problem as an instance of a vectorial version of SeqSeg, a straightforward extension of the standard

SeqSeg operating on scalar values, where an advantage is that the classic algorithms for SeqSeg can be adapted to work for our TGraphSeg problem. The second reason is that it allows natural yet sound choices for segment distance and segment representative: respectively, Hamming distance and Hamming consensus vector, i.e., the vector that minimizes the sum of the Hamming distances between itself and all the vectors within that segment. This is particularly attractive from an efficiency point of view, as it is known that the Hamming consensus vector can be very easily and efficiently computed [9,10].

With the formulation of TGraphSeg as a problem SeqSeg, we can adapt the classic algorithms defined for SeqSeg to our setting TGraphSeg, namely the exact dynamic-programming algorithm [8], and the top-down greedy heuristic [11,12]. This adaptation is relatively simple from a conceptual point of view. However, in order for the resulting algorithms to work efficiently, non-trivial implementation challenges must be addressed. The second main contribution of this work is to devise practical solutions to these implementation challenges.

Benefits. This work has the following main benefits. First, it enables large-scale processing of large temporal graphs, which are widely used types of data in a wide range of scenarios and applications. This means better efficiency (in terms of running time and/or memory usage) in their processing and a more responsible and sustainable use of computing resources. Executing any downstream task on the reduced temporal graphs is more environmentally friendly. The TGraphSeg problem proposed here can be considered a novelty in the temporal graph scenario. We believe that the formulation and algorithms proposed in this work are only a starting point, which can attract the interest of the scientific community and stimulate further research on it, from several perspectives. We report on possible and new perspectives in Section 8. Due to such benefits, the proposed TGraphSeg problem has a wide range of real-world applications across various fields where relationships among entities of interest evolve, such as social media analysis, epidemic modeling, computational biology, and the design and management of real-world networks, such as transportation networks or telecommunication networks. More concretely, in social media networks, our approach can be used to detect emerging trends, evolving friendships, or influence patterns during major events like elections or viral phenomena. In transportation networks, temporal graph segmentation can identify stable traffic patterns and detect changes such as congestion spikes, route closures, or disruptions. In epidemic modeling, our approach can help identify phases in the spread of diseases, aiding in the assessment of interventions like lockdowns or vaccinations and their impact on transmission. In systems biology, temporal graphs can model gene–protein interaction networks, with segmentation helping to identify distinct phases of biological processes and offering insights into how interactions evolve under different conditions or stages of development. In telecommunication networks, temporal graph segmentation can reveal patterns of network usage, identifying peak periods and moments of low demand for better resource management.

Summary and roadmap. To summarize, the main contributions of this work are as follows:

- We provide a formulation of TGraphSeg as an instance of the well-known *Sequence Segmentation* (SeqSeg) problem for time-series data reduction (Section 3). A formulation of TGraphSeg is provided as an instance of the well-known *Sequence Segmentation* (SeqSeg) problem for time-series data reduction (Section 3). To the best of our knowledge, the TGraphSeg problem has never been tackled before.
- We devise two algorithms for TGraphSeg: (i) an exact dynamic-programming one, and (ii) a faster greedy heuristic one (Sections 4–5).
- We tested the algorithms and results confirm the high effectiveness of our proposal even in the context of vertex similarity search and temporal community detection. We provide a quantitative experimental evaluation, which attests to the high performance of the proposed algorithms, both in general and in comparison to baselines, on a variety of real data (Section 6).
- We showcase the proposed TGraphSeg problem and the algorithms we devise for it in the context of two specific downstream tasks: vertex similarity search and temporal community detection. Results confirm the high effectiveness of our proposal (see Section 6).

Ours is a pioneering work in the landscape of temporal-graph data management. As such, it has several interesting directions for future research. We present the main ones in Section 8, right after providing a discussion about the main pros and cons of our approach (Section 7) and drawing conclusions.

2. Related work

To the best of our knowledge, the problem of segmentation of temporal graphs is a brand new one, which has never been tackled in the literature, neither in the particular formulation based on sequence segmentation that we propose in this work nor in other formulations. Thus, there is no literature closely related to our proposal. However, for the sake of completeness, we provide a (brief) overview of marginally related bodies of research.

Segmentation of ego-networks. The only existing work that is somewhat close to ours is the one by Zhong*et al.* [13], who deal with the problem of *segmenting a sequence of ego-networks*. However, the problem and algorithms they defined exploit the fact that the input is a sequence of specific graphs (ego-networks) that share the same “ego” vertex. Thus, the solution proposed in [13] is not really applicable to sequences of general graphs, like those used as input in our TGraphSeg problem. Moreover, their proposal also relies on the fact that ego networks are usually limited in their number of vertices and edges. Approximation/heuristic algorithms

for the **NP**-hard Jaccard Median problem, to run in reasonable time are also used. On the contrary, the here proposed algorithms can run efficiently on big general graphs (see Section 6).

Graph summarization, graph compression, graph sparsification. Our TGraphSeg problem falls into the broad area of generating more compact versions of big graphs. In this area, several specific problems have been tackled, including, e.g., *graph summarization*, *graph compression*, *graph sparsification*. All these problems are mainly for non-temporal graphs, thus, they are different conceptually and algorithmically from the TGraphSeg problem.

Graph summarization produces a coarse-grained version of the original graph – a so-called *summary graph* – by grouping vertices and edges into *supernodes* and *superedges* [14]. Graph summarization has mostly been studied for non-temporal graphs. However, several attempts to extend it for dynamic/temporal graphs exist [15–18]. The problem known as *graph compression* aims to develop ad-hoc data structures (and algorithms to manipulate them) to store and retrieve an exact representation of a graph using the minimum space [19].

Graph sparsification (a.k.a., *graph simplification*, or *graph backboning*) reduces a graph by removing edges/vertices, in either a general-purpose way [20,21], or in a special-purpose one, i.e., attempting to preserve specific properties, such as shortest paths, degree distribution, or spectral characteristics [22–25].

Special-purpose segmentation of graph sequences. We pursue a *general-purpose objective*, i.e., general-purpose segmentation of a graph sequence corresponding to the snapshots of any temporal graph. This goes over the special-purpose segmentation proposals, where a segmentation is tailored to an objective function of an optimization problem, e.g., dense subgraph discovery [26], community detection [27], community search [4], graph clustering [28], network untangling [29], or vertex-label dynamics [30].

3. Problem definition

3.1. Preliminaries

We are given a *temporal graph* $\mathcal{G} = (V, T, \mathcal{E})$, where V is a set of *vertices*, $T \subseteq \mathbb{R}_0^+$ is a *finite* set of *timestamps*, where a timestamp is a positive real number, and $\mathcal{E} \subseteq V \times V \times T$ is a set of *temporal (undirected) edges*. The set $\{(u, v, t)\}$ indicates all unordered vertex pairs $u, v \in V$ and timestamps $t \in T$ such that there exists an edge between u and v at the timestamp t of the graph $\mathcal{G}$. Given a timestamp $t \in T$, $E_t = \{u, v \in V \mid (u, v, t) \in \mathcal{E}\}$ and $V_t = \{u \in V \mid \exists (u, v) \in E_t\}$ denote the set of *static* edges and vertices existing at t, respectively, and $G_t = (V_t, E_t)$ is the *(graph) snapshot* of t, i.e., the *static* graph composed of all the vertices and edges of $\mathcal{G}$ present at timestamp t. $T(u) = \{t \in T \mid u \in V_t\}$ denotes the timestamps in which $u \in V$ exists.

Binary representation. Let $E = \cup_{t \in T} E_t$ be the set of all the "non-temporal" edges of $\mathcal{G}$, i.e., the set of all the unordered pairs of vertices that are linked by at least one temporal edge.

Any snapshot of a temporal graph can be represented as a $|E|$-dimensional *one-hot encoded binary vector*, formally defined as follows.

Definition 1 (Snapshot's one-hot encoding binary representation). The one-hot encoding binary representation of a snapshot $G_t = (V_t, E_t)$ of a temporal graph $\mathcal{G} = (V, T, \mathcal{E})$ is an $|E|$-dimensional binary vector $\mathbf{y}_t$ where:

$$\mathbf{y}_t[e] = \begin{cases} 1, & \text{if } e \in E_t, \\ 0, & \text{if } e \notin E_t \end{cases} \quad \text{for all } e \in E. \tag{1}$$

According to the above definition, the whole temporal graph $\mathcal{G}$ can be equivalently represented as a $|T|$-sized sequence $\mathbf{Y} = [\mathbf{y}_t]_{t \in T}$ of $|E|$-dimensional binary vectors.

Objective. Given a temporal graph $\mathcal{G}$, we aim to reduce its size to improve analysis and tractability for large temporal phenomena representation. We aim to aggregate consecutive snapshots that are similar and substitute every set of aggregated snapshots with a single, well-representative snapshot. Thus, we focus on generating a size-reduced, yet still semantically rich version of $\mathcal{G}$ with some snapshots k, where $k \ll |T|$.

The rationale of our problem is that the more compact temporal graph can be used as substitution for the original graph, achieving less resource-demanding and more responsible downstream task processing without significantly penalizing accuracy with respect to the processing on the original graph. To achieve this we map the above reported problem to the SeqSeg optimization problem [8].

3.2. The sequence segmentation problem

The SeqSeg problem consists of partitioning a sequence of numerical values into b number of segments (with b defined as input) minimizing a penalty value for each identified segment. Without loss of generality, the values input to SeqSeg can be vectors.

Given a sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of d-dimensional numerical vectors, and a function $p : \{\mathbf{X}'\}_{\mathbf{X}' \sqsubseteq \mathbf{X}} \to \mathbb{R}$ that assigns a penalty score to every subsequence $\mathbf{X}'$ of contiguous vectors of $\mathbf{X}$ (denoted with the "$\sqsubseteq$" symbol), the SeqSeg problem consists of partitioning $\mathbf{X}$ into a set $\{\mathbf{X}_j\}_{j=1}^b$ of b subsequences such that $\sum_{j=1}^b p(\mathbf{X}_j)$ is minimized.

The choice of the penalty function $p(\cdot)$ influences the evaluation of the SeqSeg problem. E.g., $p(\cdot)$ can be the sum of the *squared Euclidean norm* of the difference between every value in a segment and the so-called "centroid" of that segment, i.e., the mean of all the values in that segment. The use of centroid is motivated by the fact that it is well-known that it corresponds to the vector that minimizes the sum of the squared Euclidean norms of the differences between itself and all the values in the segment at hand.

Formally, given a segment $\mathbf{S} = [\mathbf{x}_{h_1}, \dots, \mathbf{x}_{h_2}]$ and $\mu_{\mathbf{S}} = \frac{1}{|\mathbf{S}|} \sum_{i=h_1}^{h_2} \mathbf{x}_i$, this choice of penalty function corresponds to:

$$p(\mathbf{S}) = \sum_{i=h_1}^{h_2} \|\mathbf{x}_i - \mu_{\mathbf{S}}\|_2^2. \tag{2}$$

Example 1. Let us consider the SEQSEG problem (Problem 3.2) with the penalty function of Eq. (2). Let the following be the input sequence $\mathbf{X}$:

$$\mathbf{X} = ([10,6],[10,6],[10,6],[2,2],[2,2],[2,2],[2,2],[2,5],[2,5])\,.$$

for $b = 3$ is:

$$\Big\{ \underbrace{([10,6],[10,6],[10,6])}_{\text{segment 1}}; \underbrace{([2,2],[2,2],[2,2],[2,2])}_{\text{segment 2}}; \underbrace{([2,5],[2,5])}_{\text{segment 3}} \Big\},$$

with a total penalty equal to 0.

For $b = 2$, the solution is:

$$\Big\{ \underbrace{([10,6],[10,6],[10,6])}_{\text{segment 1}}; \underbrace{([2,2],[2,2],[2,2],[2,2],[2,5],[2,5])}_{\text{segment 2}} \Big\},$$

with total penalty equal to 12 (i.e., penalty equals to 0 on segment 1 and penalty equals to 12 on segment 2).

Also, for $b = 2$, another possible solution is:

$$\Big\{ \underbrace{([10,6],[10,6],[10,6],[2,2])}_{\text{segment 1}}; \underbrace{([2,2],[2,2],[2,2],[2,5],[2,5])}_{\text{segment 2}} \Big\},$$

with total penalty 69 (60 on segment 1 and 9 on segment 2).

Thus for $b = 3$ the optimal solution is obtained (penalty minimized), whereas for $b = 2$ the optimal solution is the first one (penalty equals to 12).

3.3. Segmentation of temporal graphs as an instance of SEQSEG

We define temporal graph segmentation as an instance of SEQSEG. Specifically, we map the input sequence $\mathbf{X}$ of SEQSEG into the sequence $\mathbf{Y} = \left[\mathbf{y}_t\right]_{t \in T}$ of all the one-hot encoded binary representations of the snapshots of the input temporal graph $\mathcal{G}$, and choose an appropriate definition for the penalty function $p(\cdot)$ among (the one-hot encoded binary representations of) the various snapshots. The latter is a major technical challenge, which we achieve by selecting the *Hamming distance*, popularly used as a distance function among binary vectors. This implies the representation of any segment as the (binary) vector that minimizes the Hamming distance between itself and all the vectors in that segment, which is termed *Hamming consensus vector*. The consensus vector represents the segment, reflecting the central tendency of the data while minimizing the dissimilarity. The Hamming consensus vector is conceptually equivalent to the notion of centroid in Euclidean space, providing an optimal representation of segment's structure based on the Hamming distance metric. This approach highlights the ability of the Hamming distance to effectively capture the similarity between snapshots in a consistent manner with computational efficiency. This mainly motivates our selection of the Hamming distance. As for other distance functions, such as, e.g., Jaccard or Graph edit Distance, it would be challenging to use them with a similar tradeoff between effectiveness and efficiency. In fact, producing the consensus vectors under those distances is known to be a computationally intensive (or even **NP**-hard) problem [31,32].

Definition 2 (Hamming distance and Hamming consensus vector). Let $\mathbf{S} = [\mathbf{y}_{h_1}, \dots, \mathbf{y}_{h_2}]$ be a sequence of $|E|$-dimensional binary vectors. The Hamming-consensus-vector $\mathbf{m}(\mathbf{S})$ of $\mathbf{S}$ is defined as:

$$\mathbf{m}(\mathbf{S}) = \text{argmin}_{\mathbf{v} \in \{0,1\}^{|E|}} \sum_{i=h_1}^{h_2} d_H\left(\mathbf{y}_i, \mathbf{v}\right), \tag{3}$$

where

$$d_H(\mathbf{a}, \mathbf{b}) = \sum_{\ell=1}^{|E|} |\mathbf{a}[\ell] - \mathbf{b}[\ell]| \tag{4}$$

denotes the Hamming distance between any two $|E|$-dimensional binary vectors $\mathbf{a}$ and $\mathbf{b}$.

Finding $\mathbf{m}(\mathbf{S})$ is also known as CONSENSUS STRING problem [9,10], and has a simple and efficiently computable solution: just set every ℓ-th component $m(\mathbf{S})[\ell]$ of $\mathbf{m}(\mathbf{S})$ as equal to the value corresponding to the majority of the ℓ-th components of the various vectors in $\mathbf{S}$. This result is formalized in the next theorem.

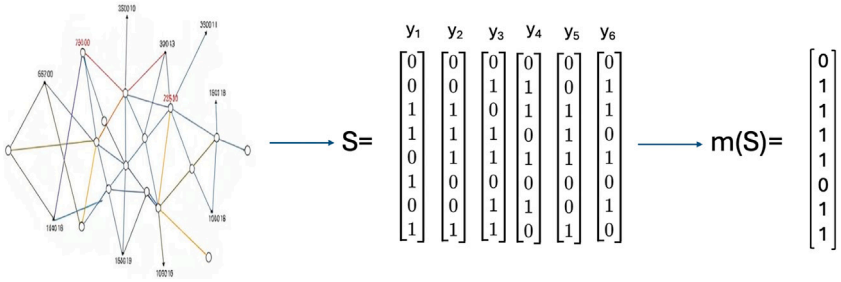


Fig. 2. Illustration of the proposed TGRAPHSEG problem (Problem 1). The sequence **S** represents the direct one-hot encoding of binary snapshots from the input temporal graph shown on the left in the figure. The Hamming consensus vector **m**(**S**) is constructed as follows: each entry of the vector is 1 if the corresponding edge is present in the majority of the snapshots of the input temporal graph, 0 otherwise.

Theorem 1 (Solution to CONSENSUS STRING problem). *The Hamming consensus vector* $\mathbf{m}(\mathbf{S})$ *of a sequence* $\mathbf{S}$ *of binary vectors is equal to:*

$$\mathbf{m}(\mathbf{S})[\ell] = \begin{cases} 1, & \text{if } \sum_{i=h_1}^{h_2} \mathbf{y}_i[\ell] > |\mathbf{S}|/2 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The penalty function $p(\cdot)$ for a given a segment $\mathbf{S} = [\mathbf{y}_{h_1}, \dots, \mathbf{y}_{h_2}]$ of snapshots (in their $|E|$-dimensional one-hot encoded binary representation), is therefore:

$$p(\mathbf{S}) = \sum_{i=h_1}^{h_2} d_H\left(\mathbf{y}_i, \mathbf{m}(\mathbf{S})\right). \tag{6}$$

In Fig. 2 an example of how Hamming consensus vector works is shown.

It is reasonable to assume that the most representative edges of a segment correspond to the ones that frequently occur in the various snapshots of the segment. Theorem 1 shows that the Hamming consensus vector exactly captures this rationale. In fact, it ensures that the most representative edges, defined by their frequency of occurrence in the graph, are preserved, while edges that appear less frequently are discarded. Moreover, by choosing the Hamming consensus vector and the corresponding Hamming distance, it is possible to reduce the temporal graph while preserving the majority of its informative content (see experimental results). The choice is based on the frequency of edges across the entire graph. Edges that appear more frequently are considered more informative. Thus, they are retained, since their presence indicates greater significance with respect to the less frequent edges.

Given the above-defined penalty function, the problem of segmentation of temporal graphs we tackle in this work is formulated as:

Problem 1 (TGRAPHSEG). Let $\mathcal{G} = (V, T, \mathcal{E})$ be a temporal graph, and let the sequence $\mathbf{Y} = \left[\mathbf{y}_t\right]_{t \in T}$ of $|E|$-dimensional binary vectors correspond to the one-hot-encoded binary representations of the snapshots of $\mathcal{G}$ (according to Definition 1).

Given $\mathcal{G} = (V, T, \mathcal{E})$, $\mathbf{Y}$, and an integer $k \in [1, |T|]$, partition $\mathbf{Y}$ into a set $\{\mathbf{Y}_j\}_{j=1}^{k}$ of k subsequences such that $\sum_{j=1}^{k} p(\mathbf{Y}_j)$ is minimized, where $p(\cdot)$ is the penalty function defined in Eq. (6). Ultimately, produce a temporal graph $\hat{\mathcal{G}} = (V, \hat{T}, \hat{\mathcal{E}})$ with k snapshots, such that every (one-hot-encoded binary representation of a) snapshot j of $\hat{\mathcal{G}}$ corresponds to the Hamming consensus vector of $\mathbf{Y}_j$, for all $j = 1, \dots, k$.

A nice peculiarity of formulating our TGRAPHSEG problem like this is that, once snapshots have been partitioned into segments, every segment can be easily represented by its corresponding Hamming consensus vector. This leads naturally to the resulting compact form of temporal graph, which corresponds to a temporal graph with k snapshots, where every snapshot j corresponds to the snapshot represented by the Hamming consensus vector of segment j. Therefore, the core algorithmic part in solving Problem 1 lies in the snapshot segmentation step.

Adapting SEQSEG to come up with the TGRAPHSEG problem formulation basically consists of feeding SEQSEG with vectors that correspond to the one-hot encoded binary representations of the snapshots of the input temporal graph. Despite this adaptation being rather simple, it leads to nontrivial algorithmic challenges, which are discussed in Sections 4 and 5. An illustration of the TGRAPHSEG problem is shown in Figs. 2 and 3.

4. An exact algorithm for TGRAPHSEG

The first algorithm we propose for our TGRAPHSEG problem is an adaptation of the well-established *exact* dynamic-programming algorithm for SEQSEG [8] to the temporal-graph setting. This is an optimization algorithm that optimally solves SEQSEG. In the following, we first overview SEQSEG's algorithm, and then we show how to adapt it in the context of TGRAPHSEG.

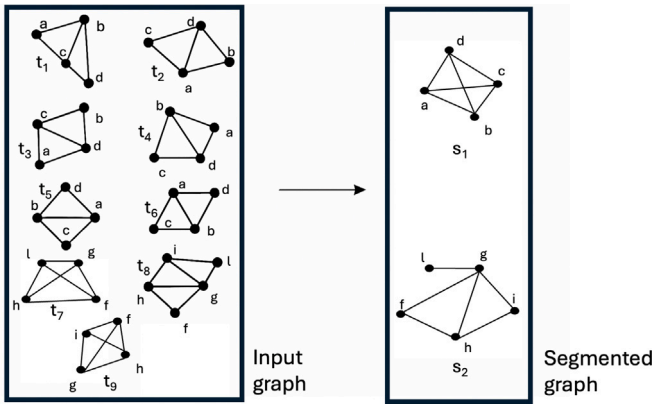


Fig. 3. Further illustration of the proposed TGRAPHSEG problem (Problem 1). In this toy example, the input to TGRAPHSEG is a temporal graph composed of nine snapshots, each of which is represented as a static graph (left). The first six snapshots are very similar to each other, as well as the last three ones. Segmenting the graph into a number $k = 2$ of segments according to the TGRAPHSEG problem produces the two segments (right) that aggregate the first six snapshots and the last three snapshots, respectively.

4.1. The dynamic programming algorithm for SEQSEG

The exact dynamic-programming algorithm for SEQSEG [8], termed SeqSeg-DP, is based on the following main recursive rule: the optimal segmentation of a subsequence $[\mathbf{x}_1, \dots, \mathbf{x}_i]$ of the input $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ sequence into j segments ($j = 2, \dots, b$) is given by the minimum over $s = 1, \dots, i-1$ of the optimal segmentation of $[\mathbf{x}_1, \dots, \mathbf{x}_s]$ into $j-1$ segments plus the penalty of the additional $[\mathbf{x}_{s+1}, \dots, \mathbf{x}_i]$ segment. Formally:

$$\begin{aligned} &p^*([\mathbf{x}_1, \dots, \mathbf{x}_i], j) = \\ &= \min_{s=1,\dots,i-1} p^*([\mathbf{x}_1, \dots, \mathbf{x}_s], j-1) + p([\mathbf{x}_{s+1} \dots \mathbf{x}_i]), \\ &\text{for all } i = 1, \dots, n, \;\; j = 2, \dots, \min\{i, b\}, \end{aligned} \tag{7}$$

where $p^*(\mathbf{Z}, c)$ denotes the cost of the optimal segmentation of a sequence $\mathbf{Z}$ into c segments.

The pseudocode of the SeqSeg-DP algorithm is reported in Algorithm 1. Essentially, it populates a penalty matrix $\mathbf{P}$ and a reconstruction matrix $\mathbf{R}$ based on the recursive rule in Eq. (7). The optimal solution is the one in the last computed cell of matrix $\mathbf{P}$, and can be reconstructed by properly back-navigating matrix $\mathbf{R}$.

Time complexity. The time complexity of Algorithm 1 is $\mathcal{O}(n^2 \times b + \sum_{i=1}^{n} \tau_p(i))$, where $\tau_p(i)$ is the time needed for computing the penalty $p(\cdot)$ of all the subsequences $\{[\mathbf{x}_1, \dots, \mathbf{x}_s]\}_{s=1}^{i}$. The $\mathcal{O}(n^2 \times b)$ factor depends on the fact that the algorithm computes every $[i, j]$ cell of the $(n \times b)$-dimensional matrices $\mathbf{P}$ and $\mathbf{R}$ by finding the minimum over i elements, for $i = 1, \dots, n$ (Lines 8 and 9). The $\mathcal{O}(\sum_{i=1}^{n} \tau_p(i))$ factor is obtained by the penalty-computation step at Line 4. If the squared Euclidean distance between all the vectors in a segment and the centroid of that segment is used as a penalty, the time $\tau_p(i)$ for computing the penalty of all the subsequences $\{[\mathbf{x}_1, \dots, \mathbf{x}_s]\}_{s=1}^{i}$ is $\mathcal{O}(i \times d)$, because incremental distance computation is possible as follows. Given a set $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_q\}$ of d-dimensional numerical vectors, it holds that the sum of the squared Euclidean distances between the centroid $\mu(\mathbf{Z}) = \frac{1}{q} \sum_{i=1}^{q} \mathbf{z}_i$ and all the vectors in $\mathbf{Z}$ can be computed as follows:

$$\begin{aligned} \sum_{i=1}^{q} \left(\mathbf{z}_i - \mu(\mathbf{Z})\right)^2 &= \sum_{i=1}^{q} \mathbf{z}_i^2 - 2\mu(\mathbf{Z}) \sum_{i=1}^{q} \mathbf{z}_i + q \times (\mu(\mathbf{Z}))^2 = \\ &= \sum_{i=1}^{q} \mathbf{z}_i^2 - 2\frac{\left(\sum_{i=1}^{q} \mathbf{z}_i\right)^2}{q} + q\frac{\left(\sum_{i=1}^{q} \mathbf{z}_i\right)^2}{q^2} = \\ &= \underbrace{\sum_{i=1}^{q} \mathbf{z}_i^2}_{\alpha_2(\mathbf{Z})} - \frac{1}{q}\Big(\underbrace{\sum_{i=1}^{q} \mathbf{z}_i}_{\alpha(\mathbf{Z})}\Big)^2 = \alpha_2(\mathbf{Z}) - \frac{1}{q}(\alpha(\mathbf{Z}))^2. \end{aligned}$$

Exploiting this rearrangement, the penalties $p([\mathbf{x}_s, \dots \mathbf{x}_i])$, for all $s = 1, \dots, i$ (see line 4 of Algorithm 1) can be computed in $\mathcal{O}(i \times d)$ time as follows:

1: $\alpha \leftarrow 0, \;\; \alpha_2 \leftarrow 0$
2: **for** $s = i, i-1, \dots, 1$ **do**
3: $\quad \alpha \leftarrow \alpha + \mathbf{x}_i, \;\; \alpha_2 \leftarrow \alpha_2 + \mathbf{x}_i^2$
4: $\quad p([\mathbf{x}_s, \dots \mathbf{x}_i]) \leftarrow \alpha_2 - \frac{1}{i-s+1}\alpha^2$
5: **end for**

Algorithm 1 SeqSeg-DP [8].

Input: A sequence $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ of d-dimensional numerical vectors, a penalty function $p\colon \{\mathbf{X}'\}_{\mathbf{X}' \sqsubseteq \mathbf{X}} \to \mathbb{R}$, an integer $b \in [1, n]$.
Output: A partition $\{\mathbf{X}_j\}_{j=1}^{b}$ of $\mathbf{X}$ into b segments.

▷ *Initialization*
1: $\mathbf{P} \leftarrow$ an empty $(n \times b)$-dimensional matrix ▷ *Penalty matrix*
2: $\mathbf{R} \leftarrow$ an empty $(n \times b)$-dimensional matrix ▷ *Reconstruction matrix*
▷ *Dynamic-programming step*
3: **for all** $i = 1, \dots, n$ **do**
4: Compute penalties $p([\mathbf{x}_s, \dots \mathbf{x}_i])$, for all $s = 1, \dots, i$
5: $\mathbf{P}[i, 1] \leftarrow p([\mathbf{x}_1, \dots, \mathbf{x}_i])$ ▷ *Optimal cost of partitioning* $[\mathbf{x}_1, \dots, \mathbf{x}_i]$ *into 1 segment, i.e., corresponding to the penalty of* $[\mathbf{x}_1, \dots, \mathbf{x}_i]$ *itself*
6: $\mathbf{R}[i, 1] \leftarrow 0$
7: **for all** $j = 2, \dots, \min\{i, b\}$ **do**
8: $\mathbf{P}[i, j] \leftarrow \min_{s=1,\dots,i-1} \mathbf{P}[s, j-1] + p([\mathbf{x}_{s+1}, \dots, \mathbf{x}_i])$ ▷ *Optimal cost of partitioning* $[\mathbf{x}_1, \dots, \mathbf{x}_i]$ *into* j *segments*
9: $\mathbf{R}[i, j] \leftarrow \operatorname{argmin}_{s=1,\dots,i-1} \mathbf{P}[s, j-1] + p([\mathbf{x}_{s+1}, \dots, \mathbf{x}_i])$
10: **end for**
11: **end for**
▷ *Reconstructing the solution*
12: $ub \leftarrow n$
13: **for all** $j = b, \dots, 1$ **do**
14: $lb \leftarrow \mathbf{R}[ub, j]$
15: $\mathbf{X}_j \leftarrow [\mathbf{x}_{lb}, \dots, \mathbf{x}_{ub}]$
16: $ub \leftarrow lb - 1$
17: **end for**

Thus, $\sum_{i=1}^{n} \tau_p(i) = \mathcal{O}(n^2 \times d)$, the overall time complexity of Algorithm 1 when the squared Euclidean distance between all the vectors in a segment and the centroid of that segment is used as a penalty is $\mathcal{O}(n^2 \times (b + d))$.

Space complexity. Algorithm 1 takes $\mathcal{O}(n \times b)$ space, which is the space required by matrices $\mathbf{P}$ and $\mathbf{R}$. Note that, as far as matrix $\mathbf{P}$, actually there is no need to keep it in memory entirely, as only the rows of the last two iterations of the algorithm matter. Anyway, the same does not hold for matrix $\mathbf{R}$, which instead has to be kept entirely, in order to allow a sound backwards reconstruction of the solution.

4.2. The TGraphSeg-DP algorithm

The algorithm we propose for our TGraphSeg problem derives from the SeqSeg-DP algorithm (Algorithm 1). We adapted the algorithm by solving nontrivial issues to have it implemented efficiently. Specifically, efficient implementation requires (*i*) allowing incremental computation of Hamming consensus vectors and corresponding Hamming distances between them and the various vectors in a segment, and (*ii*) properly exploiting the sparsity of the binary vectors at hand.

In order to compute the Hamming-distance-based penalties in an incremental way, we adopt the following methodology. Given a temporal graph $\mathcal{G} = (V, T, \mathcal{E})$, for every timestamp $t_i \in T$ and every (non-temporal) edge $e = (u, v) \in E$, we evaluate $\mathbf{a}_{t_i}[e]$ as follows:

$$\mathbf{a}_{t_i}[e] = |\{t \in [t_1, t_i] \mid (u, v, t) \in \mathcal{E}\}|. \tag{8}$$

where $\mathbf{a}_{t_i}[e]$ denotes the number of timestamps within the $[t_1, t_i]$ interval whenever the edge e exists.

The Hamming-distance-based penalty of any interval $[t_x, t_y]$, $x, y \in [1, |T|]$, $x \leq y$, can be calculated using Eq. (8) as follows:

$$p([t_x, t_y]) = \sum_{e \in E_{[t_x, t_y]}} \min\{\mathbf{a}_{t_y}[e] - \mathbf{a}_{t_{x-1}}[e],\ y - x + 1 - (\mathbf{a}_{t_y}[e] - \mathbf{a}_{t_{x-1}}[e])\}, \tag{9}$$

where $E_{[t_x, t_y]}$ is the set of all the (non-temporal) edges existing in time interval $[t_x, t_y]$, and adopting the convention that $\mathbf{a}_{t_0}[e] = 0$, for all $e \in E$.

Algorithm 2 outlines the dynamic programming algorithm used for SeqSeg mapped to the TGraphSeg problem.

Obviously, the two algorithms differ in penalty computation, i.e., Lines 2–5, and Lines 7–13. Those parts of the algorithm exploit the aforementioned incremental penalty computation and the sparsity of the binary vectors at hand.

Time complexity. Lines 2–5 take $\mathcal{O}(|T| \times |E| + |\mathcal{E}|) = \mathcal{O}(|T| \times |E|)$ time (as $|\mathcal{E}| = \mathcal{O}(|T| \times |E|)$). Executing the block at Lines 7–13 for all $i = 1, \dots, |T|$ corresponds to processing every temporal edge $e \in \mathcal{E}$ a number $\mathcal{O}(|T|)$ of times, and every time a temporal edge is processed constant operations are performed. This way, executing Lines 7–13 for all $i = 1, \dots, |T|$ takes overall $\mathcal{O}(|T|^2 + |T| \times |\mathcal{E}|)$ time.

Algorithm 2 TGraphSeg-DP.

Input: A temporal graph $\mathcal{G} = (V, T, \mathcal{E})$, an integer $k \in [1, |T|]$.
Output: A size-reduced temporal graph $\hat{\mathcal{G}}$ with k snapshots.

1: $\mathbf{P}, \mathbf{R} \leftarrow$ two empty $(|T| \times k)$-dimensional matrices ▷ *Penalty matrix and Reconstruction matrix*
▷ *Computing auxiliary $|E|$-dimensional vectors $\{\mathbf{a}_{t_i}\}_{t_i \in T}$; $\mathbf{a}_{t_i}[e] :=$ number of timestamps within $[t_1, t_i]$ in which e exists, $\forall e \in E$ (Eq. (8))*
2: $\mathbf{a}_{t_i} \leftarrow \mathbf{0}^{|E|}$, for all $t_i \in T$
3: **for all** $i = 1, \dots, |T|$, $e \in E_{t_i}$ **do**
4: $\mathbf{a}_{t_i} \leftarrow \mathbf{a}_{t_i}[e] + 1$
5: **end for**
▷ *Dynamic programming step*
6: **for all** $i = 1, \dots, |T|$ **do**
▷ *Computing penalties for intervals $\{[t_s, t_i]\}_{s=1,\dots,i}$ (Eq. (9))*
7: **for all** $s = i, i-1, \dots, 1$ **do**
8: $p_{s,i} \leftarrow 0$
9: **for all** $t \in [t_s, t_i]$, $e \in E_t$ **do**
10: $\mathbf{a}[e] \leftarrow \mathbf{a}_{t_i}[e] - \mathbf{a}_{t_s}[e]$ if $s > 1$, $\mathbf{a} \leftarrow \mathbf{a}_{t_i}[e]$, otherwise ▷*#timest. in $[t_s, t_i]$ in which e exists*
11: $p_{s,i} \leftarrow p_{s,i} + \min\{\mathbf{a}[e], s - i + 1 - \mathbf{a}[e]\}$
12: **end for**
13: **end for**
▷ *Populating* $\mathbf{P}$, $\mathbf{R}$ *matrices*
14: $\mathbf{P}[i, 1] \leftarrow p_{1,i}$ ▷ *Optimal cost of partitioning $[t_1, t_i]$ into 1 segment, i.e., penalty of $[t_1, t_i]$ itself*
15: $\mathbf{R}[i, 1] \leftarrow 0$
16: **for all** $j = 2, \dots, \min\{i, k\}$ **do**
17: $\mathbf{P}[i, j] \leftarrow \min_{s=1,\dots,i-1} \mathbf{P}[s, j-1] + p_{s+1,i}$ ▷ *Optimal cost of partitioning $[t_1, t_i]$ into j segm.*
18: $\mathbf{R}[i, j] \leftarrow \operatorname{argmin}_{s=1,\dots,i-1} \mathbf{P}[s, j-1] + p_{s+1,i}$
19: **end for**
20: **end for**
▷ *Reconstructing the partition $\{T_j\}_{j=1}^{k}$ of $T = [t_1, \dots, t_{|T|}]$ into k time intervals*
21: $ub \leftarrow |T|$
22: **for all** $j = k, \dots, 1$ **do**
23: $lb \leftarrow \mathbf{R}[ub, j]$; $T_j \leftarrow [t_{lb}, t_{ub}]$; $ub \leftarrow lb - 1$
24: **end for**
▷ *Producing the output size-reduced temporal graph $\hat{\mathcal{G}}$*
25: $\mathbf{m}_j \leftarrow$ Hamming consensus vector of T_j, for all $j = 1, \dots, k$ ▷ *Eq. (5)*
26: $\hat{T} \leftarrow \{1, \dots, j\}$; $\hat{\mathcal{E}} \leftarrow \{(u, v, j) \mid \mathbf{m}_j[(u, v)] = 1\}$; $\hat{\mathcal{G}} \leftarrow (V, \hat{T}, \hat{\mathcal{E}})$

The rest of the algorithm takes overall $\mathcal{O}(|T|^2 \times k)$ time (like the classic SeqSeg-DP algorithm). Thus, the overall time complexity of Algorithm 2 is $\mathcal{O}(|T|^2 \times k + |T| \times |\mathcal{E}|)$.

Space complexity. The auxiliary $\mathbf{a}_{t_i}$ vectors, and $\mathbf{P}$, $\mathbf{R}$ matrices take $\mathcal{O}(|T| \times |E|)$ and $\mathcal{O}(|T| \times k)$ space, respectively. Therefore, the overall space complexity of Algorithm 2 is $\mathcal{O}(|T| \times (k + |E|))$.

4.3. A space-efficient implementation of the TGraphSeg-DP algorithm

The incremental penalty computation at the basis of the TGraphSeg-DP algorithm (Algorithm 2) can be implemented in a more sophisticated yet space-efficient way, at the cost of a (small) increase in running time. The main idea is to avoid materializing all the auxiliary $\mathbf{a}_{t_i}$ vectors (Lines 2–5), and computing/storing the minimal information that is needed to reconstruct the $\mathbf{a}_{t_i}$ vectors in the subsequent dynamic-programming step.

Specifically, for every edge $e = (u, v) \in E$, we now keep track of just the set $T(u) = \{t \in T \mid (u, v, t) \in \mathcal{E}\}$ of all the timestamps where e exists, and store it in a $|T(e)|$-dimensional integer vector $\mathbf{w}_e$. Computing all such $\{\mathbf{w}_e\}_{e \in E}$ vectors takes $\mathcal{O}(\mathcal{E})$ time and $\mathcal{O}(\mathcal{E})$ space (as opposed to the $\mathcal{O}(|T| \times |E|)$ time and space required for the computation and storage of all the auxiliary $\{\mathbf{a}_{t_i}\}_{t_i \in T}$ vectors in the TGraphSeg-DP algorithm).

Afterwards, in the dynamic-programming step, for a given edge $e \in E$ and timestamp t_x, $\mathbf{a}_{t_x}[e]$ the number of timestamps within $[t_1, t_x]$ where e exists, is required. Note that $\mathbf{a}_{t_x}[e]$ can be computed by scanning the $\mathbf{w}_e$ vector as follows:

$$\mathbf{a}_{t_x}[e] = \operatorname{argmax}_{z=1,\dots,T(e)} t_x \geq \mathbf{w}_e[z]. \tag{10}$$

Example 2. Assume $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ and that an edge e exists in timestamps t_1, t_3, and t_4. This means that

$$\mathbf{w}_e = [t_1 \quad t_3 \quad t_4],$$

Algorithm 3 TGraphSeg-DP-SE.

Input: A temporal graph $\mathcal{G} = (V, T, \mathcal{E})$, an integer $k \in [1, |T|]$.
Output: A size-reduced temporal graph $\hat{\mathcal{G}}$ with k snapshots.

1: $\mathbf{P}, \mathbf{R} \leftarrow$ two empty $(|T| \times k)$-dimensional matrices ▷ *Penalty matrix and Reconstruction matrix*
▷ *Computing vectors* $\mathbf{w}_e$ *storing the timestamps at which* e *exists,* $\forall e \in E$
2: $\mathbf{w}_e \leftarrow [\,]$, for all $e \in E$
3: **for all** $e = (u, v, t) \in \mathcal{E}$ **do**
4: append t to $\mathbf{w}_e$
5: **end for**
▷ *Dynamic programming step*
6: **for all** $i = 1, \ldots, |T|$ **do**
▷ *Computing penalties for intervals* $\{[t_s, t_i]\}_{s=1,\ldots,i}$ *(Eq. (9))*
7: **for all** $s = i, i-1, \ldots, 1$ **do**
8: $p_{s,i} \leftarrow 0$
9: **for all** $t \in [t_s, t_i]$, $e \in E_t$ **do**
10: retrieve $\mathbf{a}_{t_s}[e]$, $\mathbf{a}_{t_i}[e]$ from $\mathbf{w}_e$ according to Eq. (10)
11: $\mathbf{a}[e] \leftarrow \mathbf{a}_{t_i}[e] - \mathbf{a}_{t_s}[e]$ if $s > 1$, $\mathbf{a} \leftarrow \mathbf{a}_{t_i}[e]$, otherwise ▷*#timest. in* $[t_s, t_i]$ *in which* e *exists*
12: $p_{s,i} \leftarrow p_{s,i} + \min\{\mathbf{a}[e], s - i + 1 - \mathbf{a}[e]\}$
13: **end for**
14: **end for**
▷ *Populating* $\mathbf{P}$, $\mathbf{R}$ *matrices*
15: $\mathbf{P}[i, 1] \leftarrow p_{1,i}$ ▷ *Optimal cost of partitioning* $[t_1, t_i]$ *into 1 segment, i.e., penalty of* $[t_1, t_i]$ *itself*
16: $\mathbf{R}[i, 1] \leftarrow 0$
17: **for all** $j = 2, \ldots, \min\{i, k\}$ **do**
18: $\mathbf{P}[i, j] \leftarrow \min_{s=1,\ldots,i-1} \mathbf{P}[s, j-1] + p_{s+1,i}$ ▷*Optimal cost of partitioning* $[t_{s+1}, t_i]$ *into* j *segm.*
19: $\mathbf{R}[i, j] \leftarrow \operatorname{argmin}_{s=1,\ldots,i-1} \mathbf{P}[s, j-1] + p_{s+1,i}$
20: **end for**
21: **end for**
▷ *Reconstructing the partition* $\{T_j\}_{j=1}^{k}$ *of* $T = [t_1, \ldots, t_{|T|}]$ *into* k *time intervals*
22: $ub \leftarrow |T|$
23: **for all** $j = k, \ldots, 1$ **do**
24: $lb \leftarrow \mathbf{R}[ub, j]$; $T_j \leftarrow [t_{lb}, t_{ub}]$; $ub \leftarrow lb - 1$
25: **end for**
▷ *Producing the output size-reduced temporal graph* $\hat{\mathcal{G}}$
26: $\mathbf{m}_j \leftarrow$ Hamming consensus vector of T_j, for all $j = 1, \ldots, k$ ▷ *Eq. (5)*
27: $\hat{T} \leftarrow \{1, \ldots, j\}$; $\hat{\mathcal{E}} \leftarrow \{(u, v, j) \mid \mathbf{m}_j[(u, v)] = 1\}$; $\hat{\mathcal{G}} \leftarrow (V, \hat{T}, \hat{\mathcal{E}})$

and the values $\{\mathbf{a}_{t_i}[e]\}_{t_i \in T}$ are as follows:

$$\mathbf{a}_{t_1}[e] = 1$$
$$\mathbf{a}_{t_2}[e] = 1$$
$$\mathbf{a}_{t_3}[e] = 2$$
$$\mathbf{a}_{t_4}[e] = 3$$
$$\mathbf{a}_{t_5}[e] = 3$$
$$\mathbf{a}_{t_6}[e] = 3$$

For $t_x = t_1$, the application of Eq. (10) returns $z = 1$, then $\mathbf{a}_{t_1}[e] = 1$. Similarly, for $t_x = t_2$, we have $z = 1$, then $\mathbf{a}_{t_2}[e] = 1$. For $t_x = t_5$, we have $z = 3$, then $\mathbf{a}_{t_5}[e] = 3$.

The space-efficient variant of the TGraphSeg-DP algorithm exploiting the above reasoning is termed TGraphSeg-DP-SE and is outlined in Algorithm 3. TGraphSeg-DP-SE differs from TGraphSeg-DP only in the way how penalties are (incrementally) computed (Lines 2–5 and Lines 7–14).

Time complexity. Lines 2–5 take $\mathcal{O}(|\mathcal{E}|)$ time. Executing the block at Lines 7–14 for all $i = 1, \ldots, |T|$ corresponds to processing every temporal edge $e \in \mathcal{E}$ a number $\mathcal{O}(|T|)$ of times. The most expensive step to be performed every time a temporal edge e is processed is the one at Line 10, which takes $\mathcal{O}(\log |T|)$ time (due to binary search over vector $\mathbf{w}_e$, whose dimensionality is upper-bounded by $|T|$).

Algorithm 4 TGraphSeg-Greedy.

Input: A temporal graph $\mathcal{G} = (V, T, \mathcal{E})$, an integer $k \in [1, |T|]$.
Output: A size-reduced temporal graph $\hat{\mathcal{G}}$ with k snapshots.

▷ *Computing auxiliary $|E|$-dimensional vectors $\{\mathbf{a}_{t_i}\}_{t_i \in T}$; $\mathbf{a}_{t_i}[e] :=$ number of timestamps within $[t_1, t_i]$ in which e exists, $\forall e \in E$ (Eq. (8))*

1: $\mathbf{a}_{t_i} \leftarrow \mathbf{0}^{|E|}$, for all $t_i \in T$
2: **for all** $i = 1, \ldots, |T|$, $e \in E_{t_i}$ **do**
3: $\quad \mathbf{a}_{t_i} \leftarrow \mathbf{a}_{t_i}[e] + 1$
4: **end for**

▷ *Initialization: single segment corresponding to the entire temporal sequence*

5: $\Delta_0 \leftarrow [1, |T|]$; $D \leftarrow \{\Delta_0\}$; $\mathcal{T} \leftarrow [\Delta_0]$; $p(\Delta_0) \leftarrow$ SEGMENT-PENALTY$(\mathbf{a}, \mathcal{E}, 1, |T|)$ ▷ *Algorithm 5*
6: **for all** $i = 2, \ldots, k+1$ **do**

▷ *Computing the best split for the segment(s) generated in the previous iteration*

7: $\quad$ max-$gain \leftarrow -\infty$
8: $\quad$ **for all** $\Delta = [t_s, t_e] \in D$ s.t. $|\Delta| > 1$ **do**
9: $\quad\quad E' \leftarrow \emptyset$; $E'' \leftarrow \{(u, v, t) \in \mathcal{E} \mid t \in [t_s, t_e]\}$
10: $\quad\quad$ **for all** $\hat{t} \in [t_s, t_e - 1]$ **do** ▷ *Trying all the possible splits of Δ*

▷ *Computing temporal edges falling within $[t_s, \hat{t}]$ (E') and $[\hat{t}+1, t_e]$ (E'')*

11: $\quad\quad\quad$ **for all** $e = (u, v, t) \in \mathcal{E}$ s.t. $t = \hat{t}$ **do**
12: $\quad\quad\quad\quad E' \leftarrow E' \cup \{e\}$
13: $\quad\quad\quad\quad$ **if** $\mathbf{a}_{\hat{t}}[e] = \mathbf{a}_{t_e}[e]$ **then** ▷ *It means that e is not present in time interval $[\hat{t}+1, t_e]$*
14: $\quad\quad\quad\quad\quad E'' \leftarrow E'' \setminus \{e\}$
15: $\quad\quad\quad\quad$ **end if**
16: $\quad\quad\quad$ **end for**

▷ *Checking whether the current split is the best one; if so, save it*

17: $\quad\quad\quad p_1 \leftarrow$ SEGMENT-PENALTY$(\mathbf{a}, E', t_s, \hat{t})$; $p_2 \leftarrow$ SEGMENT-PENALTY$(\mathbf{a}, E'', \hat{t}+1, t_e)$ ▷ *Algorithm 5*
18: $\quad\quad\quad$ **if** $p(\Delta) - p_1 - p_2 >$ max-$gain$ **then**
19: $\quad\quad\quad\quad$ max-$gain \leftarrow p(\Delta) - p_1 - p_2$; $\Delta^* \leftarrow \Delta$; $\Delta_1^* \leftarrow [t_s, \hat{t}]$; $\Delta_2^* \leftarrow [\hat{t}+1, t_e]$ $p_1^* \leftarrow p_1$; $p_2^* \leftarrow p_2$
20: $\quad\quad\quad$ **end if**
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: $\quad p(\Delta_1^*) \leftarrow p_1^*$; $p(\Delta_2^*) \leftarrow p_2^*$; $D \leftarrow \{\Delta_1^*, \Delta_2^*\}$; replace Δ^* in $\mathcal{T}$ with Δ_1^*, Δ_2^*
24: **end for**

▷ *Producing the output size-reduced temporal graph $\hat{\mathcal{G}}$*

25: $\mathbf{m}_j \leftarrow$ Hamming consensus vector of Δ_j, for all $\Delta_j \in \mathcal{T}$ ▷ *Eq. (5)*
26: $\hat{T} \leftarrow \{1, \ldots, j\}$; $\hat{\mathcal{E}} \leftarrow \{(u, v, j) \mid \mathbf{m}_j[(u, v)] = 1\}$; $\hat{\mathcal{G}} \leftarrow (V, \hat{T}, \hat{\mathcal{E}})$

Hence, executing Lines 7–14 for all $i = 1, \ldots, |T|$ takes overall $\mathcal{O}(|T|^2 + |T| \log |T| \times |\mathcal{E}|)$ time, and the ultimate time complexity of Algorithm 3 is $\mathcal{O}(|T|^2 \times k + |T| \log |T| \times |\mathcal{E}|)$.

Space complexity. As stated above, the auxiliary $\{\mathbf{w}_e\}_{e \in E}$ vectors take overall $\mathcal{O}(|\mathcal{E}|)$ space, which means that the overall space complexity of Algorithm 3 is $\mathcal{O}(|T| \times k + |\mathcal{E}|)$.

In conclusion, Algorithms 2 and 3 are similar to each other in terms of general design principles, and the main difference between the two is that the latter is a space-efficient variant of the former.

5. A faster heuristic algorithm for TGraphSeg

The algorithms described in the previous section are sound, but they take quadratic time in the number of snapshots of the input temporal graph. This may represent a limitation for applicability on temporal graphs with many timestamps. In this section, we propose an alternative algorithm for our TGraphSeg problem, which takes time linear in the number of snapshots of the input temporal graph. As a counterpart to having an increased efficiency, this alternative algorithm provides only heuristic solutions to the TGraphSeg problem, which means inferior accuracy compared to the exact algorithms. However, as demonstrated by our experiments (see Section 6), this drop in accuracy remains limited. This makes the proposed heuristic a competitive alternative to the exact algorithms.

Our faster heuristic is inspired by a *top-down greedy* method which has been popularly employed for the SeqSeg problem [11,12]. This greedy method has been widely used and is recognized as effective in several practical contexts where the SeqSeg problem finds application. The algorithm works as follows. It starts with an unsegmented sequence and introduces a new boundary at every greedy step. That is, in the i-th step, the algorithm introduces the i-th segment boundary by splitting one of the existing i segments into two ones. The new boundary is selected in such a way that it minimizes the overall SeqSeg penalty. No changes are made to the existing

Algorithm 5

```
1: function SEGMENT-PENALTY(a, E, t1, t2)
2:     s ← t2 − t1 + 1;  p ← 0
3:     for all e ∈ E do
4:         a1 ← 0, if t1 = 1; a1 ← a_{t1−1}[e], otherwise
5:         a_{t2}[e]
6:         p ← p + min{a2 − a1, s − (a2 − a1)}
7:     end for
8:     return p
9: end function
```

Table 1
Main characteristics of the datasets used in the experiments. $|V|$: #vertices; $|T|$: #timestamps; $|\mathcal{E}|$: #temporal edges; $|E|$: #non-temporal edges (i.e., #vertex pairs sharing at least one temporal edge).

Dataset	$\lvert V\rvert$	$\lvert T\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert E\rvert$
BrazilAir [33]	39 300	31	354 420	354 420
EUAir [33]	119 700	61	1 978 350	1 978 350
USAir [33]	348 110	101	4 487 670	4 487 670
Hospital [34,35]	75	9 453	32 424	1 139
HS [34,36]	180	11 273	45 047	2 239
AS [37]	6 054	47	1 048 575	17 364
aves-sparrow [38]	52	176	515	453
ia-workplace [38]	92	7 103	9826	754

$i-1$ boundary points once they have been set (this is the greedy functioning). The splitting is repeated until the desired number of output segments has been reached.

The heuristic we propose for our TGraphSeg method is an adaptation of such a greedy method for SeqSeg. The proposed algorithm is named TGraphSeg-Greedy, and outlined as Algorithm 4. Like Algorithms 2 and 3, the TGraphSeg-Greedy requires an efficient implementation in the temporal-graph setting. These challenges include efficiently checking all the valid splits (Lines 8–22) in every iteration and computing segment penalties incrementally (Lines 1–4, exploited in the function outlined as Algorithm 5, and Line 13).

Time complexity. Like Algorithms 2 and 3, Lines 1–4 take $\mathcal{O}(|\mathcal{E}|)$ time. Then, every iteration of the loop at Lines 10–21 (for computing the best split of a time interval Δ) requires: (*i*) a visit to all the temporal edges in Δ (Lines 11–16), which takes $\mathcal{O}(|\mathcal{E}_\Delta|)$ time, where $\mathcal{E}_\Delta$ denotes the set of all the temporal edges within Δ); and (*ii*) penalty computation for every split, which overall takes $\mathcal{O}(|\Delta| \times |E_\Delta|)$, where $\mathcal{O}(|\Delta|)$ is the number of splits, and $\mathcal{O}(|E_\Delta|)$ is the time for (efficient) penalty computation, based on Eq. (9) (with E_Δ denoting the set of all the non-temporal edges within Δ). Thus, every iteration of the loop at Lines 10–21 overall takes $\mathcal{O}(|\mathcal{E}_\Delta| + |\Delta| \times |E_\Delta|)$ time.

Every iteration $i = 2, \dots, k$ of the main loop (Lines 6–24) requires the computation of the best split of the (at most) two new segments generated in the previous iteration. In the worst case, the maximum size of the segment on which the best split has to be computed in the various k iterations is $|T|, |T|-1, \dots, |T|-k$. This means that, for every iteration i, the best split on a segment of at most size $O(|T|)$ has to be computed. This in turn means that, based on what was observed above for the cost of every iteration of the loop at Lines 10–21, the main loop overall takes $\mathcal{O}(k\ (|\mathcal{E}| + |T| \times |E|))$ time, which also corresponds to the overall time complexity of the algorithm.

Space complexity. The space taken by Algorithm 4 is determined by the space needed to keep the auxiliary $\mathbf{a}_{t_i}$ vectors, which is (as discussed in the previous section) $\mathcal{O}(|T| \times |E|)$.

To summarize, Algorithm 4 offers improved efficiency with respect to Algorithms 2 and 3, at the price of an expectedly reduced effectiveness. This is confirmed by our experiments (see Section 6).

6. Experiments

In this section, we empirically evaluate the efficiency and effectiveness of the proposed algorithms for the TGraphSeg problem and compare them to a time-series-domain-inspired baseline.

In particular, we first describe the experimental setting, including datasets, methods, assessment criteria, implementation details, and testing environment. Then, in Sections 6.1 and 6.2, we provide the experimental results.

Datasets. We experiment with eight publicly available real datasets, which have been widely employed in temporal-graph management [2,39]. These datasets are described below, and their main characteristics are shown in Table 1.

BrazilAir, EUAir, USAir [33] are temporal graphs depicting air traffic. In these datasets, vertices and temporal edges represent airports and air traffic among airports occurring at specific timestamps. Hospital [34,35] contains (RFIDs) contacts in a hospital ward

in Lyon, France, during Dec 6th-10th, 2010, in 20-second intervals. HighSchool (HS) [34,36] contains contacts in 5 classes in a high school in Marseilles, France during 7 days in Nov 2012, in 20-second intervals. AutonomousSystems (AS) [37] is a communication network from Border Gateway Protocol logs, spanning from Nov 8, 1997 to Jan 2, 2000. aves-sparrow [38] contains real-world animal interaction data from published studies of wild, captive, and domesticated animals. ia-workplace [38] represents the contacts between colleagues in a workplace.

Methods. We test the performance of the proposed algorithms, both the exact TGraphSeg-DP (Algorithm 2) and the heuristic TGraphSeg-Greedy (Algorithm 4). We also involve two baselines inspired by popular methods for segmenting time-series data, namely *piecewise aggregate approximation* (PAA) [40] and adaptive piecewise constant approximation (APCA) [41]. The PAA-inspired baseline, termed PAA-baseline, segments a given temporal graph with $|T|$ snapshots into a number k of constant-size segments, i.e., in such a way that the first segment contains the first $\lfloor |T|/k \rfloor$ snapshots, the second segment contains the second $\lfloor |T|/k \rfloor$ snapshots, and so on. Every such a constant-size segment is ultimately replaced with the corresponding Hamming consensus vector (like the proposed TGraphSeg-DP and TGraphSeg-Greedy algorithms). The APCA-inspired baseline, termed APCA-baseline, segments a given temporal graph into a not a-priori fixed number of segments which are determined by comparing any temporal snapshot to the previous one. Specifically, a new segment is identified whenever the Hamming distance between any two consecutive snapshots exceeds a certain threshold η. In our experiments, we adopt a well-established choice for η and set it to one and a half times the product of the mean and the standard deviation of the (Hamming) distance between any two consecutive snapshots of the input temporal graph [41].

Assessment. We provide two types of performance assessments. First, a quantitative one, where running time and penalty (according to Eq. (6) of segmentation are reported (Section 6.1). The penalty function metric used in our experiments quantifies the fidelity of the reduced temporal graph to the original graph.

Then, we test the various segmentation methods in a couple of popular yet fundamental downstream tasks in temporal-graph management, namely *vertex similarity search* and *temporal community detection*. These tasks, along with the corresponding assessment criteria and results are described in detail in Section 6.2.

Implementation. All the algorithms and experiments were implemented using `Python` 3.11.0 as a programming language. The following `Python` packages were used: `pandas`, `NumPy`, `tnetwork`.

All our source code will be made public upon acceptance of the paper.

Testing environment. All the experiments were run on a PC equipped with an Intel(R) Core(TM) i7-10750 H CPU running at a base frequency of 2.60 GHz (with a maximum turbo frequency of 2.59 GHz), 16GB RAM, and 64-bit Windows 11 Pro version 23H2 operating system.

6.1. Results – quantitative

Table 2 reports on the quantitative results of temporal-graph segmentation on all the selected datasets for all the proposed methods and PAA-baseline, and with varying the number k of output segments as the 10 %, 25 %, 50 %, 75 % of the total number of original snapshots. The proposed TGraphSeg-DP and TGraphSeg-Greedy algorithms are far more accurate than the baseline. As expected the exact TGraphSeg-DP generally outperforms the TGraphSeg-Greedy heuristic, but it is slower. In particular, TGraphSeg-DP's time complexity which is quadratic in the number of timestamps prevented it from terminating in a reasonable time (1 day) on the HS dataset and on three out of four k's on the Hospital dataset. The drop in accuracy of TGraphSeg-Greedy with respect to TGraphSeg-DP remains however limited in all the cases. This makes TGraphSeg-Greedy a competitive alternative to TGraphSeg-DP, perhaps to be soundly used when the runtime of the latter is too high. As a general (expected) remark, accuracy (resp. runtime) gets better (resp. worse) as the number of k of output segments increases. This is apparent with the proposed TGraphSeg-DP and TGraphSeg-Greedy, and much less evident with PAA-baseline. This further attests the suitability of the proposed algorithms and the unsuitability of the baseline for segmenting a temporal graph: in the proposed algorithms, the number of segments can be used as an effective knob to tradeoff between accuracy and efficiency, whereas such a knob is much less effective in the baseline.

Table 3 reports the results of APCA-baseline. The superior accuracy of the proposed TGraphSeg-DP and TGraphSeg-Greedy algorithms is confirmed over such a baseline too.

Fig. 4 shows a sensitivity analysis on the number k of output segments. In this analysis, we focus on the results achieved by the proposed TGraphSeg-DP algorithm on the EUAir dataset. The analysis reveals that, as expected, the penalty decreases as the number of segments increases, and such a decrease follows a trend that is roughly linear.

6.2. Results – downstream tasks

We also experiment with downstream tasks to be performed on both the original temporal graphs and their size-reduced versions. The rationale is to test how much efficiency gain and accuracy drop are achieved when executing a downstream task on the reduced temporal graph instead of having it run on the original graph. The two downstream tasks we consider are vertex similarity search and temporal community detection. However, any graph-processing downstream task can benefit from our temporal-graph-segmentation techniques. Testing our methods on further downstream tasks is deferred to future work.

In the following, we provide the details of the experiments performed on each of the selected tasks.

Table 2
Quantitative results of TGraphSeg problem. Proposed TGraphSeg-DP (exact) and TGraphSeg-Greedy (heuristic) algorithms vs. PAA-baseline. k: number of output segments. Accuracy evaluated in terms of *penalty* (Eq. (6); the lower, the better).

Dataset	k	TGraphSeg-DP		TGraphSeg-Greedy		PAA-baseline	
		runtime (h:mm:ss.ms)	*penalty*	*runtime* (h:mm:ss.ms)	*penalty*	*runtime* (h:mm:ss.ms)	*penalty*
BrazilAir	3	0:00:23.536	331 556	0:00:07.006	331 862	0:00:02.081	354 420
	8	0:00:23.575	272 264	0:00:16.785	273 278	0:00:01.988	354 419
	15	0:00:23.596	191 926	0:00:25.602	193 719	0:00:02.042	343 103
	23	0:00:23.889	99 828	0:00:29.590	99 828	0:00:02.096	343 101
EUAir	6	0:08:59.427	1 815 161	0:02:21.141	1 820 676	0:00:15.316	1 978 350
	14	0:08:46.697	1 548 288	0:05:17.515	1 557 207	0:00:15.699	1 946 863
	30	0:09:29.657	1 028 261	0:09:04.169	1 040 108	0:00:15.642	1 946 861
	45	0:08:34.325	543 630	0:09:53.201	558 558	0:00:15.088	1 946 861
USAir	10	0:54:03.533	4 082 351	0:14:59.472	4 082 351	0:00:41.915	4 487 138
	25	0:55:47.618	3 410 269	0:33:35.201	3 411 213	0:00:43.404	4 445 313
	50	0:52:54.237	2 294 656	0:43:28.201	2 301 686	0:00:38.511	4 486 065
	75	0:56:17.928	1 186 502	0:53:32.318	1 190 131	0:00:40.762	4 444 289
Hospital	945	13:35:06.560	18 149	0:00:54.654	26 521	0:00:00.210	23 705
	2 363	—	—	0:03:26.206	19 203	0:00:00.215	19 000
	4 726	—	—	0:04:04.303	10 255	0:00:00.212	14 761
	7 089	—	—	0:04:28.378	1 354	0:00:00.213	13 649
HS	1 127	—	—	0:02:55.914	34 559	0:00:00.361	31 437
	2 818	—	—	0:05:29.913	29 411	0:00:00.408	25 852
	5 637	—	—	0:14:29.654	9 510	0:00:00.487	19 582
	8 455	—	—	0:13:57.765	2 035	0:00:00.513	19 582
AS	4	0:00:16.079	30 181	0:00:04.162	35 031	0:00:03.181	36 962
	12	0:00:15.280	9 018	0:00:04.769	13 614	0:00:02.930	29 714
	23	0:00:15.044	4 817	0:00:05.161	9 784	0:00:03.089	27 727
	35	0:00:15.259	1 796	0:00:05.608	6 708	0:00:03.152	22 459
aves-sparrow	17	0:00:00.924	411	0:00:00.328	489	0:00:00.004	515
	44	0:00:00.925	286	0:00:00.733	403	0:00:00.005	515
	88	0:00:01.036	143	0:00:01.155	216	0:00:00.006	513
	132	0:00:01.013	56	0:00:01.141	78	0:00:00.005	513
ia-workplace	710	4:19:55.247	4 387	0:02:16.327	7 260	0:00:00.075	6 663
	1 776	5:44:13.489	2 194	0:04:48.694	4 117	0:00:00.102	5 127
	3 551	9:51:12.271	249	0:04:57.657	1 538	0:00:00.096	4 165
	5 327	11:34:16.547	0	0:04:55.879	725	0:00:00.094	3 825

Table 3
Quantitative results of TGraphSeg problem: APCA-baseline. k: number of output segments. η: threshold concerning the difference between consecutive snapshots (used to decide the splitting into segments). Accuracy is evaluated in terms of *penalty* (Eq. (6); the lower, the better).

Dataset	η	k	*Penalty*	*Runtime* (h:mm:ss.ms)
BrazilAir	24 977.27	2	354 419	0:00:04.328
EUAir	68 109.56	5	1 942 886	0:00:27.695
USAir	93 238.76	8	4 391 153	0:01:14.180
Hospital	6.64	808	23 189	0:00:04.380
HS	8.98	812	31 040	0:00:00.683
AS	4 907.88	5	34 585	0:00:04.776
aves-sparrow	11.99	18	424	0:00:00.142
ia-workplace	2.82	716	6 561	0:00:00.140

Vertex similarity search. We compute vertex (temporal) embeddings – i.e., numerical-vector representations of every vertex in a given temporal graph [6] – by employing the algorithm devised in [33]. Then, for every vertex, we compute the top-h most similar vertices according to their embeddings (we use $h = 25$). We evaluate accuracy by measuring precision and recall of the top-h most similar neighbours of a vertex computed on the reduced temporal graph with respect to the (ground-truth) top-h most similar neighbours computed on the original graph and averaging such precision and recall values over all the vertices. As far as efficiency,

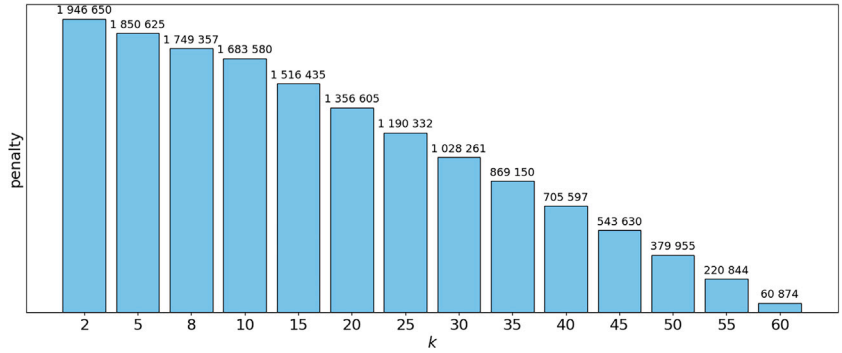


Fig. 4. Sensitivity analysis in terms of penalty on the number k of output segments (TGraphSeg-DP method, EUAir dataset).

Table 4
Accuracy results of vertex similarity search downstream task. Proposed TGraphSeg-DP (exact) and TGraphSeg-Greedy (heuristic) algorithms vs. PAA-baseline. k: number of output segments (i.e., number of snapshots of the reduced temporal graphs). Accuracy evaluated in terms of precision and recall (the higher, the better) of the top-h ($h = 25$) most similar neighbours of a vertex computed on the reduced temporal graph concerning the (ground-truth) top-h most similar neighbours computed on the original graph, and averaging such precision and recall values over all the vertices. The best results are highlighted in bold.

Dataset	k	TGraphSeg-DP		TGraphSeg-Greedy		PAA-baseline	
		Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
BrazilAir	8	**0.0042**	**0.0041**	0.0039	0.0039	0	0
EUAir	14	0.0058	0.0057	**0.0068**	**0.0057**	0.0005	0.0005
AS	12	0.1011	0.1007	**0.1035**	**0.1031**	0.0016	0.0016
aves-sparrow	44	0.7054	0.4843	**0.7054**	**0.6354**	0.0588	0.0588
ia-workplace	3 551	**0.8101**	**0.7719**	0.8007	0.7663	0.8647	0.7643

Table 5
Efficiency results of vertex similarity search downstream task. The table shows the runtimes (h:mm:ss.ms) of embedding the original input temporal graph ("*original graph*" column), and embedding the temporal graphs reduced according to the proposed TGraphSeg-DP (exact) and TGraphSeg-Greedy (heuristic) algorithms, as well as according to PAA-baseline. k: number of output segments (i.e., number of snapshots of the reduced temporal graphs).

Dataset	k	*Original graph*	TGraphSeg-DP	TGraphSeg-Greedy	PAA-baseline
BrazilAir	8	0:00:45.869	0:00:43.067	0:00:39.883	0:00:00.136
EUAir	14	0:02:42.069	0:02:29.717	0:02:25.759	0:00:45.770
AS	12	0:00:09.155	0:00:07.706	0:00:07.532	0:00:00.870
aves-sparrow	44	0:00:00.214	0:00:00.231	0:00:00.261	0:00:00.141
ia-workplace	3 551	0:00:00.644	0:00:00.448	0:00:00.433	0:00:00.519

instead, we compare the running time of computing all the vertex embeddings on the reduced temporal graph to the running time of computing all the vertex embeddings on the original temporal graph.

The results of this evaluation are shown in Tables 4 (accuracy) and 5 (efficiency). The tables show intuitive yet expected results. In fact, in terms of accuracy, the exact proposed TGraphSeg-DP algorithm is in general the best-performing method, followed (quite closely) by the other proposed algorithm, i.e., the heuristic TGraphSeg-Greedy. Both our algorithms are superior to the baseline. Regarding efficiency, all the temporal-graph-segmentation algorithms exhibit a reduction in runtime when embedding the entire input original graph. TGraphSeg-DP and TGraphSeg-Greedy get comparable running times: this confirms that they can both perform a reduction that is effective for the downstream task of temporal-graph embedding. The baseline is often faster than the proposed algorithms. At first glance, this seems a pro of the baseline. However, a more careful inspection reveals that whenever PAA-baseline is fast, the segmentation produced by it leads to embedding vectors that are poorly informative (i.e., with most components equal to zero). This is confirmed by the poor accuracy results reported in Table 4.

Temporal community detection. As a second downstream task, we employ temporal community detection. The idea is to produce a community structure of a temporal graph and measure the quality of such a community structure in terms of the well-established

Table 6
Accuracy results of temporal community detection downstream task. Proposed TGraphSeg-DP (exact) and TGraphSeg-Greedy (heuristic) algorithms vs. PAA-baseline. k: number of output segments (i.e., number of snapshots of the reduced temporal graphs). Accuracy is evaluated in terms of modularity of the produced community structures (according to the Smoothed Louvain method [42]). The best results are highlighted in bold.

Dataset	k	*Original graph*	TGraphSeg-DP	TGraphSeg-Greedy	PAA-baseline
BrazilAir	8	0.99876	0.87424	**0.87435**	0.0
USAir	25	0.98963	0.95934	**0.95941**	0.95941
EUAir	14	0.98182	**0.92737**	0.92726	0.29999
AS	12	0.63521	0.57572	**0.57682**	0.28125
aves-sparrow	44	0.27105	0.21168	**0.54136**	0.0
ia-workplace	3551	0.12403	**0.16484**	0.13892	0.09692

Table 7
Efficiency results of temporal community detection downstream task. The table shows the runtimes (h:mm:ss.ms) for performing temporal community detection (according to the Smoothed Louvain method [42]) on the original input temporal graph ("*original graph*" column), and on the temporal graphs reduced according to the proposed TGraphSeg-DP (exact) and TGraphSeg-Greedy (heuristic) algorithms, as well as according to PAA-baseline. k: number of output segments (i.e., number of snapshots of the reduced temporal graphs).

Dataset	k	*Original graph*	TGraphSeg-DP	TGraphSeg-Greedy	PAA-baseline
BrazilAir	8	03:44.3	00:53.1	01:01.8	00:00.2
USAir	25	72:10.2	56:02.1	57:11.5	62:30.5
EUAir	14	40:16.1	09:29.7	09:02.3	00:03.1
AS	12	00:12.4	00:02.4	00:04.2	00:00.2
aves-sparrow	44	00:00.1	00:00.1	00:00.1	00:00.1
ia-workplace	3551	00:03.5	00:01.6	00:01.6	00:01.0

modularity criterion [43,44]. As a temporal community detection algorithm, we use a temporal version of the popular *Louvain* algorithm [45], termed *Smoothed Louvain* [42].[1] Accuracy evaluation is accomplished by comparing the modularity achieved on the reduced temporal graph to that achieved on the original. Efficiency is assessed by measuring the runtimes of producing the temporal community structure on the reduced temporal graph vs. the original one.

The results of this evaluation are reported in Tables 6 (accuracy) and 7 (efficiency). Such results are very similar to those observed for the previous downstream task (vertex similarity search). The proposed TGraphSeg-DP and TGraphSeg-Greedy algorithms achieve the highest modularity values, much higher than the baseline. In terms of efficiency, TGraphSeg-DP and TGraphSeg-Greedy get comparable runtimes, achieving a consistent gain over running temporal community detection on the original graph. Like vertex similarity search, the baseline is often faster than the proposed algorithms. This is again motivated by a poorly effective segmentation produced by the baseline, which leads to many (almost) empty snapshots, thus resulting in rather low modularity values, as shown in Table 6.

7. Discussion

Using segmented temporal graphs offers several benefits, including faster processing, reduced memory storage requirements, and improved scalability for handling large datasets. It enables faster execution of downstream tasks (e.g., analysis and querying) while promoting energy efficiency and efficient use of computing resources. Additionally, segmentation helps filter out redundant data, enhancing the quality of insights and reducing noise, all without significant loss of important information. More specifically, we believe that segmentation of temporal graphs has a wide range of real-world applications across various fields where the underlying data of interest are constituted by time-varying relationships among entities. (i) In social media networks, such algorithms can be used to detect emerging trends, evolving friendships, or influence patterns during major events like elections or viral phenomena. (ii) In transportation networks, temporal graph segmentation can identify stable traffic patterns and detect changes such as congestion spikes, route closures, or disruptions. (iii) In epidemic modeling, this method can help identify phases in the spread of diseases, aiding in the assessment of interventions like lockdowns or vaccinations and their impact on transmission. (iv) In systems biology, temporal graphs can model gene–protein interaction networks, with segmentation helping to identify distinct phases of biological processes and offering insights into how interactions evolve under different conditions or stages of development. (v) In telecommunication networks, temporal graph segmentation can reveal patterns of network usage, identifying peak periods and moments of low demand for better resource management.

The overall time complexity of Algorithm TGraphSeg-DP is $O(|T|^2 \times k + |T| \times |E|)$, which highlights key scalability considerations. The first term, $|T|^2 \times k$, indicates that the algorithm experiences quadratic growth with respect to the number of timestamps ($|T|$). As the number of timestamps increases, the computational cost grows rapidly, making the algorithm less efficient for larger temporal

[1] We use the implementation available from the `tnetwork` `Python` package (https://tnetwork.readthedocs.io/).

graphs. This quadratic dependency can become a significant bottleneck as the graph's time span or temporal resolution increases, particularly if the number k of segments is large. The second term, $|T| \times |E|$, introduces a linear dependence on both number of timestamps and number of edges ($|E|$). As the graph grows in size and edge density, the algorithm's performance is influenced by the increased need to process more connections between time steps. In practical scenarios, where graphs can be both temporally extensive and densely connected, the algorithm's runtime will scale accordingly, leading to slower performance. Thus, the proposed algorithm may perform well on smaller or sparser graphs, while its efficiency may degrade as the graph size and complexity increase.

8. Conclusion and future directions

In this work, we introduce the problem of *segmentation of temporal graphs* (TGraphSeg). TGraphSeg problem is formulated as an instance of the well-established *sequence segmentation* (SeqSeg), a combinatorial-optimization problem popularly employed for segmenting sequences of numerical (time-series) data. We provide a variety of contributions: (i) we address the technical challenges of contextualizing SeqSeg to the temporal-graph setting, (ii) we devise algorithms (both an exact one, and a faster heuristic one) by achieving the implementation challenges of adapting classic algorithms for SeqSeg to our context, (iii) we conduct extensive (quantitative) experiments to evaluate the performance of the proposed algorithms and compare them to a baseline, and (iv) we showcase our proposal in two prominent downstream tasks, namely vertex similarity search and temporal community detection.

Due to its novelty and highly general applicability, we believe our work is pioneering in the field of temporal-graph data management. As such, it can be extended and developed further. Promising directions for future work include the following ones. First of all, further algorithms could be devised, by, e.g., contextualizing the existing approximation algorithms for SeqSeg to the temporal-graph setting. Also, our (or new) algorithms could be improved from an efficiency point of view by, e.g., exploiting local monotonicities (like the one used for concave penalty functions in SeqSeg), or pre-aggregating identical consecutive snapshots. Further turnarounds for the efficiency bottleneck might be to adapt the existing algorithms (or design brand new algorithms) so as to have them capable of being run in "non-conventional" computing settings, such as, e.g., streaming, distributed, or parallel settings. Another direction could be to eliminate the dependency of our problem definition on the user-defined parameter corresponding to the number of output segments. This could lead to alternative problem formulations where the number of segments is automatically determined according to some dynamic stopping rule, which identifies segments whenever the current snapshot differs enough from the ones in the current segment. A further idea consists of using different distances than the Hamming one. This is a very challenging and stimulating direction, as changing the distance function implies carrying out dedicated research work by itself because this change would lead to a change in the form of the snapshot representative and to the necessity of defining brand new algorithmic techniques. Similar challenges also arise in the extension of our proposal to information-richer graphs, such as directed, edge-weighted, or attributed temporal graphs. Such an extension constitutes a further interesting and potentially impactful direction for future research.

Finally, let us mention that, like any data summarization technique, our proposal may suffer from risks and limitations in particularly sensitive domains (e.g., medical or financial), where data summarization may result in a critical reduction of interpretability or trustworthiness in general during the downstream tasks. Possible mitigation strategies for this could include retaining mappings from the summarized versions back to the raw data and properly using them to provide interpretations of the outcomes of the downstream tasks in terms of the original data.

CRediT authorship contribution statement

Raffaele Giancotti: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Francesco Gullo:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Pietro H. Guzzi:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Edoardo Serra:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Pierangelo Veltri:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

PVe was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan (PNRR) funded by the European Union: NextGenerationEU. PVe is also supported by GOLD Project and Nephrology Project both under the MUR National PNRR. PVe is also supported by AI-ging Project under the MUR National PNRR (Rome Technopole).

PhG is partially funded by the Next Generation EU - Italian NRRP, Mission 4, Component 2, Investment 1.5, call for the creation and strengthening of "Innovation Ecosystems", building "Territorial R&D Leaders" (Directorial Decree n. 2021/3277) – project Tech4You – Technologies for climate change adaptation and quality of life improvement, n. ECS0000009. This work reflects only the authors' views and opinions, neither the Ministry for University and Research nor the European Commission can be considered responsible for them.

This research was also possible thanks to the NSF Award #2435964.

Data availability

All the data are publicly available. Source code will be made public upon acceptance of the manuscript.

References

[1] C.C. Aggarwal, H. Wang, Managing and Mining Graph Data, Vol. 40 of Advances in Database Systems, Springer, 2010.
[2] A. Gionis, L. Oettershagen, I. Sarpe, Mining temporal networks, in: Companion Proc. of the ACM on Web Conference (WWW), 2024, pp. 1260–1263.
[3] J. Byun, S. Woo, D. Kim, ChronoGraph: enabling temporal graph traversals for efficient information diffusion analysis over time, in: Proc. of IEEE Int. Conf. on Data Engineering (ICDE), 2020, pp. 2026–2027.
[4] E. Galimberti, M. Ciaperoni, A. Barrat, F. Bonchi, C. Cattuto, F. Gullo, Span-core decomposition for temporal networks: algorithms and applications, ACM Trans. Knowl. Discov. Data 15 (1) (2021) 2:1–2:44.
[5] S. Gandhi, Y. Simmhan, An interval-centric model for distributed computing over temporal graphs, in: Proc. of IEEE Int. Conf. on Data Engineering (ICDE), 2020, pp. 1129–1140.
[6] S.M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, P. Poupart, Representation learning for dynamic graphs: a survey, J. Mach. Learn. Res. 21 (2020) 70:1–70:73.
[7] Y. Sun, S. Ma, B. Cui, Hunting temporal bumps in graphs with dynamic vertex properties, in: Proc. of ACM Int. Conf. on Management of Data (SIGMOD), 2022, pp. 874–888.
[8] R. Bellman, On the approximation of curves by line segments using dynamic programming, Commun. ACM 4 (6) (1961) 284.
[9] A. Amir, G.M. Landau, J.C. Na, H. Park, K. Park, J.S. Sim, Efficient algorithms for consensus string problems minimizing both distance sum and radius, Theor. Comput. Sci. 412 (39) (2011) 5239–5246.
[10] J. Chen, D. Hermelin, M. Sorge, On computing centroids according to the p-norms of hamming distance vectors, in: Proc. of Europ. Symp. on Algorithms (ESA), 2019, pp. 28:1–28:16.
[11] E. Bingham, Finding segmentations of sequences, In S. Dzeroski, B. Goethals, P. Panov (Eds.), Inductive Databases and Constraint-Based Data Mining, Springer, 2010, pp. 177–197.
[12] H. Shatkay, S.B. Zdonik, Approximate queries and representations for large data sequences, in: S.Y.W. Su (Ed.), Proc. of IEEE Int. Conf. on Data Engineering (ICDE), 1996, pp. 536–545.
[13] H. Zhong, G. Loukides, A. Conte, S.P. Pissis, Ego-network segmentation via (weighted) Jaccard median, IEEE Trans. Knowl. Data Eng. (2024) 1–16.
[14] Y. Liu, T. Safavi, A. Dighe, D. Koutra, Graph summarization methods and applications: a survey, ACM Comput. Surv. 51 (3) (2018) 62:1–62:34.
[15] X. Gou, L. Zou, C. Zhao, T. Yang, Fast and accurate graph stream summarization, in: Proc. of IEEE Int. Conf. on Data Engineering (ICDE), 2019, pp. 1118–1129.
[16] Z. Jiang, H. Chen, H. Jin, Auxo: a scalable and efficient graph stream summarization structure, Proc. of the VLDB Endow. (PVLDB) 16 (6) (2023).
[17] S. Kapoor, D.K. Saxena, M. van Leeuwen, Online summarization of dynamic graphs using subjective interestingness for sequential data, Data Min. Knowl. Discov. 35 (1) (2021) 88–126.
[18] I. Tsalouchidou, F. Bonchi, G.D.F. Morales, R. Baeza-Yates, Scalable dynamic graph summarization, IEEE Trans. Knowl. Data Eng. 32 (2) (2020) 360–373.
[19] M. Besta, T. Hoefler, Survey and taxonomy of lossless graph compression and space-efficient graph representations, arXiv preprint arXiv:1806.01799, 2018.
[20] M. Coscia, F.M.H. Neffke, Network backboning with noisy data, in: Proc. of IEEE Int. Conf. on Data Engineering (ICDE), 2017, pp. 425–436.
[21] P.B. Slater, A two-stage algorithm for extracting the multiscale backbone of complex weighted networks, Proc. Natl. Acad. Sci. 106 (26) (2009) E66.
[22] W.S. Fung, R. Hariharan, N.J.A. Harvey, D. Panigrahi, A general framework for graph sparsification, SIAM J. Comput. 48 (4) (2019) 1196–1223.
[23] K.J. Ahn, S. Guha, A. McGregor, Graph sketches: sparsification, spanners, and subgraphs, in: Proc. of Symp. on Principles of Database Systems (PODS), 2012, pp. 5–14.
[24] D.A. Spielman, S. Teng, Spectral sparsification of graphs, SIAM J. Comput. 40 (4) (2011) 981–1025.
[25] Y. Zeng, C. Song, T. Ge, Selective edge shedding in large graphs under resource constraints, in: Proc. of IEEE Int. Conf. on Data Engineering (ICDE), 2021, pp. 2057–2062.
[26] P. Rozenshtein, F. Bonchi, A. Gionis, M. Sozio, N. Tatti, Finding events in temporal networks: segmentation meets densest subgraph discovery, Knowl. Inf. Syst. 62 (4) (2020) 1611–1639.
[27] C.W. Arachchi, N. Tatti, Recurrent segmentation meets block models in temporal networks, in: Proc. of Int. Conf. on Discovery Science (DS), 2022, pp. 445–459.
[28] P. Seiron, A. Lindegren, M. Magnani, C. Rohner, T. Murata, P. Holme, Modularity-based selection of the number of slices in temporal network clustering, in: Temporal Network Theory, Springer, 2023, pp. 435–447.
[29] P. Rozenshtein, N. Tatti, A. Gionis, The network-untangling problem: from interactions to activity timelines, Data Min. Knowl. Discov. 35 (1) (2021) 213–247.
[30] S.E. Amiri, L. Chen, B.A. Prakash, SnapNETS: automatic segmentation of network sequences with node labels, in: Proc. of AAAI Conf. on Artificial Intelligence (AAAI), 2017, pp. 3–9.
[31] F. Chierichetti, R. Kumar, S. Pandey, S. Vassilvitskii, Finding the Jaccard median, in: Proc. ACM-SIAM SODA Conf, 2010, pp. 293–311.
[32] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Anal. Appl. 13 (1) (2010) 113–129.
[33] J. Layne, J. Carpenter, E. Serra, F. Gullo, Temporal SIR-GN: efficient and effective structural representation learning for temporal graphs, Proc. of the VLDB Endowment (PVLDB) 16 (9) (2023) 2075–2089.
[34] A.A. Barrat, C. Cattuto, J.-F. Pinton, W. Van den Broeck, SocioPatterns, 2008.
[35] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Régis, B.-A. Kim, B. Comte, N. Voirin, Estimating potential infection transmission routes in hospital wards using wearable proximity sensors, PLOS ONE 8 (9) (2013) 1–9.
[36] J. Fournet, A. Barrat, Contact patterns among high school students, PLOS ONE 9 (9) (2014) 1–17.
[37] J. Leskovec, A. Krevl, SNAP datasets: stanford large network dataset collection, Jun. 2014. http://snap.stanford.edu/data.
[38] R.A. Rossi, N.K. Ahmed, The network data repository with interactive graph analytics and visualization, in: Proc. of AAAI Conf. on Artificial Intelligence (AAAI), 2015, pp. 4292–4293, https://networkrepository.com.
[39] P. Rozenshtein, A. Gionis, Mining temporal networks, in: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), 2019, pp. 3225–3226.
[40] E.J. Keogh, M.J. Pazzani, Scaling up dynamic time warping for datamining applications, in: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), 2000, pp. 285–289.
[41] K. Chakrabarti, E.J. Keogh, S. Mehrotra, M.J. Pazzani, Locally adaptive dimensionality reduction for indexing large time series databases, ACM Trans. Database Syst. 27 (2) (2002) 188–228.
[42] T. Aynaud, J. Guillaume, Static community detection algorithms for evolving networks, in: Proc. of Int. Symp. on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt), 2010, pp. 513–519.
[43] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[44] M.E.J. Newman, Modularity and community structure in networks, Proc. Natl. Acad. Sci. 103 (23) (2006) 8577–8582.
[45] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. 2008 (10) (2008) P10008.