Machine Learning (2024) 113:5623–5653
https://doi.org/10.1007/s10994-023-06507-6

Recurrent segmentation meets block models in temporal networks

Chamalee Wickrama Arachchi[1] · Nikolaj Tatti[1]

Received: 3 March 2023 / Revised: 7 August 2023 / Accepted: 16 December 2023 /
Published online: 30 January 2024

Abstract

A popular approach to model interactions is to represent them as a network with nodes being the agents and the interactions being the edges. Interactions are often timestamped, which leads to having timestamped edges. Many real-world temporal networks have a recurrent or possibly cyclic behaviour. In this paper, our main interest is to model recurrent activity in such temporal networks. As a starting point we use stochastic block model, a popular choice for modelling static networks, where nodes are split into R groups. We extend the block model to temporal networks by modelling the edges with a Poisson process. We make the parameters of the process dependent on time by segmenting the time line into K segments. We require that only $H \leq K$ different set of parameters can be used. If $H < K$, then several, not necessarily consecutive, segments must share their parameters, modelling repeating behaviour. We propose two variants where a group membership of a node is fixed over the course of entire time line and group memberships are allowed to vary from segment to segment. We prove that searching for optimal groups and segmentation in both variants is **NP**-hard. Consequently, we split the problem into 3 subproblems where we optimize groups, model parameters, and segmentation in turn while keeping the remaining structures fixed. We propose an iterative algorithm that requires $\mathcal{O}\left(KHm + Rn + R^2 H\right)$ time per iteration, where n and m are the number of nodes and edges in the network. We demonstrate experimentally that the number of required iterations is typically low, the algorithm is able to discover the ground truth from synthetic datasets, and show that certain real-world networks exhibit recurrent behaviour as the likelihood does not deteriorate when H is lowered.

Keywords Temporal networks · Block models · Segmentation · Recurrent

Editors: Dino Ienco, Roberto Interdonato, Pascal Poncelet.

✉ Chamalee Wickrama Arachchi
chamalee.wickramaarachch@helsinki.fi

Nikolaj Tatti
nikolaj.tatti@helsinki.fi

[1] HIIT, University of Helsinki, Helsinki, Finland

1 Introduction

A popular approach to model interactions between set of agents is to represent them as a network with nodes being the agents and the interactions being the edges. Naturally, many interactions in real-world datasets have a timestamp, in which case the edges in networks also have timestamps. Consequently, developing methodology for temporal networks has gained attention in data mining literature (Holme & Saramäki, 2012).

Many temporal phenomena have recurrent or possibly cyclic behaviour. For example, social network activity may be heightened during certain hours of day. Our main interest is to model recurrent activity in temporal networks. As a starting point we use stochastic block model, a popular choice for modelling static networks. We can immediately extend this model to temporal networks, for example, by modelling the edges with a Poisson process. Furthermore, Corneli et al. (2018) modelled the network by also segmenting the timeline and modelled each segment with a separate Poisson process.

To model the recurrent activity we can either model it explicitly, for example, by modelling explicitly cyclic activity, or we can use more flexible approach where we look for segmentation but restrict the number of distinct parameters. Such notion was proposed by Gionis and Mannila (2003) in the context of segmenting sequences of real valued vectors.

In this paper we extend the model proposed by Corneli et al. (2018) using the ideas proposed by Gionis and Mannila (2003). More formally, we consider the following problem: given a temporal graph with n nodes and m edges, we are looking to partition the nodes into R groups and segment the timeline into K segments that are grouped into H levels. Note that a single level may contain non-consecutive segments. An edge $e = (u, v)$ is then modelled with a Poisson process with a parameter λ_{ijh}, where i and j are the groups of u and v, and h is the level of the segment containing e.

This paper is an extension of a conference paper (Arachchi & Tatti, 2022). The model considered in the conference version did not allow node groups to vary over time. We extend the approach in this paper by also considering a variant where the node groups depend on the level.

To obtain good solutions for both of the variants, we rely on an iterative method by splitting the problem into three subproblems: (i) optimize groups while keeping the remaining parameters fixed, (ii) optimize model parameters Λ while keeping the groups and the segmentation fixed, (iii) optimize the segmentation while keeping the remaining parameters fixed. We approach the first subproblem by iteratively optimizing group assignment of each node while maintaining the remaining nodes fixed. We show that such single round can be done in $\mathcal{O}\left(m + Rn + R^2H + K\right)$ time, where n is the number of nodes and m is the number of edges. Fortunately, the second subproblem is trivial since there is an analytic solution for optimal parameters, and we can obtain the solution in $\mathcal{O}\left(m + R^2H + K\right)$ time. Finally, we show that we can find the optimal segmentation with a dynamic program. Using a stock dynamic program leads to a computational complexity of $\mathcal{O}\left(m^2KH\right)$. Fortunately, we show that we can speed up the computation by using a SMAWK algorithm (Aggarwal et al., 1987), leading to a computational complexity of $\mathcal{O}\left(mKH + HR^2\right)$.

In summary, we extend a model by Corneli et al. (2018) to have recurring segments. We prove that the two variants of our main problem is **NP**-hard as well as several related optimization problems where we fix a subset of parameters. Navigating around these **NP**-hard problems we propose an iterative algorithm where a single iteration requires $\mathcal{O}\left(KHm + Rn + R^2H\right)$ time, a linear time in edges and nodes.

The rest of the paper is organized as follows. First we introduce preliminary notation, the model, and the optimization problems in Sect. 2. We then proceed to describe the iterative algorithm in Sect. 3. We present the related work in Sect. 4. Finally, we present our experiments with both fixed-membership and level-dependent membership problems in Sect. 5 and conclude the paper with discussion in Sect. 6. The computational complexity proofs are provided in Appendix.[1]

2 Preliminary notation and problem definition

Assume an undirected *temporal graph* $G = (V, E)$, where V is a set of nodes and E is a set of edges, where each edge is tuple (u, v, t) with $u, v \in V$ and t being the timestamp. We will use $n = |V|$ to denote the number of nodes and $m = |E|$ the number of edges. For simplicity, we assume that we do not have self-loops, though the models can be adjusted for such case. We write $t(e)$ to mean the timestamp of the edge e. We also write $N(u)$ to denote all the edges adjacent to a node $u \in V$.

Perhaps the simplest way to model a graph (with no temporal information) is with Erdos–Renyi model, where each edge is sampled independently from a Bernoulli probability parameterized with q. Let us consider two natural extensions of this model. The first extension is a block model, where nodes are divided into k groups, and an edge (u, v) are modelled with a Bernoulli probability parameterized with q_{ij}, where i is the group of u and j is the group of v. Given a graph, the optimization problem is to cluster nodes into groups so that the likelihood of the model is optimized. For the sake of variability we will use the words *group* and *cluster* interchangeably.

A convenient way of modelling events in temporal data is using Poisson process: Assume that you have observed c events with timestamps $t_1, \ldots, t_c$ in a time interval T of length Δ. The log-likelihood of observing these events at these exact times is equal to $c \log \lambda - \lambda\Delta$, where λ is a model parameter. Note that the log-likelihood does not depend on the individual timestamps.

If we were to extend the block model to temporal networks, the log-likelihood of c edges occurring between the nodes u and v in a time interval is equal to $c \log \lambda_{ij} - \lambda_{ij}\Delta$, where λ_{ij} is the Poisson process parameter and i is the group of u and j is the group of v. Note that λ_{ij} does not depend on the time, so discovering optimal groups is very similar to discovering groups in a static model.

A natural extension of this model, proposed by Corneli et al. (2018), is to make the parameters depend on time. Here, we partition the model into k segments and assign different set of λs to each segment.

More formally, we define a *time interval* T to be a continuous interval either containing the starting point $T = [t_1, t_2]$ or excluding the starting point $T = (t_1, t_2]$. In both cases, we define the duration as $\Delta(T) = t_2 - t_1$.

Given a time interval T, let us define

$$c(u, v, T) = |\{e = (u, v, t) \in E \mid t \in T\}|$$

to be the number of edges between u and v in T.

The log-likelihood of Poisson model for nodes u, v and a time interval T is

[1] The appendix is available as a supplementary material, and also at https://arxiv.org/abs/2205.09862.

$$\ell(u, v, T, \lambda) = c(u, v, T) \log \lambda - \lambda \Delta(T).$$

We extend the log-likelihood between the two sets of nodes U and W, by writing

$$\ell(U, W, T, \lambda) = \sum_{u,w \in U \times W} \ell(u, w, T, \lambda),$$

where $U \times W$ is a set of all node pairs $\{u, w\}$ with $u \in U$ and $w \in W$ and $u \neq v$. We consider $\{u, w\}$ and $\{w, u\}$ the same, so only one of these pairs is visited.

Given a time interval $D = [a, b]$, a K-segmentation $\mathcal{T} = T_1, \ldots, T_K$ is a sequence of K time intervals, such that $T_1 = [a, t_1], T_2 = (t_1, t_2], \ldots T_i = (t_{i-1}, t_i], \ldots$, and $T_K = (t_{K-1}, b]$. For notational simplicity, we require that the boundaries t_i must collide with the timestamps of individual edges. We also assume that D covers the edges. If D is not specified, then it is set to be the smallest interval covering the edges.

Given a K-segmentation, a partition of nodes $\mathcal{P} = P_1, \ldots, P_R$ into R groups, and a set of $KR(R+1)/2$ parameters $\Lambda = \left\{\lambda_{ijk}\right\}$,[2] the log-likelihood is equal to

$$\ell(\mathcal{P}, \mathcal{T}, \Lambda) = \sum_{i=1}^{R} \sum_{j=i}^{R} \sum_{k=1}^{K} \ell\left(P_i, P_j, T_k, \lambda_{ijk}\right).$$

This leads immediately to the problem considered by Corneli et al. (2018).

Problem 2.1 (SFix) Given a temporal graph G, a time interval D, integers R and K, find a node partition with R groups, a K-segmentation, and a set of parameters Λ so that $\ell(\mathcal{P}, \mathcal{T}, \Lambda)$ is maximized.

We should point out that for fixed $\mathcal{P}$ and $\mathcal{T}$, the optimal Λ is equal to

$$\lambda_{ijk} = \frac{c\left(P_i, P_j, T_k\right)}{\left|P_i \times P_j\right| \Delta\left(T_k\right)}.$$

In this paper we consider an extension of (K, R) model. Many temporal network exhibit cyclic or repeating behaviour. Here, we allow network to have K segments but we also limit the number of distinct parameters to be at most $H \leq K$. If we select $H < K$, then we are forcing that certain segments *share* their parameters, that is, we force repeating behaviour. We do not know beforehand which segments should share the parameters.

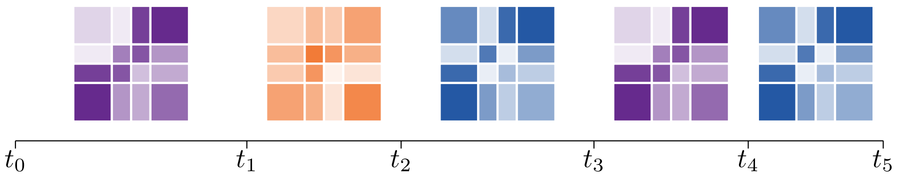
We can express this constraint more formally by introducing a mapping $g : [K] \rightarrow [H]$ that maps a segment index to its matching parameters. We can now define the likelihood as follows: given a K-segmentation, a partition of nodes $\mathcal{P} = P_1, \ldots, P_R$ into R groups, a mapping $g : [K] \rightarrow [H]$, and a set of $HR(R+1)/2$ parameters $\Lambda = \left\{\lambda_{ijh}\right\}$, the log-likelihood is equal to

$$\ell(\mathcal{P}, \mathcal{T}, g, \Lambda) = \sum_{i=1}^{R} \sum_{j=i}^{R} \sum_{k=1}^{K} \ell\left(P_i, P_j, T_k, \lambda_{ijg(k)}\right).$$

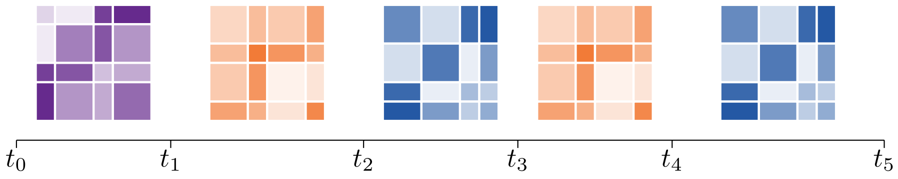
We will refer to g as *level mapping*.

This leads to the following optimization problem.

[2] For notational simplicity we will equate λ_{ijh} and λ_{jih}.



(a) Fixed membership model with 4 groups, 5 segments, and 3 levels.



(b) Level-dependent membership model with 4 groups, 5 segments, and 3 levels.

Fig. 1 Toy examples of models with 4 groups, 5 segments, and 3 levels. The lightness of the color in the matrices illustrates the parameters Λ while the color indicates which segments share the same parameters. For illustrative purposes the node groups are shown as continuous regions in matrices

Problem 2.2 (RSFix) Given a temporal graph G, a time interval D, integers R, H, and K, find a node partition with R groups, a K-segmentation, a level mapping $g : [K] \to [H]$, and parameters Λ maximizing $\ell(\mathcal{P}, \mathcal{T}, g, \Lambda)$.

We will refer to this problem as *fixed membership problem*.

Next we extend the previous model by allowing group memberships to vary over time. More specially, group membership of a node is attached to a level, similar to Λ. Hence, a node partition which is constrained on level is denoted as $\mathcal{P} = \left\{P_{ih}\right\}$. The log-likelihood of this new model is given by

$$\ell(\mathcal{P}, \mathcal{T}, g, \Lambda) = \sum_{i=1}^{R} \sum_{j=i}^{R} \sum_{k=1}^{K} \ell\left(P_{ig(k)}, P_{jg(k)}, T_k, \lambda_{ijg(k)}\right).$$

Problem 2.3 (RSDep) Given a temporal graph G, a time interval D, integers R, H, and K, find a set of H partitions of nodes $\mathcal{P}_h = P_{1h}, \ldots, P_{Rh}$, for all $h = 1, \ldots, H$, into R groups, a K-segmentation, a level mapping $g : [K] \to [H]$, and parameters Λ maximizing $\ell(\mathcal{P}, \mathcal{T}, g, \Lambda)$.

We will refer to this problem as *level-dependent membership problem*.

Toy examples of both models is given in Fig. 1.

3 Fast algorithm for obtaining good model

In this section we will introduce an iterative, fast approach for obtaining a good model. The computational complexity of one iteration is $\mathcal{O}\big(KHm + Rn + R^2H\big)$, which is linear in both the nodes and edges.

3.1 Iterative approach

Unfortunately, finding optimal solution for RSFix and RSDep is **NP**-hard.

Proposition 1 *Problems 2.2–2.3 are* **NP**-*hard, even for* $H = K = 1$ *and* $R = 2$.

Proof of Proposition 1 is in Appendix.

Consequently, we resort to a natural heuristic approach, where we optimize certain parameters while keeping the remaining parameters fixed.

Algorithm 1 Main loop of the algorithm

```
1 $\mathcal{P} \leftarrow$ random groups;  $\Lambda \leftarrow$ random values;
2 $\mathcal{T}, g \leftarrow$ FindSegments($\mathcal{P}, \Lambda$);
3 $\Lambda \leftarrow$ UpdateLambda($\mathcal{P}, \mathcal{T}, g$);
4 while convergence do
5     $\mathcal{P} \leftarrow$ FindGroups($\mathcal{P}, \Lambda, \mathcal{T}, g$);
6     $\Lambda \leftarrow$ UpdateLambda($\mathcal{P}, \mathcal{T}, g$);
7     $\mathcal{T}, g \leftarrow$ FindSegments($\mathcal{P}, \Lambda$);
8     $\Lambda \leftarrow$ UpdateLambda($\mathcal{P}, \mathcal{T}, g$);
```

We split RSFix into 3 subproblems as shown in Algorithm 1. First, we find good groups, then update Λ, and then optimize segmentation, followed by yet another update of Λ.

When initializing, we select groups $\mathcal{P}$ and parameters Λ randomly, then proceed to find optimal segmentation, followed by optimizing Λ.

The following proposition states the the complexity of Algorithm 1, and it is proven in the following subsections.

Proposition 2 *The computational complexity of a single iteration of Algorithm* 1 *is* $\mathcal{O}\big(KHm + Rn + R^2H\big)$.

Proof The proof follows from Lemmas 3.1, 3.2, and 3.3. □

Next we will explain each step in details followed by their computational complexities. Finally, we describe the difference in algorithm for RSDep .

3.2 Finding groups

Our first step is to update groups $\mathcal{P}$ while maintaining the remaining parameters fixed. Unfortunately, finding the optimal solution for this problem is **NP**-hard.

Proposition 3 *Finding optimal partition $\mathcal{P}$ for fixed Λ, $\mathcal{T}$ and g is* **NP**-*hard, even for $H = K = 1$ and $R = 2$.*

Proof of Proposition 3 is in Appendix.

Due to the previous proposition, we perform a simple greedy optimization where each node is individually reassigned to the optimal group while maintaining the remaining nodes fixed.

We should point out that there are more sophisticated approaches, for example based on SDP relaxations, see a survey by Abbe (2017). However, we resort to a simple greedy optimization due to its speed.

A naive implementation of computing the log-likelihood gain for a single node may require $\Theta(m)$ steps, which would lead in $\Theta(nm)$ time as we need to test every node. Luckily, we can speed-up the computation using the following straightforward proposition.

Proposition 4 *Let $\mathcal{P}$ be the partition of nodes, Λ set of parameters, and $\mathcal{T}$ and g the segmentation and the level mapping. Let $\mathcal{S}_h = \left\{T_k \in \mathcal{T} \mid h = g(k)\right\}$ be the segments using the hth level.*

Let u be a node, and let P_b be the set such that $u \in P_b$. Select P_a, and let $\mathcal{P}'$ be the partition where u has been moved from P_b to P_a. Then

$$\ell\left(\mathcal{P}', \mathcal{T}, g, \Lambda\right) - \ell(\mathcal{P}, \mathcal{T}, g, \Lambda) = Z + \sum_{h=1}^{H} \lambda_{bah} t_h + \sum_{j=1}^{R} c_{jh} \log \lambda_{ajh} - \left|P_j\right| \lambda_{ajh} t_h,$$

where Z is a constant, not depending on a, $t_h = \Delta\left(\mathcal{S}_h\right)$ is the total duration of the segments using the hth level and $c_{jh} = c\left(u, P_j, \mathcal{S}_h\right)$, is the number of edges between u and P_j in the segments using the hth level.

Proof Let us write $\mathcal{Q}$ to be the partition obtained from $\mathcal{P}$ by deleting u, that is $Q_b = P_b \setminus \{u\}$ and $Q_j = P_j$ for $j \neq b$. Note that

$$\ell\left(P'_a, P'_j, T, \lambda\right) - \ell\left(Q_a, Q_j, T, \lambda\right) = \ell\left(u, P'_j, T, \lambda\right)$$

for any T and λ. Moreover, $\ell\left(P'_i, P'_j, T, \lambda\right) = \ell\left(Q_i, Q_j, T, \lambda\right)$ if $i,j \neq a$. Write, for brevity, $\Delta = \ell\left(\mathcal{P}', \mathcal{T}, g, \Lambda\right) - \ell(\mathcal{Q}, \mathcal{T}, g, \Lambda)$. The score difference is equal to

$$\begin{aligned}
\Delta &= \sum_{i\leq j}\sum_{k=1}^{K} \ell\left(P'_i, P'_j, T_k, \lambda_{ijg(k)}\right) - \ell\left(Q_i, Q_j, T_k, \lambda_{ijg(k)}\right) \\
&= \sum_{j=1}^{R}\sum_{k=1}^{K} \ell\left(P'_a, P'_j, T_k, \lambda_{ajg(k)}\right) - \ell\left(Q_a, Q_j, T_k, \lambda_{ajg(k)}\right) \\
&= \sum_{j=1}^{R}\sum_{k=1}^{K} \ell\left(u, P'_j, T_k, \lambda_{ajg(k)}\right) \\
&= \sum_{j=1}^{R}\sum_{h=1}^{H} \ell\left(u, P'_j, \mathcal{S}_h, \lambda_{ajh}\right) \\
&= \sum_{j=1}^{R}\sum_{h=1}^{H} c\left(u, P'_j, \mathcal{S}_h\right) \log \lambda_{ajh} - t_h \left|Q_j\right| \lambda_{ajh} \\
&= \sum_{j=1}^{R}\sum_{h=1}^{H} c_{jh} \log \lambda_{ajh} - t_h \left|Q_j\right| \lambda_{ajh} \\
&= \sum_{h=1}^{H} \lambda_{bah} t_h + \sum_{j=1}^{r} c_{jh} \log \lambda_{ajh} - t_h \left|P_j\right| \lambda_{ajh} \quad .
\end{aligned}$$

Here we used the fact that $c\left(u, P_j, \mathcal{S}_h\right) = c\left(u, P'_j, \mathcal{S}_h\right)$. The claim follows by setting $Z = \ell(\mathcal{Q}, \mathcal{T}, g, \Lambda) - \ell(\mathcal{P}, \mathcal{T}, g, \Lambda)$. ☐

The proposition leads to the pseudo-code given in Algorithm 2. The algorithm computes an array c and then uses Proposition 4 to compute the gain for each swap, and consequently to find the optimal gain.

Algorithm 2 Algorithm FindGroups $(\mathcal{P}, \Lambda)$ for finding groups for a fixed segmentation $\mathcal{T}$, g and parameters Λ

```
1  $p(v) \leftarrow$ group index of $v$;
2  $s(e) \leftarrow$ segment index of $e$;
3  $d[h] \leftarrow \sum_{g(k)=h} \Delta(T_k)$;
4  **foreach** $v \in V$ **do**
5      $b \leftarrow p(v)$;
6      $c[j, h] \leftarrow$ array $c_{jh}$ as defined in Proposition 4;
7      **foreach** $a = 1, \ldots R$ **do**
8          $x[a] \leftarrow \sum_{h=1}^{H} \lambda_{bah} d[h] + \sum_{j=1}^{R} c[j, h] \log \lambda_{ajh} - |P_j| \lambda_{ajh} d[h]$ ;
9      $p(v) \leftarrow \arg\max_a x[a]$ (update $\mathcal{P}$ also);
10 **return** $\mathcal{P}$;
```

Lemma 3.1 *FindGroups runs in* $\mathcal{O}\left(m + Rn + R^2 H + K\right)$ *time.*

Proof Computing the array requires iterating over the adjacent edges, leading to $\mathcal{O}(|N(v)|)$ time where $|N(v)| = 2m$. Computing the gains requires $\mathcal{O}\left(R^2H\right)$ time.

The running time can be further optimized by modifying Line 8. There are at most $2m$ non-zero $c[i, j]$ entries (across all $v \in V$), consequently we can speed up the computation of a second term by ignoring the zero entries in $c[i, j]$. In addition, for each a, the remaining terms

$$\sum_{h=1}^{H} \lambda_{bah} d[h] - \sum_{j=1}^{R} \left|P_j\right| \lambda_{ajh} d[h]$$

can be precomputed in $\mathcal{O}(RH)$ time and maintained in $\mathcal{O}(1)$ time. In line 9, for each node we pick the group that maximizes the gain which requires $\mathcal{O}(Rn)$ time. This leads to a running time of $\mathcal{O}\left(m + Rn + R^2H + K\right)$. □

3.3 Updating Poisson process parameters

Our next step is to update Λ while maintaining the rest of the parameters fixed. This refers to UpdateLambda in Algorithm 1. Fortunately, this step is straightforward as the optimal parameters are equal to

$$\lambda_{ijh} = \frac{c\left(P_i, P_j, \mathcal{S}_h\right)}{\left|P_i \times P_j\right| \Delta\left(\mathcal{S}_h\right)},$$

where $\mathcal{S}_h = \left\{T_k \in \mathcal{T} \mid h = g(k)\right\}$ are the segments using the hth level.

Lemma 3.2 *Updating the* Λ *parameters requires* $\mathcal{O}\left(m + R^2H + K\right)$ *time.*

Proof First we need to count the number of interactions between ith group and jth group within hth level by iterating over the edges which requires $\mathcal{O}\left(m + R^2H\right)$ time. Computing $\Delta\left(\mathcal{S}_h\right)$ requires $\mathcal{O}(K + H)$ time. Next we update parameters by executing a triple loop which runs twice over the groups and once over the levels, requiring $\mathcal{O}\left(R^2H\right)$ time. Consequently, updating the parameters requires $\mathcal{O}\left(m + R^2 H + K\right)$ time which proves the claim. □

In practice, we would like to avoid having $\lambda = 0$ as this forbids any edges occurring in the segment, and we may get stuck in a local maximum. We approach this by shifting λ slightly by using

$$\lambda_{ijh} = \frac{c\left(P_i, P_j, \mathcal{S}_h\right) + \theta}{\left|P_i \times P_j\right| \Delta\left(\mathcal{S}_h\right) + \eta},$$

where θ and η are user parameters.

3.4 Finding segmentation

Our final step is to update the segmentation $\mathcal{T}$ and the level mapping g, while keeping Λ and $\mathcal{P}$ fixed. Luckily, we can solve this subproblem in linear time.

Note that we need to keep Λ fixed, as otherwise the problem is **NP**-hard.

Proposition 5 *Finding optimal* Λ, $\mathcal{T}$ *and* g *for fixed* $\mathcal{P}$ *is* **NP**-*hard.*

Proof of Proposition 5 is in Appendix.

On the other hand, if we fix Λ, then we can solve the optimization problem with a dynamic program. To be more specific, assume that the edges in E are ordered, and write $o[e, k]$ to be the log-likelihood of k-segmentation covering the edges prior and including e. Given two edges $s, e \in E$, let $y(s, e; h)$ be the log-likelihood of a segment $(t(s), t(e)]$ using the hth level of parameters, $\lambda_{\cdot\cdot h}$. If s occurs after e we set y to be $-\infty$. Then the identity

$$o[e,k] = \max_h \max_s y(s,e;h) + o[s,k-1]$$

leads to a dynamic program.

Using an off-the-shelf approach by Bellman (1961) leads to a computational complexity of $\mathcal{O}\left(m^2KH\right)$, assuming that we can evaluate $y(s, e; h)$ in constant time.

However, we can speed-up the dynamic program by using the SMAWK algorithm (Aggarwal et al., 1987). Given a function $x(i, j)$, where $i,j = 1, \ldots, m$, SMAWK computes $z(j) = \arg\max_i x(i,j)$ in $\mathcal{O}(m)$ time, under two assumptions. The first assumption is that we can evaluate x in constant time. The second assumption is that x is *totally monotone*. We say that x is totally monotone, if $x(i_2,j_1) > x(i_1,j_1)$, then $x(i_2,j_2) \geq x(i_1,j_2)$ for any $i_1 < i_2$ and $j_1 < j_2$.

We have the immediate proposition.

Proposition 6 *Fix* h. *Then the function* $x(s, e) = y(s, e;h) + o[s, k-1]$ *is totally monotone.*

Proof Assume four edges s_1, s_2, e_1 and e_2 with $t(s_1) \leq t(s_2)$ and $t(e_1) \leq t(e_2)$. We can safely assume that $t(s_2) \leq t(e_1)$. We can write the difference as

$$x(s_1,e_2) - x(s_1,e_1) = y(e_1,e_2;h) = x(s_2,e_2) - x(s_2,e_1).$$

Thus, if $x(s_2,e_1) > x(s_1,e_1)$, then $x(s_2,e_2) > x(s_1,e_2)$, completing the proof. □

Our last step is to compute x in constant time. This can be done by first precomputing $f[e, h]$, the log-likelihood of a segment starting from the epoch and ending at $t(e)$ using the hth level. The log-likelihood of a segment is then $y(s,e;h) = f[e,h] - f[s,h]$, which we can compute in constant time.

Algorithm 3 Algorithm FindSegments$(\mathcal{P}, \Lambda)$ for finding optimal segmentation for fixed groups $\mathcal{P}$ and parameters Λ

```
1  $t_{min} \leftarrow \min\{t \mid (u,v,t) \in E\}$;
2  $f[e,h] \leftarrow$ log-likelihood of a segment $[t_{min}, t(e)]$ using parameters $\lambda_{\cdot\cdot h}$ ;
3  foreach $e \in E$ do  $o[e,1] \leftarrow \max_h f[e,h]$ ;
4  foreach $k = 2, \ldots, K$ do
5      $x(s,e;h) \leftarrow o[s,k-1] + f[e,h] - f[s,h]$;
6      foreach $h = 1, \ldots, H$ do
7          $z[e,h] \leftarrow \arg\max_s x(s,e;h)$ for each $e \in E$ (use SMAWK);
8      $o[e,k] \leftarrow \max_h x(z[e,h],e;h)$ for each $e \in E$;
9      $r[e,k] \leftarrow \arg\max_h x(z[e,h],e;h)$;
10     $q[e,k] \leftarrow z[e,r[e,k]]$;
11 Use $r$ and $q$ to recover the optimal segmentation $(T_1, \ldots, T_K)$ and the
     level mapping $g$ ;
12 return $(T_1, \ldots, T_K)$, $g$;
```

The pseudo-code for finding the segmentation is given in Algorithm 3. A more detailed version of the pseudo-code is given in Appendix. Here, we first precompute $f[e, h]$. We then solve segmentation with a dynamic program by maintaining 3 arrays: $o[e, k]$ is the log-likelihood of k-segmentation covering the edges up to e, $q[e, k]$ is the starting point of the last segment responsible for $o[e, k]$, and $r[e, k]$ is the level of the last segment responsible for $o[e, k]$.

In the inner loop we use SMAWK to find optimal starting points. Note that we have to do this for each h, and only then select the optimal h for each segment. Note that we do define x on Line 5 but we do not compute its values. Instead this function is given to SMAWK and is evaluated in a lazy fashion.

Once we have constructed the arrays, we can recursively recover the optimal segmentation and the level mapping from q and r, respectively.

Lemma 3.3 *FindSegments runs in $\mathcal{O}(mKH + HR^2)$ time.*

Proof SMAWK runs in $\mathcal{O}(m)$ time and we need to call SMAWK $\mathcal{O}(HK)$ times. Therefore it requires $\mathcal{O}(mKH)$ time to find the optimal segmentation, given precomputed f mentioned in Line 2 in Algorithm 3. To compute f, we need to compute α as stated in Line 5 of the detailed version of the algorithm; Algorithm 5. To compute α, we loop over the groups twice and once over the levels which requires $\mathcal{O}(HR^2)$ time. Consequently, computational complexity of FindSegments is $\mathcal{O}(mKH + HR^2)$. □

We were able to use SMAWK because the optimization criterion turned out to be totally monotone. This was possibly only because we fixed Λ. The notion of using SMAWK to speed up a dynamic program with totally monotone scores was proposed by Galil and Park (1990). Fleischer et al. (2006); Hassin and Tamir (1991) used this approach to solve dynamic program segmenting monotonic one-dimensional sequences with L_1 cost.

We fixed Λ because Proposition 5 states that the optimization problem for $H < K$ cannot be solved in polynomial time if we optimize $\mathcal{T}$, g, and Λ at the same time. Proposition 5 is the main reason why we cannot use directly the ideas proposed by Corneli et al. (2018) as the authors use the dynamic program to find $\mathcal{T}$ and Λ at the same time.

However, if $K = H$, then the problem is solvable with a dynamic program but requires $\mathcal{O}(Km^2R^2)$ time. This may be impractical for large values of m, so using Algorithm 1 may be preferable even when $K = H$. The downside of the approach is that having more variables fixed may lead to the algorithm getting to the local optimum.

We should point out we can also approximate the aforementioned segmentation problem, if we consider it as a minimization problem and shift the cost with a constant so that it is always positive, then using algorithms by Tatti (2019); Guha et al. (2006) we can obtain $(1+\epsilon)$-approximation with $\mathcal{O}(K^3 \log K \log m + K^3\epsilon^{-2} \log m)$ number of cost evaluations. Finding the optimal parameters and computing the cost of a single segment can be done in $\mathcal{O}(R^2)$ time with $\mathcal{O}(R^2 + m)$ time for precomputing. This leads to a total time of $\mathcal{O}(R^2(K^3 \log K \log m + K^3\epsilon^{-2} \log m) + m)$ for the special case of $K = H$.

Algorithm 4 Modified algorithm FindGroups $(\mathcal{P}, \Lambda)$ for finding level-specific groups for a fixed segmentation $\mathcal{T}$, g and parameters Λ

```
1  $p(v,h) \leftarrow$ group index of $v$ for level $h$;
2  $d[h] \leftarrow \sum_{g(k)=h} \Delta(T_k)$;
3  foreach $h = 1, \ldots, H$ do
4      foreach $v \in V$ do
5          $b \leftarrow p(v,h)$;
6          $c[j] \leftarrow c(v, P_j, \mathcal{S}_h)$;
7          foreach $a = 1, \ldots R$ do
8              $x[a] \leftarrow \lambda_{bah} d[h] + \sum_{j=1}^{R} c[j] \log \lambda_{ajh} - |P_j| \lambda_{ajh} d[h]$;
9          $p(v,h) \leftarrow \arg\max_a x[a]$ (update $\mathcal{P}$ also);
10 return $\mathcal{P}$;
```

At the end of each subproblem, the log-likelihood either improves or stays constant. Therefore our algorithm always converges and have a finite number of iterations.

3.5 Solving level-dependent membership problem

Next we present the algorithm for solving RSDep .

Finding groups We continue to use a similar greedy approach as in Algorithm 2. The only difference is that when computing gain of a node, we pick only the adjacent edges residing in a particular level and then iterate over all levels in a separate fashion. We adopt Proposition 4, except now we apply it only for one level. The modified algorithm is stated in Algorithm 4. Moreover, the running time, $\mathcal{O}(m + R^2Hn + K)$, remains the same as of Algorithm 2.

Finding Poisson parameters and segmentation These algorithms for these two subproblems remain essentially unchanged: the only difference is that we use P_{ih} whenever P_i is involved.

4 Related work

The closest related work is the paper by Corneli et al. (2018) which can be viewed as a special case of our approach by requiring $K = H$, in other words, while the Poisson process may depend on time they do not take into account any recurrent behaviour. Having $K = H$ simplifies the optimization problem somewhat. While the general problem still remains difficult, we can now solve the segmentation $\mathcal{T}$ *and* the parameters Λ simultaneously using a dynamic program as was done by Corneli et al. (2018). In our problem we are forced to fix Λ while solving the segmentation problem. Interestingly enough, this gives us an advantage in computational time: we only need $\mathcal{O}\left(KHm + HR^2\right)$ time to find the optimal segmentation while the optimizing $\mathcal{T}$ and Λ simultaneously requires $\mathcal{O}\left(R^2Km^2\right)$ time. On the other hand, by fixing Λ we may have a higher chance of getting stuck in a local maximum.

The other closely related work is by Gionis and Mannila (2003), where the authors propose a segmentation with shared centroids. Here, the input is a sequence of real valued vectors and the segmentation cost is either L_2 or L_1 distance. Note that there is no notion of groups $\mathcal{P}$, the authors are only interested in finding a segmentation with recurrent sources. The authors propose several approximation algorithms as well as an iterative method. The approximation algorithms rely specifically on the underlying cost, in this case L_1 or L_2 distance, and cannot be used in our case. Interestingly enough, the proposed iterative method did not use SMAWK optimization, so it is possible to use the optimization described in Sect. 3 to speed up the iterative method proposed by Gionis and Mannila (2003).

In this paper, we used stochastic block model (see (Holland et al., 1983; Anderson et al., 1992), for example) as a starting point and extend it to temporal networks with recurrent sources. Several past works have extended stochastic block models to temporal networks: Yang et al. (2011); Matias and Miele (2017) proposed an approach where the nodes can change group memberships over time. In a similar fashion, Xu and Hero (2014) proposed a model where the adjacency matrix snapshots are generated with a logistic function whose latent parameters evolve over time. The main difference with our approach is that in these models the group memberships of nodes are changing while in our case we keep the memberships constant and update the probabilities of the nodes. Moreover, these methods are based on graph snapshots while we work with temporal edges. In another related work, Matias et al. (2018) modelled interactions using Poisson processes conditioned by stochastic block model. Their approach was to estimate the intensities non-parametrically through histograms or kernels while we model intensities with recurring segments. For a survey on stochastic block models, including extensions to temporal settings, we refer the reader to a survey by Lee and Wilkinson (2019).

Stochastic block models group similar nodes together; here similarity means that nodes in the same group have the similar probabilities connecting to nodes from other group. A similar notion but a different optimization criterion was proposed by Arockiasamy et al. (2016). Moreover, Henderson et al. (2012) proposed a method where nodes with similar neighborhoods are discovered.

In this paper we modelled the recurrency by forcing the segments to share their parameters. An alternative approach to discover recurrency is to look explicitly for recurrent patterns (Ozden et al., 1998; Han et al., 1998, 1999; Ma & Hellerstein, 2001; Yang et al., 2003; Galbrun et al., 2019). We should point out that these works are not design to work with graphs; instead they work with event sequences. We leave adapting this methodology for temporal networks as an interesting future line of work.

Using segmentation to find evolving structures in networks have been proposed in the past: Kostakis et al. (2017) introduced a method where a temporal network is segmented into k segments with $h < k$ summaries. A summary is a graph, and the cost of an individual segment is the difference between the summary and the snapshots in the segment. Moreover, Rozenshtein et al. (2020) proposed discovering dense subgraphs in individual segments.

5 Experimental evaluation

The goal in this section is to experimentally evaluate our algorithm. Towards that end, we first test how well the algorithm discovers the ground truth using synthetic datasets. Next we study the performance of the algorithm on real-world temporal datasets in terms of running time and likelihood. We compare our results to the following baselines: the running times are compared to a naive implementation where we do not utilize SMAWK algorithm, and the likelihoods are compared to the likelihoods of the (R, K) model.

We implemented the algorithm in Python[3] and performed the experiments using a 2.4 GHz Intel Core i5 processor and 16 GB RAM.

5.1 Experiments with fixed membership model

Synthetic datasets To test our algorithm, we generated 5 temporal networks with known groups and known parameters Λ which we use as a ground truth. To generate data, we first chose a set of nodes V, number of groups R, number of segments K, and number of levels H. Next we assumed that each node has an equal probability of being chosen for any group. Based on this assumption, the group memberships were selected at random.

We then randomly generated Λ from a uniform distribution. More specifically, we generated H distinct values for each pair of groups and map them to each segment. Note that, we need to ensure that each distinct level is assigned to at least one segment. To guarantee this, we first deterministically assigned the set of H levels to first H segments and the remaining $(K - H)$ segments are mapped by randomly selecting $(K - H)$ elements from H level set.

Given the group memberships and their related Λ, we then generated a sequence of timestamps with a Poisson process for each pair of nodes. The sizes of all synthetic datasets are given in Table 1.

Real-world datasets We used 7 publicly available temporal datasets. *Email-Eu-1* and *Email-Eu-2* are collaboration networks between researchers in a European research institution.[4] *Math Overflow* contains user interactions in Math Overflow web site while answering to the questions.[4] *CollegeMsg* is an online message network at the University of California,

[3] The source code is available at https://version.helsinki.fi/dacs/.

[4] http://snap.stanford.edu.

Table 1 Experiments for the synthetic datasets

Dataset	*n*	*m*	*R*	*K*	*H*	LL_g	*G*	LL_f	*I*	*CT*
Synthetic-1	50	76,332	2	2	2	0.95	1	0.94	2	2.81 s
Synthetic-2	30	95,889	3	3	3	0.94	1	0.94	3	5.36 s
Synthetic-3	20	65,056	3	3	3	0.97	1	0.97	3	3.91 s
Synthetic-4	60	537,501	3	4	3	0.94	1	0.93	3	23.13 s
Synthetic-5	10	33,475	2	10	5	0.91	1	0.91	4	10.27 s

Here, n is the number of nodes, m is the number of edges, R is the number of groups, K is the number of segments, H is the number of levels, LL_g is the normalized log-likelihood for the ground truth, G is the Rand index, LL_f is the discovered normalized log-likelihood, I is the number of iterations, and CT is the computational time in seconds

Table 2 Sizes and computational times for the real-time datasets

Dataset	*n*	*m*	*R*	*K*	*H*	RSFix			RSDep		
						LL_f	*I*	*CT*	LL_d	*I*	*CT*
Email-Eu-1	309	61,046	3	10	7	0.8935	12	188 s	0.8875	18	531 s
Email-Eu-2	162	46,772	4	8	7	0.872	9	177 s	0.8559	26	575 s
MathOverflow	21,688	107,581	2	3	2	0.9087	20	263 s	0.9082	21	313 s
CollegeMsg	899	59 835	3	8	5	0.8711	19	662 s	0.8503	20	592 s
MOOC	7047	411,749	2	3	2	0.811	6	208 s	0.7968	6	216 s
Bitcoin	3783	24,186	3	5	3	0.9272	17	25 s	0.901	19	29 s
Santander	735	33,116	3	7	5	0.9404	20	60 s	0.8923	5	1 s

Here, n is the number of nodes, m is the number of edges, R is the number of groups, K is the number of segments, H is the number of levels, LL_d and LL_f are the discovered normalized log-likelihoods using the respective level dependent membership model and fixed membership model, I is the number of iterations, and CT is the computational time in seconds

Irvine.[4] *MOOC* contains actions by users of a popular MOOC platform.[4] *Bitcoin* contains member rating interactions in a bitcoin trading platform.[4] *Santander* contains station-to-station links that occurred on Sep 9, 2015 from the Santander bikes hires in London.[5] The sizes of these networks are given in Table 2.

Results for synthetic datasets To evaluate the accuracy of our algorithm, we compare the set of discovered groups with the ground truth groups. Here, our algorithm found the ground truth: in Table 1 we can see that Rand index (Rand, 1971) (column G) is equal to 1,

Next we compare the log-likelihood values from true models against the log-likelihoods of discovered models. To evaluate the log-likelihoods, we normalize the log-likelihood, that is we computed $\ell(\mathcal{P}, \mathcal{T}, g, \Lambda)/\ell\left(\mathcal{P}', \mathcal{T}', g', \Lambda'\right)$, where $\mathcal{P}', \mathcal{T}', g', \Lambda'$ is a model with a single group and a single segment. Since all our log-likelihood values were negative, the *normalized log-likelihood* values were between 0 and 1, and *smaller* values are better.

[5] https://cycling.data.tfl.gov.uk.

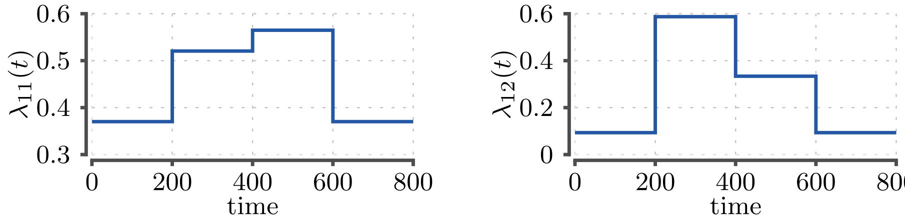


Fig. 2 Discovered parameters $\lambda_{11}(t)$, $\lambda_{12}(t)$ for the *Synthetic-4* dataset. Parameter $\lambda_{12}(t)$ implies the Poisson process parameter between group 1 and group 2 as a function of time

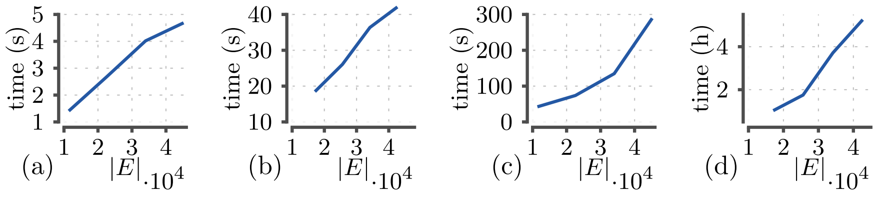


Fig. 3 Computational time as a function of number of temporal edges ($|E|$) for *Synthetic-large* (**a, c**) and *Santander-large* (**b, d**). This experiment was done with $R = 3$, $K = 5$, and $H = 3$ using SMAWK algorithm (**a–b**) and naive dynamic programming (**c–d**). The times are in seconds in (**a–c**) and in hours in (**d**)

As demonstrated in column LL_g and column LL_f of Table 1, we obtained similar normalized log-likelihood values when compared to the normalized log-likelihood of the ground truth. The obtained normalized log-likelihood values were all slightly better than the log-likelihoods of the generated models, that is, our solution is as good as the ground truth.

An example of the discovered parameters, λ_{11} and λ_{12}, for *Synthetic-4* dataset are shown in Fig. 2. The discovered parameters matched closely to the generated parameters with the biggest absolute difference being 0.002 for *Synthetic-4*. The figures for other values and other synthetic datasets are similar.

Computational time Next we consider the computational time of our algorithm. We varied the parameters R, K, and H for each dataset. The model parameters and computational times are given in Table 2 under RSFix section. From the last column of CT in RSFix section, we see that the running times are reasonable despite using inefficient Python libraries: for example we were able to compute the model for *MOOC* dataset, with over 400,000 edges, under four minutes. This implies that the algorithm scales well for large networks. This is further supported by a low number of iterations, column I in RSFix section of Table 2.

Next we study the computational time as a function of m, number of edges.

We first prepared 4 datasets with different number of edges from a real-world dataset; *Santander-large*. To vary the number of edges, we uniformly sampled edges without replacement. We sampled like a .4, .6, .8, and 1 fraction of edges.

Next we created 4 different *Synthetic-large* dataset with 30 nodes, 3 segments with unique λ values but with different number of edges. To do that, we gradually increase the number of Poisson samples we generated for each segment.

From the results in Fig. 3 we see that generally computational time increases as $|E|$ increases. For instance, a set of 17,072 edges accounts for 18.46 s whereas a set of 34,143 edges accounts for 36.36 s w.r.t *Santander-large*. Thus a linear trend w.r.t $|E|$ is evident via this experiment.

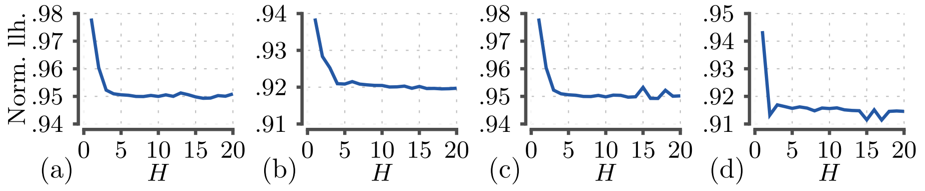


Fig. 4 Normalized log-likelihood as a function of number of levels (H) for the *Santander* dataset (**a**), *bitcoin* dataset (**b**), *Synthetic-5* dataset (**c**), and *Email-Eu-1* dataset (**d**). This experiment is done for $R = 2$, $K = 20$, and $H = 1, \ldots, 20$

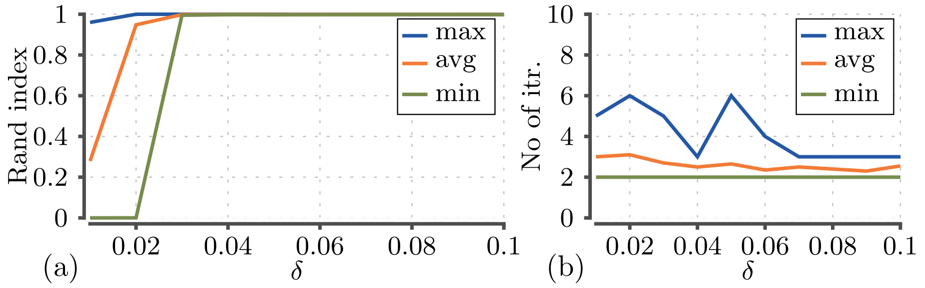


Fig. 5 **a** Gives Rand index as a function of δ and **b** gives number of iterations as a function of δ. This experiment is conducted for $R = 2$, $K = 4$, and $H = 2$ with fixed membership model

To emphasize the importance of SMAWK, we replaced it with a stock solver of the dynamic program, and repeat the experiment. We observe in Fig. 3 that computational time has increased drastically when stock dynamic program algorithm is used. For example, a set of 34,143 edges required 3.7 h for *Santander-large* dataset but only 36.36 s when SMAWK is used.

Likelihood vs number of levels Our next experiment is to study how normalized log-likelihood behaves upon the choices of H. We conducted this experiment for $K = 20$ and vary the number of levels (H) from $H = 1$ to $H = 20$. The results for the *Santander*, *Bitcoin*, *Synthetic-5*, and *Email-Eu-1* dataset are shown in Fig. 4. From the results we see that generally normalized log-likelihood decreases as H increases. That is due to the fact that higher the H levels, there exists a higher degree of freedom in terms of optimizing the likelihood. Note that if $H = K$, then our model corresponds to the model studied by Corneli et al. (2018). Interestingly enough, the log-likelihood values plateau for values of $H \ll K$ suggesting that existence of recurring segments in the displayed datasets.

Effect of Λ Next we study how does λ parameter affect the performance of our algorithm. To conduct our study with fixed membership model, we first prepared a collection of datasets with $R = 2$, $K = 4$, $H = 2$, and varying λ values. Since $H = 2$, we set two specific λ values for each pair of groups. We generate Λ as a function of δ such that $\lambda_{11\cdot} = (\delta, 0.01)$, $\lambda_{22\cdot} = (0.01, \delta)$, and $\lambda_{12\cdot} = (\delta, \delta)$. Next, we vary δ parameter from 0.01 to 0.1. When $\delta = 0.01$, each pair of λs are set with equal values and then we gradually increase the difference between them by varying δ parameter from 0.01 to 0.1. The case $\delta = 0.01$ corresponds to the event where all pairs of λ values are equal to 0.01. $\delta = 0.1$ corresponds to the case where 2 distinct λs are the furthest away from each other for both λ_{11} and λ_{22}. We generated 20 number of such datasets for each δ and used the same method of generating synthetic datasets as described in Sect. 5.1.

Table 3 Experiments for the synthetic datasets

Dataset	n	m	R	K	H	LL_g	LL_d	LL_f	G	I	CT
Synthetic-d-1	40	207,428	2	4	2	0.8698	0.8676	0.9278	1	3	15 s
Synthetic-d-2	20	45,868	2	3	2	0.8557	0.8541	0.9427	1	2	2 s
Synthetic-d-3	20	159,974	2	5	3	0.9457	0.9449	0.9541	1	3	23 s
Synthetic-d-4	15	151,332	2	8	4	0.9613	0.9597	0.9697	0.93	5	33 s
Synthetic-d-5	30	374,442	3	6	3	0.8858	0.8843	0.9173	1	4	46 s

Here, n is the number of nodes, m is the number of edges, R is the number of groups, K is the number of segments, H is the number of levels, LL_g is the ground truth normalized log-likelihood, G is the Rand index, LL_d and LL_f are the discovered normalized log-likelihoods using the respective level depedendent and fixed membership models, I is the number of iterations, and CT is the computational time in seconds

Next we ran our algorithm across 20 different datasets for each δ, and reported maximum, minimum, and average Rand indices for both group memberships and segments. To check the speed of convergence, we recorded maximum, minimum, and average number of iterations for the convergence. Interestingly, for all the datasets our algorithm always achieved Rand index of 1 for discovered groups with compared to the ground truth. Next, we will see how did the Rand index for segments vary with δ.

First, let us look at Fig. 5a which shows the average, maximum, and minimum Rand index for segments as a function of δ. First we observe the blue curve which corresponds to the maximum Rand index obtained. We can conclude that unless λs are too closer to each other; i.e $\delta < 0.02$, our algorithm always produced maximum Rand index more than 0.9 which indicates the ability to produce good solutions even when the datasets get more difficult to cope up. Next let us observe the average case which reaches Rand index close to 1 after $\delta = 0.03$. Therefore we can conclude that the accuracy of discovered segments are very high as δ approaches 0.03. Similar to average case, minimum case achieves its max at 0.03 and continues the value of 0.99 afterwards. In summary, at $\delta = 0.03$, all three curves approximately coincide with each other and maintain the same streak beyond.

Second, let us observe Fig. 5b which shows the number of iterations took until the convergence. We can observe that minimum number of iterations is always 2 over all δ values that we experimented. Furthermore, average number of iterations always lies between 2 and 3.1. In the maximum case we see that the number of iterations lies between 3 and 6 which implies a fast convergence. Finally, we can conclude that the number of iterations are not directly correlated with the ground truth λ values of the dataset.

5.2 Experiments with level-dependent membership model

Synthetic datasets When generating synthetic datasets for level-dependent model, we used the same procedure as with the fixed membership model, except we assigned a group to a node depending on its specific current level. That means that a node can have several group memberships over the course of timeline depending on its timestamp instance. We prepared 5 synthetic datasets in total; characteristics and results are shown in Table 3.

First, for each dataset, we compare the normalized log-likelihood values from the ground truth model against the fixed membership and level-dependent membership models. We observe that in these datasets, discovered normalized log-likelihood value obtained using level-dependent membership model (column LL_d) is always approximately equal and

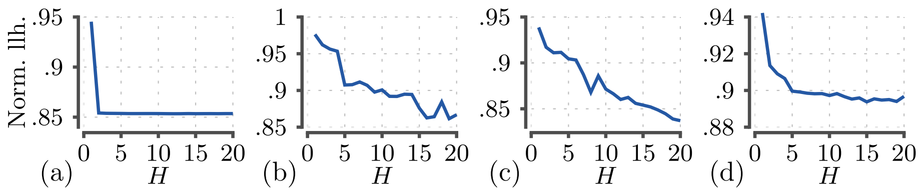


Fig. 6 Normalized log-likelihood as a function of number of levels (H) for the *Synthetic-2* dataset (**a**), *Santander* dataset (**b**), *Bitcoin* dataset (**c**), and *Email-Eu-2* dataset (**d**). This experiment is done for $R = 2$, $K = 20$, and $H = 1, \ldots, 20$. The reported scores are minimum of 20 restarts

even slightly better than the ground truth (column LL_g). As shown in LL_f column, even though fixed membership model produces a competitive score, the values are always worse than the ground truth. Next, we compare the group sets discovered using level-dependent model, with the ground truth groups. Here, we can see that Rand index (column G) is equal to 1 except for *Synthetic-d-4*. Finally, we see the running times which are shown in last column of Table 3. For over 100,000 edges, our algorithm converges in less than a minute.

Real-world datasets For each dataset, we conducted the experiments based on two group membership models. The statistics of discovered log-likelihood values for real-world datasets are shown in Table 2 under RSDep section. First, let us look at column LL_d and LL_f in RSDep section. For all datasets, normalized log-likelihood values obtained for level-dependent membership model are better than fixed membership model. This phenomenon illustrates that level-dependent membership model utilizes its higher degree of freedom while optimizing group memberships with compared to the case where memberships remain constant. The datasets *Email-Eu-2*, *CollegeMsg*, and *MOOC* have a significant gap between the likelihood values of two models indicating that the level-dependent model fit these datasets better. Nevertheless, *MathOverflow* dataset creates nearly equal likelihood results with both models that group membership of a node stays almost constant over all the segments is evident.

Likelihood vs number of levels Next we study the effect of H towards obtained normalized log-likelihood. We conducted this experiment for $K = 20$ and $R = 2$, then vary the number of levels (H) from $H = 1$ to $H = 20$. For each level, we repeated the experiment 20 times and took the minimum of normalized log likelihood values over all such repetitions. The results for the *Synthetic-2*, *Santander*, *Bitcoin*, and *Email-Eu-2* dataset are shown in Fig. 6. From the results we see that generally normalized log-likelihood decreases as H increases. That is due to the fact that higher the H levels, there exists more flexibility in terms of optimizing the group memberships. For *Synthetic-2* the log-likelihood values generally are plateauing indicating no significant variation in model parameters from $H = 2$ which is the ground truth number of H levels. Same plateauing behaviour is evident in *Email-Eu-2* dataset after $H = 15$ suggesting recurrent segments even if group memberships are allowed to vary from segment to segment.

Effect of Λ Next, we study the affect of λs towards the performance of our algorithm using level-dependent membership model. As described at the end of Sect. 5.1, we adhered to the same process of generating λ varying datasets. Only difference here is that group memberships are also allowed to vary based on the level. Over all the datasets, our algorithm achieved Rand index of 1 for discovered group memberships. Next, we will observe the Rand index for discovered segments.

First, let us look at Fig. 7a which shows the average, maximum, and minimum Rand index for segments as a function of δ. For the max case, we can observe that

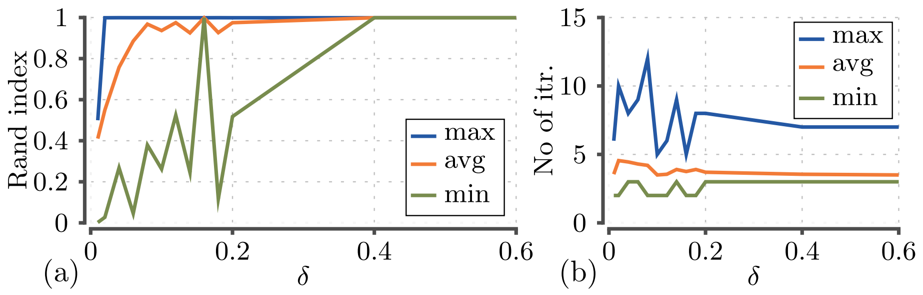


Fig. 7 **a** Gives Rand index as a function of δ and **b** gives number of iterations as a function of δ. This experiment is conducted for $R = 2$, $K = 4$, and $H = 2$ with level-dependent model

unless $\delta < 0.02$ our algorithm could discover segments with a Rand index value more than 0.99. For the average case, we can initially observe an increasing trend between $\delta = 0.01$ and $\delta = 0.08$. Afterwards, the curve tends to fluctuate from $\delta = 0.08$ to $\delta = 0.2$ and becomes constant at Rand index of 0.99 after $\delta = 0.4$. For the minimum case, we can see that Rand index tends to fluctuate until $\delta < 0.2$ and then reaches its maximum at $\delta = 0.4$ and continues to be approximately constant at its max afterwards. In summary, at $\delta = 0.04$, all three curves coincide with each other at a value close to 1.

Second, let us observe Fig. 7b which shows the number of iterations took until the convergence. We can observe that minimum number of iterations is always either 2 or 3. For the average case, average number of iterations always lies between 3 and 5 which shows a sufficiently fast convergence on average. Among all cases we experimented, except 2 outlier cases, in all other cases maximum number of iterations lies below 10. Similar to fixed membership model, we can not observe any trend that varies with δ.

5.3 Case study

Finally we present a case study based on London cycling hire usage dataset[4] in which the interactions were occurred on 9th of September, 2015. The size of this network is given in Table 2. Our goal is to discover how the clusters are partitioned, that is, can we find some hidden pattern behind their locations and how did the λ parameters behave during the course of the day, that is, whether our model can discover peak and non-peak recurrent patterns accurately. We conducted the experiment for $R = 5$, $K = 5$, and $H = 2$. That is, we are interested in segmenting the day into 5 segments with only 2 levels with the intention of finding peak and non-peak hours using Algorithm 1.

First, let us look at the clusters obtained from London cycling dataset which are shown in Fig. 8. In general, we can observe that the stations which are geographically close to each other are most likely to be partitioned together into the same cluster. This is due to the fact that the stations which are grouped together are most likely to interact with the stations from other clusters and the stations within the same cluster in similar fashion.

Figure 9 shows how did the λ parameters vary during the day. Interestingly we can clearly observe 2 peak segments during the day; approximately, one peak segment lies in the morning from 7.15 AM to 9.30 AM and the other peak segment lies in the evening from 4 PM to 7.45 PM. We can observe in Fig. 9 that the highest interaction levels of 10^{-5} were attained for both $\lambda_{14}(t)$ and $\lambda_{44}(t)$ which means that cluster 4 which has shown in

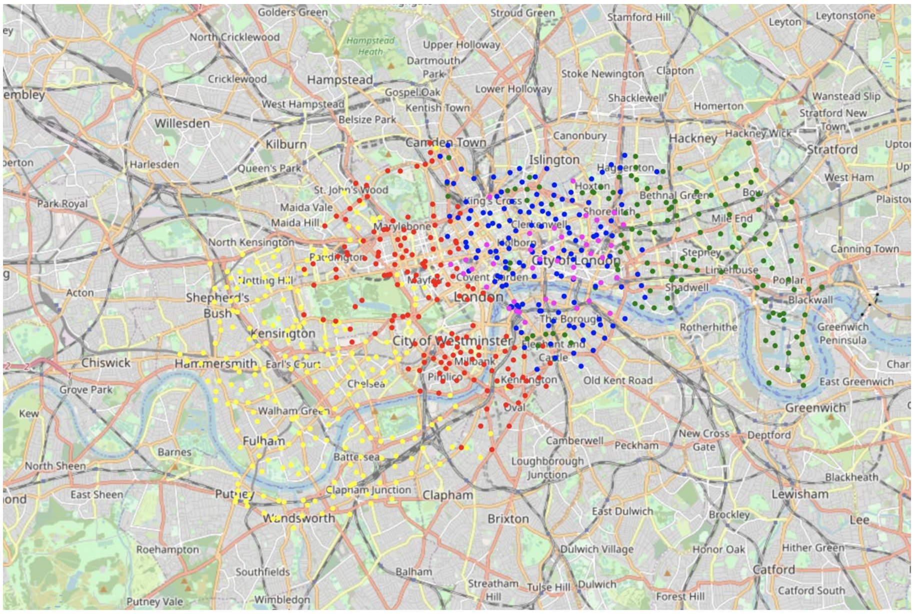


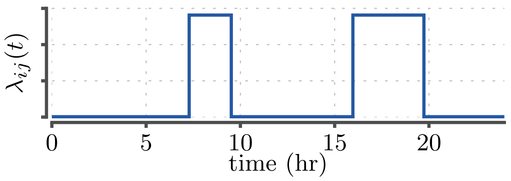
Fig. 8 Clusters found by our algorithm using London cycling hire usage dataset. Cluster 1, 2, 3, 4, and 5 are denoted by blue, yellow, green, pink, and red markings respectively (Color figure online)

	1	2	3	4	5
1	2.02	0.05	0.57	3.80	0.60
2	0.05	1.08	0.01	0.16	0.49
3	0.57	0.01	1.29	0.08	1.39
4	3.80	0.16	0.08	8.98	2.35
5	0.60	0.49	1.39	2.35	1.27

(a) Λ values for 1, 3, 5 segments

	1	2	3	4	5
1	7.63	0.35	2.63	21.10	2.50
2	0.35	3.77	0.03	1.05	2.10
3	2.63	0.03	6.48	0.28	7.50
4	21.10	1.05	0.28	63.30	8.25
5	2.50	2.10	7.50	8.25	4.47

(b) Λ values for 2, 4 segments



(c) segmentation

Fig. 9 Discovered parameters Λ and the segmentation for London cycling data ($R = 5$, $K = 5$, and $H = 2$). The Λ values are scaled by 10^6 for the sake of presentation

Fig. 8 in *blue* color interacts with its geographically overlapping cluster 1 which is shown in *pink* color more frequently while keeping its intra-cluster interactions more busier with compared to other clusters.

On the other hand, we can observe in Fig. 9 that $\lambda_{12}(t)$, $\lambda_{23}(t)$, and $\lambda_{35}(t)$ record the least levels of interactions ticking. To justify our observed result, we see the geographic closeness between cluster 1 and 2, cluster 2 and 3, and cluster 3 and 5. It is evident in Fig. 8 that above cluster pairs are not clustered together as its neighbouring counterparts thus the interaction levels have become too low when compared to other inter-cluster interactions.

6 Concluding remarks

In this paper we introduced a problem of finding recurrent sources in temporal network: we introduced a stochastic block model with recurrent segments. We proposed two variants of this problem where group membership of a node was fixed over the course of timeline and group memberships were varying from segment to segment.

We showed that finding optimal groups and recurrent segmentation in both variants was **NP**-hard. Therefore, to find good solutions we introduced an iterative algorithm by considering 3 subproblems, where we optimize groups, model parameters, and segmentation in turn while keeping the remaining structures fixed. We demonstrate how each subproblem can be optimized in $\mathcal{O}(m)$ time. Here, the key step is to use SMAWK algorithm for solving the segmentation. This leads to a computational complexity of $\mathcal{O}\left(KHm + Rn + R^2H\right)$ for a single iteration. We show experimentally that the number of iterations is low, and that the algorithm can find the ground truth using synthetic datasets.

The paper introduces several interesting directions: Gionis and Mannila (2003) considered several approximation algorithms but they cannot be applied directly for our problem because our optimization function is different. Adopting these algorithms in order to obtain an approximation guarantee is an interesting challenge. We used a simple heuristic to optimize the groups. We chose this approach due to its computational complexity. Experimenting with more sophisticated but slower methods for discovering block models, such as methods discussed in Abbe (2017), provides a fruitful line of future work.

Appendix A: Proofs

Proof of Proposition 1 Assume that we are given an instance of MaxCut, that is, a static graph H with n nodes and $m \geq n$ edges. Define

$$r = 2^6(n+1)^2 \quad \text{and} \quad \alpha = \max(2^{20}m^2, r^2) \quad .$$

The temporal graph G consists of two copies of H: we will denote the nodes of the copies with $U = u_1, \ldots, u_n$ and $V = v_1, \ldots, v_n$. We connect the nodes in U (and V) to match the edges in H at timestamp 1. We connect the corresponding nodes u_i and v_i with $n\alpha$ cross edges at timestamp 0. We also add two sets of r nodes, which we will denote by X and Y, and connect each node pair (x, y), where $x \in X$ and $y \in Y$, with α edges at timestamp 0.

We set $H = K = 1$, which forces the segmentation to be a single segment [0, 1].

Set $R = 2$. Let $\mathcal{P} = \left\{P_1, P_2\right\}$ be the optimal solution, and let Λ be its parameters.

We will prove in Lemma A.1 that $X \subseteq P_1$ and $Y \subseteq P_2$ or $X \subseteq P_2$ and $Y \subseteq P_1$. Moreover, $u_i \in P_1$ implies that $v_i \in P_2$ and $v_i \in P_1$ implies $u_i \in P_2$.

This immediately implies that $\lambda_{11} = \lambda_{22}$. Moreover, since $r > n$, we have $\lambda_{12} > \alpha/4 > 1 \geq \lambda_{11}$.

Let us define $C_i = U \cap P_i$ and $D_i = V \cap P_i$. Write x to be the number of cross edges between C_1 and C_2.

Let $C_1' \cup C_2' = U$ be the maximum cut, and let $D_1' \cup D_2' = V$ be the corresponding cut in V. Define $\mathcal{P}' = \left\{X \cup C_1' \cup D_2', Y \cup C_2' \cup D_1'\right\}$. Write x' to be the number of cross edges between C_1' and C_2'. By optimality $x' \geq x$.

The log-likelihood of $\mathcal{P}'$ is

$$\begin{aligned}\ell\left(\mathcal{P}'\right) &\geq \ell\left(\mathcal{P}', \Lambda\right) \\ &= (2m - 2x')\log\lambda_{11} + 2x'\log\lambda_{12} + Z \\ &\geq (2m - 2x)\log\lambda_{11} + 2x\log\lambda_{12} + Z \\ &= \ell(\mathcal{P}) \geq \ell\left(\mathcal{P}'\right),\end{aligned}$$

where

$$Z = \alpha(n^2 + r^2)\log\lambda_{12} - 2m - \alpha(n^2 + r^2).$$

We have shown that $x' = x$, proving the NP-hardness of finding $\mathcal{P}$ with the optimal likelihood. □

Lemma A.1 *Let $\mathcal{P}$ be the partition as defined in the proof of Proposition 1. Then $X \subseteq P_1$ and $Y \subseteq P_2$ or $X \subseteq P_2$ and $Y \subseteq P_1$. Moreover, $u_i \in P_1$ implies that $v_i \in P_2$ and $v_i \in P_1$ implies $u_i \in P_2$.*

Proof To prove the lemma we will need several counters: let us define $a_i = \left|P_i \cap X\right|$, $b_i = \left|P_i \cap Y\right|$, $c_i = \left|P_i \cap U\right|$, and $d_i = \left|P_i \cap V\right|$. We also write $x_i = a_i + b_i$, $y_i = c_i + d_i$ and $z_i = x_i + y_i$.

Define k_{ij} to be the number of cross edges between U and V in (i, j)th group of $\mathcal{P}$. Similarly, let m_{ij} to be the number of edges in U and V (that is, the cross edges are excluded) in (i, j)th group of $\mathcal{P}$. Note that $k_{12} + k_{11} + k_{22} = n$ and $m_{12} + m_{11} + m_{22} = 2m$.

The parameters for $\mathcal{P}$ are

$$\lambda_{12} = \frac{\alpha a_1 b_2 + \alpha a_2 b_1 + \alpha n k_{12} + m_{12}}{z_1 z_2},$$
$$\lambda_{11} = \frac{\alpha a_1 b_1 + \alpha n k_{11} + m_{11}}{\binom{z_1}{2}},$$

and

$$\lambda_{22} = \frac{\alpha a_2 b_2 + \alpha n k_{22} + m_{22}}{\binom{z_2}{2}}.$$

Let us define $\mathcal{P}' = \{X \cup U, Y \cup V\}$. Note that the parameters for $\mathcal{P}'$ are equal to

$$\lambda_{12}' = \frac{\alpha n^2 + \alpha r^2}{(r+n)^2} \quad \text{and} \quad \lambda_{11}' = \lambda_{22}' = \frac{m}{\binom{n+r}{2}}.$$

To prove the claim we will assume that $a_1b_1 + a_2b_2 > 0$ or $k_{12} < n$, and show that $\ell(\mathcal{P}') > \ell(\mathcal{P})$ which is a contradiction.

First, note that we can write the score difference as

$$\ell(\mathcal{P}') - \ell(\mathcal{P}) = A + B + C,$$

where

$$A = \alpha(a_1b_2 + a_2b_1 + nk_{12}) \log \frac{\lambda'_{12}}{\lambda_{12}},$$
$$B = \alpha(a_1b_1 + nk_{11}) \log \frac{\lambda'_{12}}{\lambda_{11}} + \alpha(a_2b_2 + nk_{22}) \log \frac{\lambda'_{12}}{\lambda_{22}},$$

and

$$C = m_{11} \log \frac{\lambda'_{11}}{\lambda_{11}} + m_{22} \log \frac{\lambda'_{11}}{\lambda_{22}} + m_{12} \log \frac{\lambda'_{11}}{\lambda_{12}}. \quad (1)$$

We claim that

$$A \geq 0, \quad B > \alpha 2^{-7}, \quad \text{and} \quad C \geq 2m \log \frac{1}{4\alpha n r^2}.$$

This proves the lemma since

$$\begin{aligned} B + C &> \alpha 2^{-7} - 2m \log 4\alpha n r^2 \\ &\geq \alpha 2^{-7} - 8m \log \alpha \\ &\geq \alpha 2^{-7} - 8m\sqrt{\alpha} \\ &= \sqrt{\alpha}\left(\sqrt{\alpha}2^{-7} - 8m\right) \geq 0. \end{aligned}$$

We will first bound C. Since we may have at most αn edges per node pair, we have $\lambda_{11}, \lambda_{12}, \lambda_{22} \leq \alpha n$. Moreover, since $r > n$, we have $\lambda'_{11} \geq (r + n)^{-2} \geq r^{-2}/4$. The bound follows from Eq. (1).

Next we will bound B. Assume that $x_1, x_2 \geq 2n$. Our next step is to upper bound λ_{11} and λ_{22}. In order to do this, first note that since $m_{11} \leq \alpha/2$ and $k_{11} \leq y_1$, we have

$$\frac{m_{11} + \alpha n k_{11}}{\binom{y_1}{2} + y_1 x_1} \leq \alpha \frac{1/2 + n y_1}{\binom{y_1}{2} + 2n y_1} \leq \alpha \frac{1/2 + n y_1}{1 + 2n y_1} = \alpha/2.$$

In addition, since $x_1 \geq 3$, we have

$$\alpha \frac{a_1 b_1}{\binom{x_1}{2}} \leq 2\alpha/3 \quad .$$

We can combine the two bounds, leading to

$$\lambda_{11} = \frac{\alpha a_1 b_1 + m_{11} + \alpha n k_{11}}{\binom{z_1}{2}} = \frac{a_1 b_1 + m_{11} + \alpha n k_{11}}{\binom{x_1}{2} + \binom{y_1}{2} + y_1 x_1} \leq 2\alpha/3.$$

The same bound holds for λ_{22}.

Since $r \geq 4n$, we have $\lambda'_{12} \geq 17\alpha/25$. Thus,

$$B \geq \alpha(a_1b_1 + a_2b_2 + nk_{11} + nk_{22}) \log \frac{17 \times 3}{25 \times 2} > \alpha 2^{-7}.$$

Assume now that $x_1 < 2n$. Then $a_2 + b_2 = x_2 > 2r - 2n > 1.5r$. Since $a_2, b_2 \leq r$, we must have $a_2, b_2 \geq r/2$. Moreover, using the previous arguments, we have

$$\log \frac{\lambda'_{12}}{\lambda_{22}} \geq 2^{-7} \quad \text{and} \quad \log \frac{\lambda'_{12}}{\lambda_{11}} \geq \log \frac{17}{25n}.$$

Consequently,

$$\begin{aligned} B &= \alpha(a_2b_2 + nk_{22}) \log \frac{\lambda'_{12}}{\lambda_{22}} + \alpha(a_1b_1 + nk_{11}) \log \frac{\lambda'_{12}}{\lambda_{11}} \\ &\geq \alpha\left(\frac{r^2}{2^9} - 4n^2 \log \frac{25n}{17}\right) \\ &\geq \alpha(8(n+1)^4 - 4n^2 \log 2n) \\ &\geq \alpha(8(n+1)^4 - 8n^4) \geq \alpha > \alpha 2^{-7}. \end{aligned}$$

Finally we will bound A by showing that $\lambda_{12} \leq \lambda'_{12}$. Assume for simplicity that $x_1 \leq x_2$. Let us define

$$N = a_1b_2 + a_2b_1 + 1 + nk_{12} \quad .$$

Assume that $z_1, z_2 \geq r$. We claim that $N \leq r^2 + (z_1 - r)(z_2 - r)$, which leads to

$$\lambda_{12} \leq \alpha \frac{N}{z_1z_2} \leq \alpha \frac{r^2 + (z_1 - r)(z_2 - r)}{z_1z_2} = \alpha - \alpha \frac{2nr}{z_1z_2}.$$

Here the first inequality holds since $m_{12} \leq 2m \leq \alpha$. The right hand side achieves its maximum when z_1z_2 is maximized, that is, $z_1 = z_2 = n + r$. In such case, the upper bound is equal to λ'_{12}.

To prove the claim, first note that

$$nk_{12} \leq \max(y_1, y_2) \min(y_1, y_2) = y_1y_2 \tag{2}$$

with the equality holding if and only if $k_{12} = y_1 = y_2 = n$.

Assume that $x_1 = x_2 = r$. If $a_1b_1 + a_2b_2 > 0$, then

$$N = r^2 - a_1b_1 + a_2b_2 + 1 + nk_{12} \leq r^2 + y_1y_2 = r^2 + (z_2 - r)(z_1 - r).$$

Assume $a_1b_1 + a_2b_2 = 0$. Then $k_{12} < n$, and the inequality is strict in Eq. (2). Consequently,

$$N = r^2 + nk_{12} + 1 \leq r^2 + y_1y_2 = r^2 + (z_2 - r)(z_1 - r)$$

Assume now that $x_1 \leq x_2 - 1$. Since a node in $(X \cup Y) \cap P_1$ is connected to r nodes, we must have $a_1b_2 + a_2b_1 \leq x_1r$. Due to the assumption, $x_1 \leq r - 1$ and $x_2 \geq r + 1$, which leads to

$$\begin{aligned} k_{12} &\leq \min(y_1, y_2) \\ &= \min(z_1 - x_1, z_2 - x_2) \\ &\leq \min(z_1 - x_1, z_2 - r) \end{aligned}$$

and

$$\begin{aligned} N &\leq x_1 r + 1 + n \min(z_1 - x_1, z_2 - r) \\ &\leq (r-1)r + 1 + n \min(z_1 - r + 1, z_2 - r) \\ &\leq r^2 + (1 + n - r) + n \min(z_1 - r, z_2 - r) \\ &\leq r^2 + \max(z_1 - r, z_2 - r) \min(z_1 - r, z_2 - r) \\ &= r^2 + (z_2 - r)(z_1 - r). \end{aligned}$$

As a final case assume that $z_1 < r$. If $x_1 = z_1$, then

$$\lambda_{12} \leq \alpha \frac{r z_1}{z_1 z_2} \leq \alpha \frac{r}{r + 2n} = \alpha \frac{r^2}{r^2 + 2nr} \leq \alpha \frac{r^2 + n^2}{r^2 + 2nr + n^2} = \lambda'_{12}.$$

If $x_1 < z_1$, then

$$N \leq x_1 r + 1 + n(z_1 - x_1) \leq z_1 r$$

and again

$$\lambda_{12} \leq \alpha \frac{N}{z_1 z_2} \leq \frac{z r_1}{z_1 z_2} \leq \lambda'_{12}.$$

The case for $z_2 < r$ is symmetrical.

We have now proven our claim, and thus proved that $\lambda_{12} \leq \lambda'_{12}$. Consequently, $A \geq 0$. □

Proof of Proposition 3 To prove **NP**-hardness we will reduce the MaxCut problem, where we are asked to partition graph into 2 subgraphs and maximize cross-edges.

Assume that we given a static graph H. We will use H as our temporal graph G by setting the edges to the same timestamp, say t. We also set $H = K = 1$, and use $R = 2$ groups. We also set the segmentation $\mathcal{T} = [t, t]$.

Select two values $\alpha < \beta$ and set the parameters $\lambda_{11} = \lambda_{22} = \alpha$ and $\lambda_{12} = \beta$.

Let P_1, P_2 be a partition of the nodes and let x be the number of the inner edges, that is, edges (u, v, t) with $u, v \in P_1$ or $u, v \in P_2$. Note that $m - x$ is the number of cross edges.

The log-likelihood is then equal to

$$\ell(\mathcal{P}, \mathcal{T}, g, \Lambda) = x \log \alpha + (m - x) \log \beta,$$

which is maximized when $m - x$ is maximized since $\beta > \alpha$. Since $m - x$ is the number of cross-edges, this completes the proof. □

Proof of Proposition 5 Assume that we given an instance of 3-Matching, that is, a domain X of size n, where n is divisible by 3, and a collection $\mathcal{S}$ of m sets such that $S \subseteq X$ and $|S| = 3$ for each $S \in \mathcal{S}$. The problem whether there is a disjoint subcollection in $\mathcal{S}$ covering X is known to be **NP**-complete.

Let $\mathcal{T} = \{S \subseteq X \mid |S| = 3, S \notin \mathcal{S}\}$ be the complement collection of $\mathcal{S}$. For each $i \leq j \leq n$, define c_{ij} to be the number of sets in S containing i and j,

$$c_{ij} = |\{S \in \mathcal{S} \mid \{i,j\} \subset S\}|.$$

To construct the dynamic graph G we will use 5 sets of nodes, namely $\{u\}, A, B, C, D$. The first set consists only of one node u. Every edge will be adjacent to u. The second set A contains as many nodes as there are sets in $\mathcal{T}$. For each $i \in T_j \in \mathcal{T}$, we add an edge (u, a_j) at timestamp i. The third set B contains $\sum_{i<j} c_{ij}$ nodes which we divide further into $n(n-1)/2$ sets B_{ij} with $\left|B_{ij}\right| = c_{ij}$. For each $i < j$ we connect nodes in B_{ij} with u at timestamp i and at timestamp j. The fourth set C contains $n(m - c_{ii})$ nodes which we divide further into n sets C_i with $\left|C_i\right| = m - c_{ii}$. For each $i \leq n$ we connect nodes in C_i with u at timestamp i. The fifth set D contains nw nodes, where $w = 24n^6$, which we divide further into n sets D_i with $\left|D_i\right| = w$. For each $i \leq n$ we connect w nodes with u at timestamp i.

We will set $K = n$ and $H = n/3$. We set R to be the number of nodes and set the partition $\mathcal{P}$ to be the partition where each node is contained in its own group. We require for a segmentation to start from 0. Since there are only n timestamps, the segmentation consists of n segments of form $(i-1, i]$ or $[0, 1]$.

Let g be the optimal grouping of segments and let Λ be its parameters. We first claim that g groups timestamps into groups of 3. To prove this assume that there is a group of size $y = 1, 2$. Then there is another group with a size of $x \geq 6 - y$. Let g' be a mapping where we move $3 - y$ timestamps from the larger group to the smaller group and let Λ' be the new optimal parameters.

The number of edges adjacent to A, B, and C can be bound by $3n^3 + 2n^2m + nm \leq 6n^5$. Moreover, the non-zero parameters can be bound by $\lambda' \geq 1/n$ and $\lambda \leq 1$. Consequently, the score difference can be bound by

$$\ell\left(g'\right) - \ell(g) \geq 6n^5(\log 1/n - \log 1) + Z(x) \geq 6n^6 + Z(x),$$

where $Z(x)$ is equal to

$$w\left((x-3+y)\log\frac{1}{x-3+y} + 3\log\frac{1}{3} - x\log\frac{1}{x} - y\log\frac{1}{y}\right).$$

The derivative of $Z(x)$ with respect to x is equal to $\log(x/(x-3+y)) > 0$, that is, $Z(x)$ is at smallest when $x = 6 - y$. A direct calculation shows that $Z(x)$ is the smallest when $y = 2$, leading to

$$\begin{aligned}\ell\left(g'\right) - \ell(g) &\geq -6n^6 + w\left(6\log\frac{1}{3} - 4\log\frac{1}{4} - 2\log\frac{1}{2}\right)\\ &> -6n^6 + w/4 = 0.\end{aligned}$$

In summary, $\ell\left(g'\right) > \ell(g)$ which is a contradiction. Thus, g groups of segments to size of at least 3. Since $K = 3H$, the groups are exactly of size 3.

Our next step is to calculate the impact of a single group to the score. In order to do that first note that for any $i < j$ there are

$$\binom{n}{3} - c_{ij} + c_{ij} = \binom{n}{3}$$

edges joining the same nodes at timestamp i and timestamp j. Similarly, there are

$$\binom{n}{3} - c_{ii} + \left(\sum_{i<j} c_{ij}\right) + m - c_{ii} + w = \binom{n}{3} + m + w$$

edges adjacent to node i.

Consider a set S of size 3 induced by g. Assume that $S \notin \mathcal{T}$. Then there are $3\binom{n}{3}$ parameters in Λ associated with the group with value 2/3, and $3(m+w)$ parameters with value 1/3. The remaining parameters are 0. Consequently, the impact to the score is equivalent to

$$\alpha = 6\binom{n}{3}\log 2/3 + 3(m+w)\log 1/3.$$

Assume that $S \in \mathcal{T}$. Then using exclusion-inclusion principle, there are $3\binom{n}{3} - 3$ parameters with value 2/3, $3(m+w)+3$ parameters with value 1/3, one parameter with value 3/3 and the remaining parameters are 0. Consequently, the impact to the score is equivalent to

$$\beta = 6\binom{n}{3}\log 2/3 - 6\log 2/3 + 3(1+m+w)\log 1/3 \quad .$$

We immediately see that

$$\alpha - \beta = 6\log 2/3 - 3\log 1/3 = 3\log 4/3 > 0 \quad .$$

Let k be the number of groups induced by g that are in $\mathcal{S}$. Then the score is equal to

$$\ell(g) = k\alpha + (H-k)\beta - |E(G)|.$$

Since $\alpha > \beta$, there is a disjoint subcollection in $\mathcal{S}$ covering X if and only if $\ell(g) = H\alpha - |E(G)|$. □

Appendix B: Detailed pseudo-code for FindSegments

Algorithm 5 Algorithm FindSegments $(\mathcal{P}, \Lambda)$ for finding optimal segmentation for fixed groups $\mathcal{P}$ and parameters Λ

```
p(v) ← group index of v;
f[e, h] ← 0 for each e ∈ E and h = 1, ..., H;
t_min ← min {t | (u, v, t) ∈ E};
foreach h = 1, ..., H do
    α ← Σ_{i≤j} |P_i × P_j| λ_{ijh} ;
    β ← 0;
    foreach e = (u, v, t) ∈ E in chrono. order do
        i ← p(u);
        j ← p(v);
        β ← β + log λ_{ijh};
        f[e, h] ← β − α(t − t_min);
foreach e ∈ E and k = 1, ..., K do
    o[e, k] ← r[e, k] ← q[e, k] ← 0;
foreach e ∈ E do
    o[e, 1] ← max_h f[e, h];
    r[e, 1] ← arg max_h f[e, h];
foreach k = 2, ..., K do
    z[e, h] ← 0 for each e ∈ E and h = 1, ..., H;
    x(s, e; h) ← o[s, k − 1] + f[e, h] − f[s, h];
    foreach h = 1, ..., H do
        z[e, h] ← arg max_s x(s, e; h) for each e ∈ E use SMAWK;
    r[e, k] ← arg max_h x(z[e, h], e; h);
    o[e, k] ← max_h x(z[e, h], e; h);
    q[e, k] ← z[e, r[e, k]];
e ← last edge in E;
foreach k = K, ..., 2 do
    s ← q[e, k];
    T_k ← (t(s), t(e)];
    g(k) ← r[e, k];
    e ← s;
T_1 ← [t_min, t(e)];
g(1) ← r[e, 1];
return (T_1, ..., T_K), g;
```

Author contributions Nikolaj Tatti formulated the problems. Chamalee Wickrama Arachchi implemented the algorithms and conducted the experiments. Both authors wrote the manuscript.

Funding Open Access funding provided by University of Helsinki (including Helsinki University Central Hospital). This research is supported by the Academy of Finland projects MALSOME (343045).

Availability of data and material Publicly available datasets were used. See Sect. 5 for the links.

Code availability https://version.helsinki.fi/dacs/.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

Abbe, E. (2017). Community detection and stochastic block models: recent developments. *JMLR, 18*(1), 6446–6531.

Aggarwal, A., Klawe, M., Moran, S., Shor, P., & Wilber, R. (1987). Geometric applications of a matrix-searching algorithm. *Algorithmica, 2*(1–4), 195–208.

Anderson, C. J., Wasserman, S., & Faust, K. (1992). Building stochastic blockmodels. *Social Networks, 14*(1), 137–161.

Arachchi, C. W., & Tatti, N. (2022). Recurrent segmentation meets block models in temporal networks. In *Discovery Science* (pp. 445–459). Springer.

Arockiasamy A., Gionis A., & Tatti N. (2016). A combinatorial approach to role discovery. In *ICDM* (pp. 787–792).

Bellman, R. (1961). On the approximation of curves by line segments using dynamic programming. *Communications of the ACM, 4*(6), 284–284.

Corneli, M., Latouche, P., & Rossi, F. (2018). Multiple change points detection and clustering in dynamic networks. *Statistics and Computing, 28*(5), 989–1007.

Fleischer, R., Golin, M. J., & Zhang, Y. (2006). Online maintenance of k-medians and k-covers on a line. *Algorithmica, 45*(4), 549–567.

Galbrun, E., Cellier, P., Tatti, N., Termier, A., & Crémilleux, B. (2019) Mining periodic patterns with a MDL criterion. In *ECML PKDD* (pp. 535–551).

Galil, Z., & Park, K. (1990). A linear-time algorithm for concave one-dimensional dynamic programming. *IPL, 33*(6), 309–311.

Gionis, A., & Mannila, H. (2003). Finding recurrent sources in sequences. In *RECOMB* (pp. 123–130).

Guha, S., Koudas, N., & Shim, K. (2006). Approximation and streaming algorithms for histogram construction problems. *TODS, 31*(1), 396–438.

Han, J., Dong, G., & Yin, Y. (1999) Efficient mining of partial periodic patterns in time series database. In *ICDE* (pp. 106–115).

Han, J., Gong, W., & Yin, Y. (1998). Mining segment-wise periodic patterns in time-related databases. In *KDD*.

Hassin, R., & Tamir, A. (1991). Improved complexity bounds for location problems on the real line. *Operations Research Letters, 10*(7), 395–402.

Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., & Li, L. (2012). RolX: Structural role extraction & mining in large graphs. In *KDD* (pp. 1231–1239).

Holland, P. W., Laskey, K. B., & Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social Networks, 5*(2), 109–137.

Holme, P., & Saramäki, J. (2012). Temporal networks. *Physics Reports, 519*(3), 97–125.

Kostakis, O., Tatti, N., & Gionis, A. (2017). Discovering recurring activity in temporal networks. *DMKD, 31*(6), 1840–1871.

Lee, C., & Wilkinson D. J. (2019). A review of stochastic block models and extensions for graph clustering. *Applied Network Science,4*(122).

Ma, S., & Hellerstein, J. L. (2001). Mining partially periodic event patterns with unknown periods. In *ICDE* (pp. 205–214).

Matias, C., Rebafka, T., & Villers, F. (2018) Estimation and clustering in a semiparametric Poisson process stochastic block model for longitudinal networks. *Biometrika,105*(3).

Matias, C., & Miele, V. (2017). Statistical clustering of temporal networks through a dynamic stochastic block model. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79*(4), 1119–1141.

Ozden, B., Ramaswamy, S., & Silberschatz, A. (1998). Cyclic association rules. In *ICDE* (pp. 412–421).

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association, 66*(336), 846–850.

Rozenshtein, P., Bonchi, F., Gionis, A., Sozio, M., & Tatti, N. (2020). Finding events in temporal networks: Segmentation meets densest subgraph discovery. *KAIS, 62*(4), 1611–1639.

Tatti, N. (2019). Strongly polynomial efficient approximation scheme for segmentation. *Information Processing Letters, 142*, 1–8.

Xu, K. S., & Hero, A. O. (2014). Dynamic stochastic blockmodels for time-evolving social networks. *JSTSP, 8*(4), 552–562.

Yang, T., Chi, Y., Zhu, S., Gong, Y., & Jin, R. (2011). Detecting communities and their evolutions in dynamic social networks—a Bayesian approach. *Machine Learning, 82*, 157–189.

Yang, J., Wang, W., & Yu, P. S. (2003). Mining asynchronous periodic patterns in time series data. *TKDE, 15*(3), 613–628.