Recurrent Segmentation Meets Block Models in Temporal Networks

Chamalee Wickrama Arachchi(✉) and Nikolaj Tatti

HIIT, University of Helsinki, Helsinki, Finland
{chamalee.wickramaarachch,nikolaj.tatti}@helsinki.fi

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. For example, social network activity may be heightened during certain hours of day. 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 this 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. To enforce the recurring activity we require that only $H < K$ different set of parameters can be used, that is, several, not necessarily consecutive, segments must share their parameters. We prove that the searching for optimal blocks and segmentation is an **NP**-hard problem. Consequently, we split the problem into 3 subproblems where we optimize blocks, model parameters, and segmentation in turn while keeping the remaining structures fixed. We propose an iterative algorithm that requires $\mathcal{O}\left(KHm + Rn + R^2H\right)$ 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.

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 [17].

Many temporal phenomena have recurrent or possibly cyclic behaviour. For example, social network activity may be heightened during certain hours of day.

P. Pascal and D. Ienco (Eds.): DS 2022, LNAI 13601, pp. 445–459, 2022.
https://doi.org/10.1007/978-3-031-18840-4_32

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. [6] 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 [10] in the context of segmenting sequences of real valued vectors.

In this paper we extend the model proposed by Corneli et al. [6] using the ideas proposed by Gionis and Mannila [10]. 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.

To obtain good solutions we rely on an iterative method by splitting the problem into three subproblems: (i) optimize blocks while keeping the remaining parameters fixed, (ii) optimize model parameters Λ while keeping the blocks and the segmentation fixed, (iii) optimize the segmentation while keeping the remaining parameters fixed. We approach the first subproblem by iteratively optimizing block 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 [2], leading to a computational complexity of $\mathcal{O}\left(mKH + HR^2\right)$.

In summary, we extend a model by Corneli et al. [6] to have recurring segments. We prove that the 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 problem 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 in Sect. 5 and conclude the paper with discussion in Sect. 6. The proofs are provided in Appendix[1].

[1] The appendix is available at https://arxiv.org/abs/2205.09862.

2 Preliminary Notation and Problem Definition

Assume a *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 blocks, and an edge (u, v) are modelled with a Bernoulli probability parameterized with q_{ij}, where i is the block of u and j is the block of v. Given a graph, the optimization problem is to cluster nodes into blocks so that the likelihood of the model is optimized. For the sake of variability we will use the words *block* and *group* 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 block of u and j is the block of v. Note that λ_{ij} does not depend on the time, so discovering optimal blocks is very similar to discovering blocks in a static model.

A natural extension of this model, proposed by Corneli et al. [6], 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

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

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 $\Lambda = \{\lambda_{ijk}\}$[2], the log-likelihood is equal to

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

This leads immediately to the problem considered by Corneli et al. [6].

Problem 1. ((K, R) model). 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

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

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 $H \leq K$. In other words, we are forcing that certain segments *share* their parameters. We do not know beforehand which segments should share the parameters.

We can express this constraint more formally by introducing a mapping $g : [K] \to [H]$ 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 $g : [K] \to [H]$, and a set of $HR(R+1)/2$ parameters $\Lambda = \{\lambda_{ijh}\}$, 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.

Problem 2. ((K, H, R) model). 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 $\ell(\mathcal{P}, \mathcal{T}, g, \Lambda)$.

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

Algorithm 1: Main loop of the algorithm

$\mathcal{P} \leftarrow$ random groups; $\Lambda \leftarrow$ random values;
$\mathcal{T}, g \leftarrow \text{FindSegments}(\mathcal{P}, \Lambda)$;
$\Lambda \leftarrow \text{UpdateLambda}(\mathcal{P}, \mathcal{T}, g)$;
while *convergence* **do**
  $\mathcal{P} \leftarrow \text{FindGroups}(\mathcal{P}, \Lambda, \mathcal{T}, g)$;
  $\Lambda \leftarrow \text{UpdateLambda}(\mathcal{P}, \mathcal{T}, g)$;
  $\mathcal{T}, g \leftarrow \text{FindSegments}(\mathcal{P}, \Lambda)$;
  $\Lambda \leftarrow \text{UpdateLambda}(\mathcal{P}, \mathcal{T}, g)$;

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 $\mathcal{O}\left(KHm + Rn + R^2H\right)$, which is linear in both the nodes and edges.

3.1 Iterative Approach

Unfortunately, finding optimal solution for our problem is **NP**-hard.

Proposition 1. *Problem 2 is **NP**-hard, even for $H = K = 1$ and $R = 2$.*

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

We split the original problem 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.

Next we will explain each step in details.

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

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 [1]. However, we resort to a simple greedy optimization due to its speed.

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

```
1  p(v) ← group index of v;
2  s(e) ← segment index of e;
3  d[h] ← Σ_{g(k)=h} Δ(T_k);
4  foreach v ∈ V do
5      b ← p(v);
6      c[j,h] ← array c_jh as defined in Proposition 3;
7      foreach a = 1,... R do
8          x[a] ← Σ_{h=1}^{H} λ_bah d[h] + Σ_{j=1}^{R} c[j,h] log λ_ajh − |P_j| λ_ajh d[h] ;
9      p(v) ← arg max_a x[a] (update P also);
10 return P;
```

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

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

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

The proposition leads to the pseudo-code given in Algorithm 2. The algorithm computes an array c and then uses Proposition 3 to compute the gain for each swap, and consequently to find the optimal gain.

Computing the array requires iterating over the adjacent edges, leading to $\mathcal{O}(|N(v)|)$ time, and computing the gains requires $\mathcal{O}(R^2 H)$ time. Consequently, the computational complexity for FindGroups is $\mathcal{O}(m + R^2 H n + K)$.

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

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

can be precomputed in $\mathcal{O}(RH)$ time and maintained in $\mathcal{O}(1)$ 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

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

where $\mathcal{S}_h = \{T_k \in \mathcal{T} \mid h = g(k)\}$ are the segments using the hth level. Updating the parameters requires $\mathcal{O}\left(m + R^2H + K\right)$ time.

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

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

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

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 [5] 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 [2]. 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 5. *Fix h. Then the function $x(s,e) = y(s,e;h) + o[s,k-1]$ is totally monotone.*

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 ← min {t | (u, v, t) ∈ E};
2  f[e, h] ← log-likelihood of a segment [t_min, t(e)] using parameters λ_{·h};
3  foreach e ∈ E do o[e, 1] ← max_h f[e, h]
4  foreach k = 2, ..., K do
5      x(s, e; h) ← o[s, k − 1] + f[e, h] − f[s, h];
6      foreach h = 1, ..., H do
7          z[e, h] ← arg max_s x(s, e; h) for each e ∈ E (use SMAWK);
8      o[e, k] ← max_h x(z[e, h], e; h) for each e ∈ E;
9      r[e, k] ← arg max_h x(z[e, h], e; h);
10     q[e, k] ← z[e, r[e, k]];
11 Use r and q to recover the optimal segmentation (T_1, ..., T_K) and the level
   mapping g ;
12 return (T_1, ..., 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.

FindSegments runs in $\mathcal{O}\left(mKH + HR^2\right)$ time since we need to call SMAWK $\mathcal{O}\left(HK\right)$ times.

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 [9]. Fleischer et al. [7], Hassin and Tamir [14] used this approach to solve dynamic program segmenting monotonic one-dimensional sequences with L_1 cost.

We fixed Λ because Proposition 4 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 4 is the main reason why we cannot use directly the ideas proposed by Corneli et al. [6] 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 $\mathcal{O}\left(Km^2R^2\right)$ time. However, if we consider the optimization problem as a minimization problem and shift the cost with a constant so that it is always positive, then using algorithms by Guha et al. [26], Tatti [11] we can obtain $(1+\epsilon)$-approximation with $\mathcal{O}\left(K^3 \log K \log m + K^3\epsilon^{-2} \log m\right)$ number of cost evaluations. Finding the optimal parameters and computing the cost of a single segment can be done in $\mathcal{O}\left(R^2\right)$ time with $\mathcal{O}\left(R^2 + m\right)$ time for precomputing. This leads to a total time of $\mathcal{O}\left(R^2(K^3 \log K \log m + K^3\epsilon^{-2} \log m) + m\right)$ for the special case of $K = H$.

4 Related Work

The closest related work is the paper by Corneli et al. [6] 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. [6]. 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 [10], 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 [10].

In this paper, we used stochastic block model (see [3,16], 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: Matias and Miele [29], Yang et al. [21] proposed an approach where the nodes can change block memberships over time. In a similar fashion, Xu and Hero [27] 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. [22] 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 [19].

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. [4]. Moreover, Henderson et al. [15] 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 [8,12,13,20,23,28]. 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. [18] 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. [25] 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.

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

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

Table 1. Dataset characteristics and results from the experiments. 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_1 is the normalized log-likelihood for the ground truth, G is the Rand index, LL_2 is the discovered normalized log-likelihood, I is the number of iterations, and CT is the computational time in seconds.

Dataset	n	m	L	K	H	LL_1	R	LL_2	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	0.95	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
Email-Eu-1	309	61 046	3	10	7			0.89	12	188 s
Email-Eu-2	162	46 772	4	8	7			0.87	9	177 s
MathOverflow	21 688	107 581	2	3	2			0.91	20	263 s
CollegeMsg	1 899	59 835	3	8	5			0.87	19	662 s
MOOC	7 047	411 749	2	3	2			0.81	6	208 s
Bitcoin	3 783	24 186	3	10	10			0.91	7	115 s
Santander	735	33 116	3	7	5			0.94	20	60 s

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, 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

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

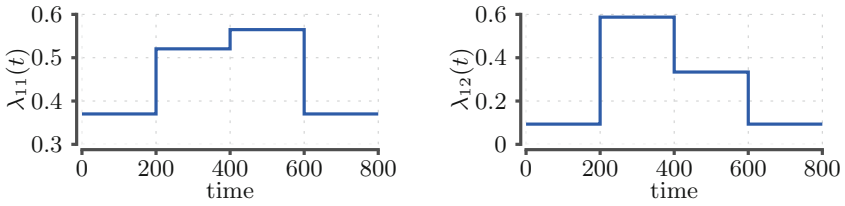

Fig. 1. Discovered parameters $\lambda_{11}(t)$, $\lambda_{12}(t)$ for the *Synthetic-4* dataset. Parameter $\lambda_{12}(t)$ implies the Poisson process parameter between group 1 and group 2 as a function of time.

that occurred on Sep 9, 2015 from the Santander bikes hires in London.[5] The sizes of these networks are given in Table 1.

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 [24] (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 $\ell(\mathcal{P}, \mathcal{T}, g, \Lambda)/\ell(\mathcal{P}', \mathcal{T}', g', \Lambda')$, 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.

As demonstrated in column LL_1 and column LL_2 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. 1. 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 1. From the last column CT, 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 Table 1.

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

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

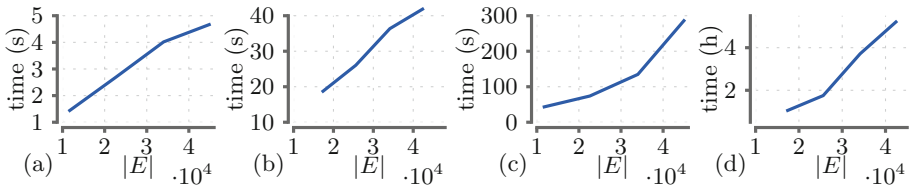

Fig. 2. Computational time as a function of number of temporal edges ($|E|$) for *Synthetic-large* (a, c) and *Santander-large* (b, d). This experiment was done with $R = 3$, $K = 5$, and $H = 3$ using SMAWK algorithm (a–b) and naive dynamic programming (c–d). The times are in seconds in (a–c) and in hours in (d).

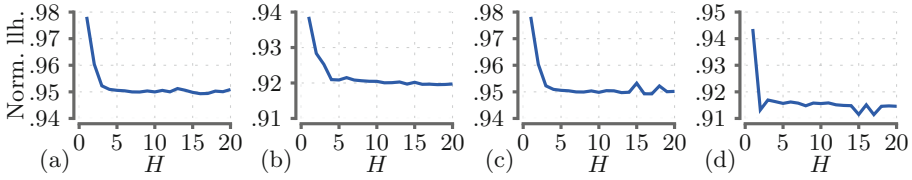

Fig. 3. Normalized log-likelihood as a function of number of levels (H) for the *Santander* dataset (a), *bitcoin* dataset (b), *Synthetic-5* dataset (c), and *Email-Eu-1* dataset (d). This experiment is done for $R = 2$, $K = 20$, and $H = 1, \ldots, 20$.

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

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. 2 that computational time has increased drastically when stock dynamic program algorithm is used. For example, a set of 34 143 edges required 3.7h for *Santander-large* dataset but only 36.36s 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. 3. 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. [6]. Interestingly enough, the log-likelihood values plateau for values of $H \ll K$ suggesting that existence of recurring segments in the displayed datasets.

6 Concluding Remarks

In this paper we introduced a problem of finding recurrent sources in temporal network: we introduced stochastic block model with recurrent segments.

We showed that finding optimal blocks and recurrent segmentation was an **NP**-hard problem. Therefore, to find good solutions we introduced an iterative algorithm by considering 3 subproblems, where we optimize blocks, 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 [10] 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 [1], provides a fruitful line of future work.

Acknowledgements. This research is supported by the Academy of Finland projects MALSOME (343045).

References

1. Abbe, E.: Community detection and stochastic block models: recent developments. JMLR **18**(1), 6446–6531 (2017)
2. Aggarwal, A., Klawe, M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica **2**(1–4), 195–208 (1987)
3. Anderson, C.J., Wasserman, S., Faust, K.: Building stochastic blockmodels. Soc. Netw. **14**(1), 137–161 (1992)
4. Arockiasamy, A., Gionis, A., Tatti, N.: A combinatorial approach to role discovery. In: ICDM, pp. 787–792 (2016)
5. Bellman, R.: On the approximation of curves by line segments using dynamic programming. Commun. ACM **4**(6), 284–284 (1961)
6. Corneli, M., Latouche, P., Rossi, F.: Multiple change points detection and clustering in dynamic networks. Stat. Comput. **28**(5), 989–1007 (2018)
7. Fleischer, R., Golin, M.J., Zhang, Y.: Online maintenance of k-medians and k-covers on a line. Algorithmica **45**(4), 549–567 (2006)

8. Galbrun, E., Cellier, P., Tatti, N., Termier, A., Crémilleux, B.: Mining periodic patterns with a MDL criterion. In: ECML PKDD, pp. 535–551 (2019)
9. Galil, Z., Park, K.: A linear-time algorithm for concave one-dimensional dynamic programming. IPL **33**(6), 309–311 (1990)
10. Gionis, A., Mannila, H.: Finding recurrent sources in sequences. In: RECOMB, pp. 123–130 (2003)
11. Guha, S., Koudas, N., Shim, K.: Approximation and streaming algorithms for histogram construction problems. TODS **31**(1), 396–438 (2006)
12. Han, J., Dong, G., Yin, Y.: Efficient mining of partial periodic patterns in time series database. In: ICDE, pp. 106–115 (1999)
13. Han, J., Gong, W., Yin, Y.: Mining segment-wise periodic patterns in time-related databases. In: KDD (1998)
14. Hassin, R., Tamir, A.: Improved complexity bounds for location problems on the real line. Oper. Res. Lett. **10**(7), 395–402 (1991)
15. Henderson, K., et al.: RolX: structural role extraction & mining in large graphs. In: KDD, pp. 1231–1239 (2012)
16. Holland, P.W., Laskey, K.B., Leinhardt, S.: Stochastic blockmodels: first steps. Soc. Netw. **5**(2), 109–137 (1983)
17. Holme, P., Saramäki, J.: Temporal networks. Phys. Rep. **519**(3), 97–125 (2012)
18. Kostakis, O., Tatti, N., Gionis, A.: Discovering recurring activity in temporal networks. DMKD **31**(6), 1840–1871 (2017)
19. Lee, C., Wilkinson, D.J.: A review of stochastic block models and extensions for graph clustering. Appl. Netw. Sci. **4**(122), 1–50 (2019)
20. Ma, S., Hellerstein, J.: Mining partially periodic event patterns with unknown periods. In: ICDE, pp. 205–214 (2001)
21. Matias, C., Miele, V.: Statistical clustering of temporal networks through a dynamic stochastic block model. J. Roy. Stat. Soc. Seri. B (Stat. Methodol.) **79**(4), 1119–1141 (2017)
22. Matias, C., Rebafka, T., Villers, F.: Estimation and clustering in a semiparametric poisson process stochastic block model for longitudinal networks. Biometrika **105**(3), 665–680 (2018)
23. Ozden, B., Ramaswamy, S., Silberschatz, A.: Cyclic association rules. In: ICDE, pp. 412–421 (1998)
24. Rand, W.M.: Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc. **66**(336), 846–850 (1971)
25. Rozenshtein, P., Bonchi, F., Gionis, A., Sozio, M., Tatti, N.: Finding events in temporal networks: segmentation meets densest subgraph discovery. KAIS **62**(4), 1611–1639 (2020)
26. Tatti, N.: Strongly polynomial efficient approximation scheme for segmentation. Inf. Process. Lett. **142**, 1–8 (2019)
27. Xu, K.S., Hero, A.O.: Dynamic stochastic blockmodels for time-evolving social networks. JSTSP **8**(4), 552–562 (2014)
28. Yang, J., Wang, W., Yu, P.: Mining asynchronous periodic patterns in time series data. TKDE **15**(3), 613–628 (2003)
29. Yang, T., Chi, Y., Zhu, S., Gong, Y., Jin, R.: Detecting communities and their evolutions in dynamic social networks–a Bayesian approach. Mach. Learn. **82**, 157–189 (2011)