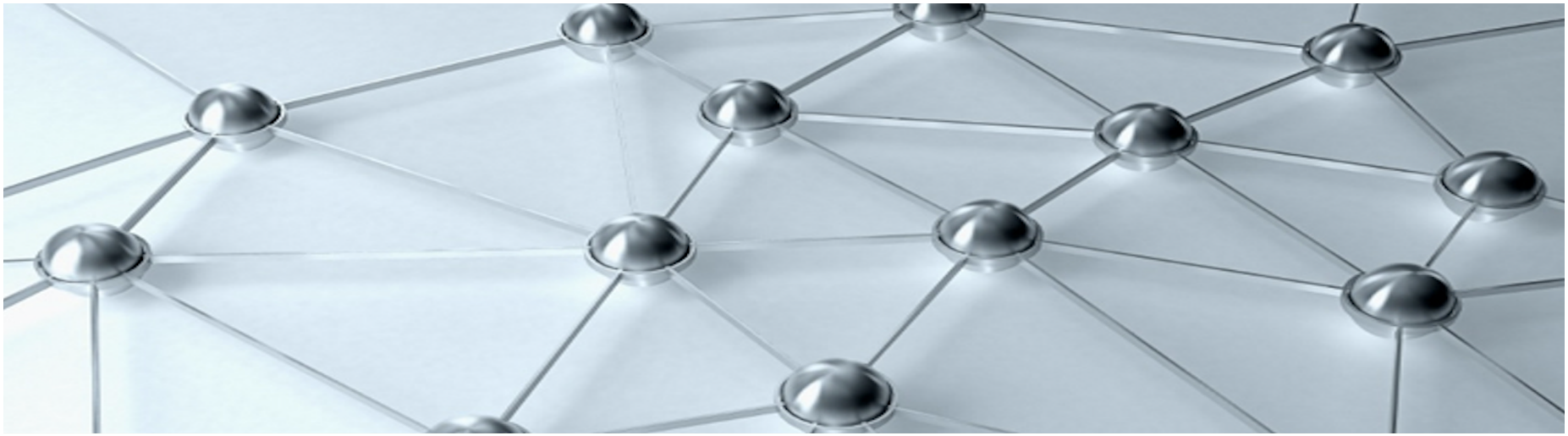

Recurrent segmentation meets block models in temporal networks

Chamalee Wickrama Arachchi, Nikolaj Tatti

University of Helsinki

Discovery Science : 2022

Outline

1. Introduction and Background.
2. Problem Formulation.
3. Algorithms.
4. Monotonicity, SMAWK.
5. Experiments.

Stochastic Block Model (SBM)

- SBM is a generative model for random graphs.
- It generates graphs containing communities such that connections between subsets of nodes are characterized with particular edge densities.
- Each node is assumed to belong to a particular group / cluster and the way nodes interact depends only on their respective cluster[1].

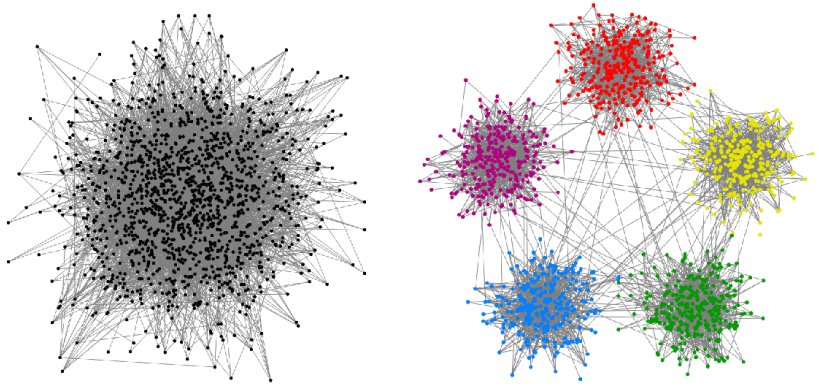

Figure: : The above two graphs are the same graph re-organized and drawn from the SBM model with 1000 vertices, 5 balanced communities, within-cluster probability of $.02$ and across-cluster probability of $.001$[2].

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

[2] Emmanuel Abbe, Community detection and stochastic block models: recent developments, 2017.

- Temporal Networks.
  Unlike static networks, real-world networks are often timestamped.
  Ex. Time stamped interactions between actors.
  - X1 sends an SMS to Y1 at time t1.
  - X2 sends an email to Y1 at time t2.
  - X4 likes/answers to Y3's post at time t3.
  - X1 phones to Y4 at time t4.
  - and also: citations (patents, articles), web links, tweets, moving objects, etc.
- Recurrent Activities.
  Many temporal networks exhibit cyclic or repeating behaviour.
  They might have *shared* parameters.
  Ex. Genomic sequence might be generated with h hidden sources such that some of the sources contribute for multiple segments.

Recurrent Segmentation Problem

Given

Sequence of edges with timestamps.

R = No of groups,
K = No of segments,
H = No of levels.

Goal

Partition nodes using SBM core.

Segment the timeline into K segments.

Segments should have H distinct levels.

Model

Each node is assumed to belong to a particular group; extension of SBM to dynamic networks.

Edges are modelled with a Non-homogeneous Poisson process.

Timeline of activities has the support for recurrent activities.

Maximize the likelihood.

Key questions

Is problem NP-hard?

Linear time algorithm?

Toy Example

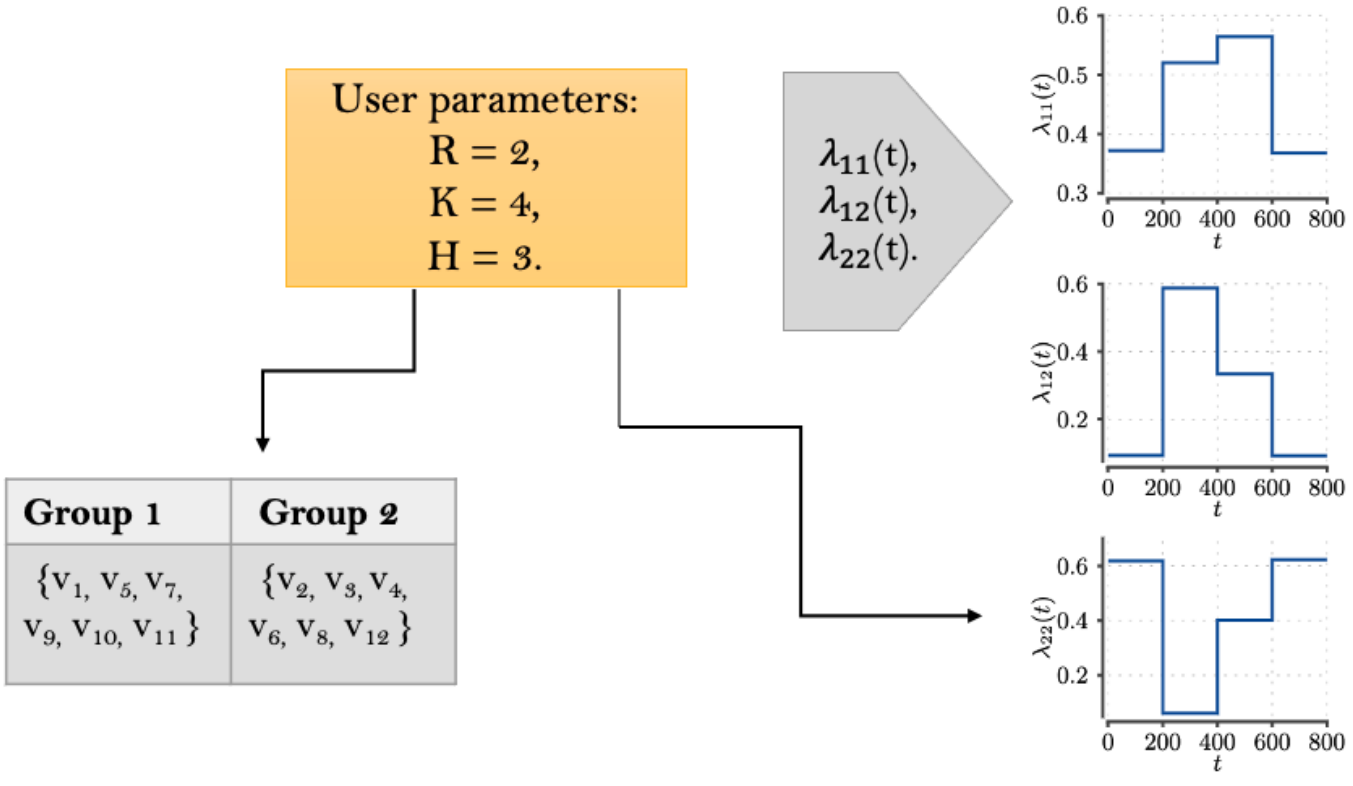

Figure: Toy Example

Problem Formulation

- R - Number of groups.
- K - Number of segments.
- H - Number of levels.
- P_i - Set of vertices in ith group.
- T_k - kth time segment.
- λ_{ijk} - Poisson parameter between ith group and jth group in kth segment.
- $\Lambda = \{\lambda_{ijk}\}$.

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

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

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

Algorithms

Searching for optimal groups and segments is computationally intractable. Therefore, we split the problem into three subproblems and alternately solve each subproblem.

- Optimize groups.
- Optimize Poisson parameters.
- Optimize segmentation.

Algorithm 1: Main loop of the algorithm

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

Subproblem 1 : Optimize Groups

- Greedy algorithm.
- Based on likelihood gain, each node is assigned to a group one at a time while group assignments of other nodes are fixed.
- 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.
- However, constant terms can be omitted when computing loglikelihood gain.
- For each node, we can consider only adjacent edges and then precompute edge count array $c[j, k]$ (jth group in kth segment) in $\mathcal{O}(m)$ time.
- The computational complexity for FindGroups is $\mathcal{O}\big(m + Rn + R^2H + K\big)$.

Subproblem 2 : Optimize Poisson Parameters

- The optimal parameters are equal to

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

- Updating the parameters requires $\mathcal{O}\big(m + R^2 H + K\big)$ time.

Subproblem 3 : Optimize Segmentation

Naive:

- Given a sequence of edges ordered by timestamp (Chronological order), use Dynamic Programming (DP) for segmentation.
- DP recurrence formula for sequence segmentation:

$$s[1, m; k] = \max_{k \leq j \leq m-1} s[1, j; k-1] + y(j+1, m; 1)$$

- Fill DP table (m x k) and store also maximizing j values such that we can recover the optimal segmentation by doing a trace back.
- Computational complexity is $\mathcal{O}\left(m^2 K\right)$.

Speedup:

- Use SMAWK to speed up the running time.
- Quadratic time to linear time using SMAWK.
- Due to totally monotonicity of matrix.

Totally Monotone Property

- We call that x is *totally monotone*,
  if $x(i_1, j_1) > x(i_1, j_2)$, then $x(i_2, j_1) > x(i_2, j_2)$ for any $i_1 < i_2$ and $j_1 < j_2$.

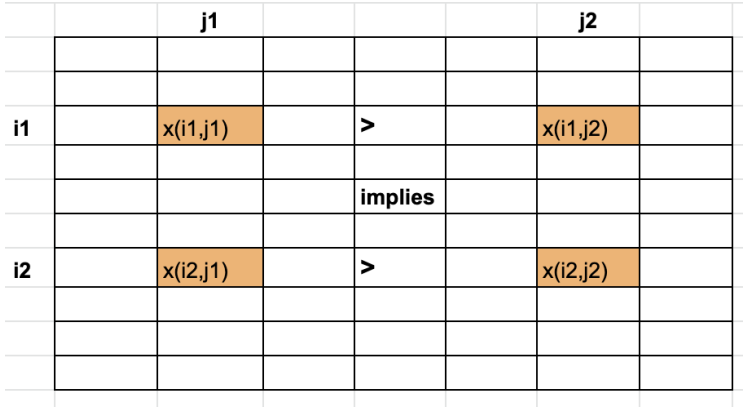

Figure: Totally monotone matrix

SMAWK

- Due to totally - monotonicity.
- SMAWK can find the row maxima in every row in $\mathcal{O}(m)$ time on nxm wide totally monotone matrix.
- SMAWK recursively computes the leftmost row maxima in every other row, and then fill in the remaining row maxima in $\mathcal{O}(m)$ time.

Segmentation Algorithm

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

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

Figure: FindSegments

SMAWK to Speedup DP

- $o[e, k]$: log-likelihood of k-segmentation up to e.
- $y(s, e; h)$: log-likelihood of a segment $(t(s), t(e)]$ using the hth level of Poisson parameters.

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

- The function $x(s, e) = o[s, k-1] + y(s, e; h)$ is totally monotone; for each fixed h.
- To compute x in $\mathcal{O}(1)$, we precompute $f[e, h]$ which is the log likelihood from epoch to e using corresponding h levels params; $y(s, e; h) = f[e, h] - f[s, h]$.
- SMAWK computes x by lazy fashion.
- Optimal segmentation is found after iterating over all h values.
- FindSegments runs in $\mathcal{O}\left(mKH + HR^2\right)$ time since we need to call SMAWK $\mathcal{O}(HK)$ times.

Experiments with Synthetic Datasets

Dataset	n	m	R	K	H	LL_1	G	LL_2	I	CT
Synthetic-1	50	76 332	2	2	2	0.95	1	0.94	2	2.81s
Synthetic-2	30	95 889	3	3	3	0.94	1	0.94	3	5.36s
Synthetic-3	20	65 056	3	3	3	0.97	1	0.97	3	3.91s
Synthetic-4	60	537 501	3	4	3	0.94	1	0.93	3	23.13s
Synthetic-5	10	33 475	2	10	5	0.91	1	0.91	4	10.27s

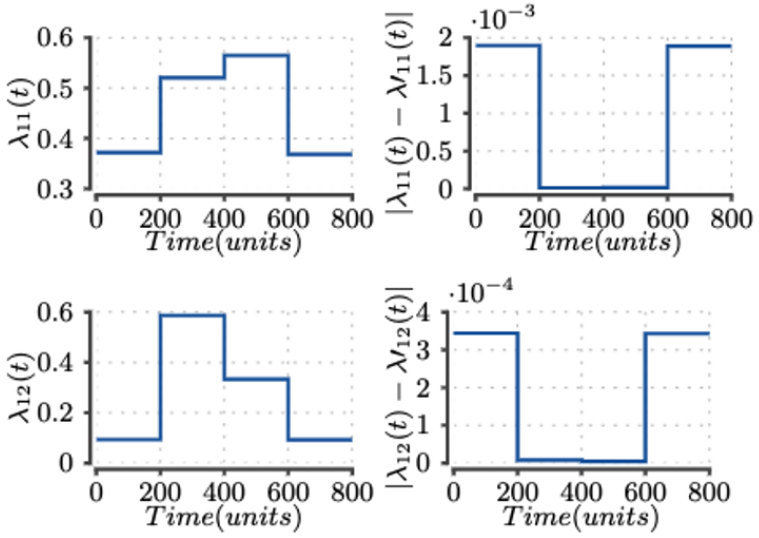

Figure: Table: Experiments for the synthetic datasets. Here, n is the number of nodes, m is the number of edges, R is the number of groups, K is the number of segments, H is the number of levels, LL_1 is the initial normalized log-likelihood, G is the Rand index, LL_2 is the discovered normalized log-likelihood, I is the number of iterations, and CT is the computational time., Plot: Right : True parameters $\lambda_{11}(t)$, $\lambda_{12}(t)$ (left) and their difference to the estimated estimated parameters $\lambda\prime_{11}(t)$, $\lambda\prime_{12}(t)$(right) for the *Synthetic-4* dataset. $\lambda_{12}(t)$ implies the Poisson process parameter between group 1 and group 2 as a function of time. This experiment is done with $n = 60$, $m = 537\,501$, $R = 3$, $K = 4$, and $H = 3$.

Experiments with Real-world Datasets

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

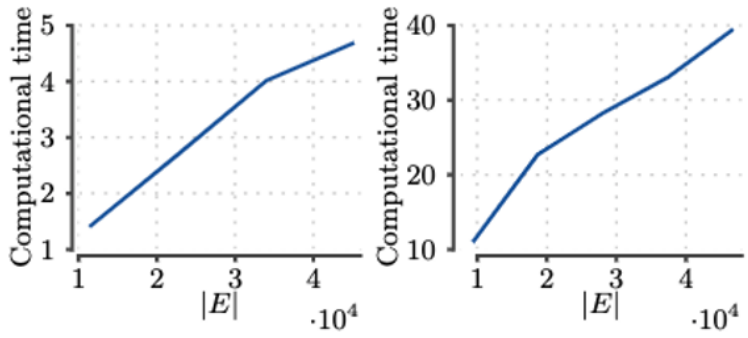

Figure: Table: Sizes and computational times for the real-time datasets. Here, n is the number of nodes, m is the number of edges, R is the number of groups, K is the number of segments, H is the number of levels, LL is the normalized log-likelihood, I is the number of iterations, and CT is the computational time in seconds., Plot : Computational time (in seconds) as a function of number of temporal edges ($|E|$) for the *Synthetic* datasets (left) and *Email-Eu-2* dataset (right). This experiment is done for $R = 3$, $K = 5$, and $H = 3$.

Likelihood vs Number of Levels (H)

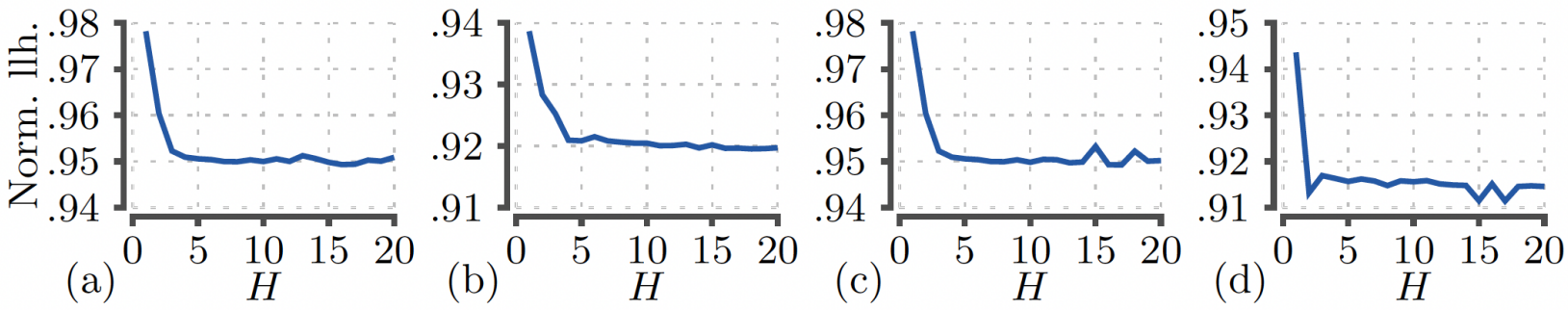

Figure: Normalized log-likelihood as a function of number of levels (H) for the *Santander* dataset (a), *bitcoin* dataset (b), *Synthetic-5* dataset (c), and *Email-Eu-1* dataset (d). This experiment is done for $R = 2$, $K = 20$, and $H = 1, \ldots, 20$.

Any Questions???

Thank you!!!