STATISTICAL CLUSTERING OF TEMPORAL NETWORKS THROUGH A DYNAMIC STOCHASTIC BLOCK MODEL USING OPTIMIZER

A PREPRINT

Dongwoo Won
Department of Mathematical Science
KAIST
Daejeon, South Korea
aiwwdw@kaist.ac.kr

Hyonho Chun
Department of Mathematical Science
KAIST
Daejeon, South Korea
hyonhochun@kaist.ac.kr

January 22, 2025

ABSTRACT

Keywords First keyword · Second keyword · More

1 Introduction

Statistical network analysis is one the major field of research in biology, sociology, internet, ect.

One of reaserch of statistical network analysis is clustering of temporal network.

There are a lot of effort to resolve clustering of temporal network. For example, SBM...

Deep learning is in the ascendant method in various field, such as image, language, even video. Almost of the deep learning focus on establishing the loss function and minimizing them.

In our work, we examine a novel yet underutilized method in the field of statistics to efficiently and swiftly solve the objective function. Specifically, we employ optimization techniques from deep learning to find the maximum of the objective function. Our setup is based on Matias and Miele [2017]. What is difference with previous research is that they made effort to solve the objective function with fixed point equation by their hands which is not easy to provide the exact formula instead suggesting the proportional value of number making the objective function maximum. However, We showed that Using optimizer is not bad even better performance in finding a solution on aspect of speed and accuracy. To implement this work, Python is used for ability to compute tensor computing with pytorch. Therefore, Providing more accurate objective function is enough to find the good clustering, using optimizer.

2 Setup and Notation

2.1 Model description

What we given is weighted interactions between N nodes over time interval in form of concatenated matrix $\mathbf{Y} = (Y^t)_{1\leq t\leq T}$ where T is the number of time points. For each time point $t \in \{1, ..., T\}$, $Y^t = (Y^t_{ij})_{1\leq i\neq j\leq N}$ is adjacency matrix contained the real number measured the interactions between each nodes $i, j \in \{1, \ldots, N\}^2$. In our model, we consider undirected random graphs without self-loops, so that Y^t is a symmetric matrix with no diagonal elements.

We assume that each node is assigned one of the Q latent numbers in each time, and the assigned number indicates the clustering number. This is represented by each $\mathbf{Z} = (Z^t_i)_{1\leq t\leq T, 1\leq i\leq N}$, which varies depending on the time and the nodes where $Z^t_i \in \{1, ..., Q\}$ or latent variable in each time and node represented in random vector $Z^t_i = (Z^t_{i1}, \ldots, Z^t_{iQ}) \in \{0, 1\}^Q$ constrained to $\sum_q Z^t_{iq} = 1$. We will use this notation to express the equation.

The process of model is based on Dynamic Stochastic Block Model. The latent variable are independent and identically distributed across nodes. Latent variable on each nodes is irreducible, aperiodic stationary markov chain with transition matrix $\boldsymbol{\pi} = (\pi_{qq'})_{1\leq q,q'\leq Q}$ and initial stationary distribution $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_Q)$.

Given latent groups $\mathbf{Z}$, the time varying random graphs $\mathbf{Y} = (Y^t)_{1\leq t\leq T}$ are independent, the conditional distribution of each Y^t depending only on Z^t. Then, for fixed $1 \leq t \leq T$, random graph Y^t follows a stochastic block model. In other words, for each time t, conditional on Z^t, random variables $(Y^t_{ij})_{1\leq i<j\leq N}$ are independent and the distribution of each Y^t_{ij} only depends on Z^t_i, Z^t_j. For now, we assume a very general parametric form for this distribution on $\mathbb{R}$. Following Ambroise and Matias [2011], in order to take into account possible sparse weighted graphs, we explicitly introduce a Dirac mass at 0, denoted by δ_0, as a component of this distribution. More precisely, we assume

$$Y^t_{ij}|\{Z^t_{iq}Z^t_{jl} = 1\} \sim (1 - \beta^t_{ql})\delta_0(\cdot) + \beta^t_{ql}F(\cdot, \gamma^t_{ql}), \tag{1}$$

where $\{F(\cdot, \gamma), \gamma \in \Gamma\}$ is a parametric family of distributions with no point mass at 0 and densities (with respect to Lebesgue or counting measure) denoted by $f(\cdot, \gamma)$. This could be the Gaussian family with unknown mean and variance, the truncated Poisson family on $\mathbb{N} \setminus \{0\}$ (leading to a 0-inflated or 0-deflated distribution on the edges of the graph), a finite space distribution on M values (a case which comprises nonparametric approximations of continuous distributions through discretization into a finite number of M bins), etc. Note that the binary case is encompassed in this setup with $F(\cdot, \gamma) = \delta_1(\cdot)$, namely the parametric family of laws is reduced to a single point, the Dirac mass at 1 and conditional distribution of Y^t_{ij} is simply a Bernoulli $\mathcal{B}(\beta^t_{ql})$. In the following and by opposition to the 'binary case', we will call 'weighted case' any setup where the set of distributions F is parametrized and not reduced to a single point. Here, the sparsity parameters $\beta^t = (\beta^t_{ql})_{1\leq q,l\leq Q}$ satisfy $\beta^t_{ql} \in [0, 1]$, with $\beta^t \equiv 1$ corresponding to the particular case of a complete weighted graph.

As a result of considering undirected graphs, the parameters $\beta^t_{ql}, \gamma^t_{ql}$ moreover satisfy $\beta^t_{ql} = \beta^t_{lq}$ and $\gamma^t_{ql} = \gamma^t_{lq}$ for all $1 \leq q, l \leq Q$. Note that for the moment, SBM parameters may be different across time points. We will go back to this point in the next sections. The model is thus parameterized by

$$\theta = (\boldsymbol{\pi}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = (\boldsymbol{\pi}, \{\beta^t, \gamma^t\}_{1\leq t\leq T}) = (\{\pi_{qq'}\}_{1\leq q,q'\leq Q}, \{\beta^t_{ql}, \gamma^t_{ql}\}_{1\leq t\leq T, 1\leq q\leq l\leq Q}) \in \Theta,$$

and we let $\mathbb{P}_\theta$ denote the probability distribution on the whole space $\mathcal{Q}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}}$. We also let $\phi(\cdot; \beta, \gamma)$ denote the density of the distribution given by (1), namely

$$\forall y \in \mathbb{R}, \quad \phi(y; \beta, \gamma) = (1 - \beta)1\{y = 0\} + \beta f(y, \gamma)1\{y \neq 0\},$$

where $1\{A\}$ is the indicator function of set A. With some abuse of notation and when no confusion occurs, we shorten $\phi(\cdot; \beta^t_{ql}, \gamma^t_{ql})$ to $\phi^t_{ql}(\cdot)$ or $\phi^t_{ql}(\cdot; \theta)$.

3 Previous Work

In previous inference to estimate the parameter in the above setting, Expectation maximization(EM) was emplyed. Firstly, complete-data log-likelihood of model is given by

$$\begin{aligned}\log\{\mathbb{P}_\theta(\mathbf{Y}, \mathbf{Z})\} = &\sum_{i=1}^{N}\sum_{q=1}^{Q} Z^1_{iq}\log(\alpha_q) + \sum_{t=2}^{T}\sum_{i=1}^{N}\sum_{1\leq q,q'\leq Q} Z^{t-1}_{iq}Z^t_{iq'}\log(\pi_{qq'}) \\ &+ \sum_{t=1}^{T}\sum_{1\leq i<j\leq N}\sum_{1\leq q,l\leq Q} Z^t_{iq}Z^t_{jl}\log\{\phi(Y^t_{ij}; \beta^t_{ql}, \gamma^t_{ql})\}.\end{aligned}$$

and dependence structure of the conditional distribution is given by

$$\mathbb{P}_\theta(\mathbf{Z}|\mathbf{Y}) = \mathbb{P}_\theta(Z^1|Y^1)\prod_{t=2}^{T}\mathbb{P}_\theta(Z^t|Z^{t-1}, Y^t).$$

However, the matter of factorization of conditional distribution of latent $P(Z|Y)$ makes them employ the Variational Expectation Maximization (VEM) algorithm. Variational approximation consider class of probability distribution.

$$\begin{aligned}\mathbb{Q}_\tau(\mathbf{Z}) &= \prod_{i=1}^{N}\mathbb{Q}_\tau(\mathbf{Z}_i) = \prod_{i=1}^{N}\mathbb{Q}_\tau(Z^1_i)\prod_{t=2}^{T}\mathbb{Q}_\tau(Z^t_i|Z^{t-1}_i) \\ &= \prod_{i=1}^{N}\left\{\prod_{q=1}^{Q}\tau(i,q)^{Z^1_{iq}}\right\} \times \prod_{t=2}^{T}\prod_{1\leq q,q'\leq Q}\tau(t,i,q,q')^{Z^{t-1}_{iq}Z^t_{iq'}},\end{aligned}$$

where, for any values (t, i, q, q'), we have $\tau(i, q)$ and $\tau(t, i, q, q')$ both belong to the set $[0, 1]$ and are constrained by $\sum_q \tau(i, q) = 1$ and $\sum_{q'} \tau(t, i, q, q') = 1$. This class of probability distributions $\mathbb{Q}_\tau$ corresponds to considering independent laws through individuals, whereas, for each $i \in \{1, \ldots, N\}$, the distribution of Z_i under $\mathbb{Q}_\tau$ is the distribution of a Markov chain (through time t), with inhomogeneous transition $\tau(t, i, q, q') = \mathbb{Q}_\tau(Z_i^t = q' | Z_i^{t-1} = q)$ and initial distribution $\tau(i, q) = \mathbb{Q}_\tau(Z_i^1 = q)$.

We shall need the marginal components of $\mathbb{Q}_\tau$, namely $\tau_{\text{marg}}(t, i, q) := \mathbb{Q}_\tau(Z_i^t = q)$. These quantities are computed recursively by

$$\tau_{\text{marg}}(1, i, q) = \tau(i, q) \quad \text{and} \quad \forall t \geq 2,$$

$$\tau_{\text{marg}}(t, i, q) = \sum_{q'=1}^{Q} \tau_{\text{marg}}(t-1, i, q')\, \tau(t, i, q', q).$$

The entropy of $\mathbb{Q}_\tau$ is denoted by $\mathcal{H}(\mathbb{Q}_\tau)$. Using this class of probability distributions on $\mathbb{Q}^{\mathbb{N}}$, the VEM algorithm is an iterative procedure to optimize the criterion

$$\begin{aligned} J(\theta, \tau) &:= \mathbb{E}_{\mathbb{Q}_\tau}\left[\log\{\mathbb{P}_\theta(\mathbf{Y}, \mathbf{Z})\}\right] + \mathcal{H}(\mathbb{Q}_\tau) \\ &= \sum_{i=1}^{N} \sum_{q=1}^{Q} \tau(i, q) \left[\log(\alpha_q) - \log\{\tau(i, q)\}\right] \\ &+ \sum_{t=2}^{T} \sum_{i=1}^{N} \sum_{1 \leq q, q' \leq Q} \tau_{\text{marg}}(t-1, i, q)\, \tau(t, i, q, q') \left[\log(\pi_{qq'}) - \log\{\tau(t, i, q, q')\}\right] \\ &+ \sum_{t=1}^{T} \sum_{1 \leq i < j \leq N} \sum_{1 \leq q, l \leq Q} \tau_{\text{marg}}(t, i, q)\, \tau_{\text{marg}}(t, j, l) \log\{\phi_{ql}^t(Y_{ij}^t)\}. \end{aligned}$$

During the Variational Expectation (VE) step, the parameter τ is optimized to maximize $J(\theta, \tau)$, while in the Maximization (M) step, the parameter θ is optimized to maximize $J(\theta, \tau)$. To determine the maximum likelihood estimator (MLE) of the parameters, fixed-point equations are solved, and the resultant values are directly used. However, for certain parameters, such as $\tau(i, q)$ and α_q, which cannot be accurately obtained through fixed-point equations, empirical means are applied in the calculations.

Nonetheless, previous work exhibits several notable drawbacks. Firstly, the computational cost associated with calculating the MLE of the parameters is substantial. Prior methods for computing the J value exhibit an overall time complexity of $O(TN^2Q^2)$. Additionally, executing the VEM algorithm requires several iterations, incurring a computational cost of $O(TN^3Q^3)$, as derived from Proposition 2 in the referenced study. Moreover, this solution yields only a local maximum, lacking a comprehensive search for a global optimum. The algorithm directly computes a solution where the derivative equals zero without exploring the neighboring solution space. Consequently, the initialization process, often utilizing K-means clustering, plays a crucial role in avoiding local minima during maximization, leading to a high time cost to compute the MLE.

The implementation of this method also presents several challenges. It is essential to use `float64` as the parameter value type to prevent underflow, given that most values represent probabilities between 0 and 1. Multiplying these values $O(TN^3Q^3)$ times can drive the result close to zero, requiring substantial memory and diminishing computational efficiency. Furthermore, it is non-trivial to observe a decrease in performance by a factor of five to ten at certain time points. As the dataset size increases, the average performance should ideally improve.

4 Our Work

To address these challenges, our proposed method calculates the MLE of the parameters using an *optimizer* with *float32* precision, *random initialization*, and *tensor operations* to compute J. The algorithm of our method is described in Algorithm 2.

Firstly, our approach leverages tensor operations to substantially reduce the time complexity. Specifically, we achieve a time complexity of $O(T)$, primarily attributed to the sequential computation of the marginal probability of τ, which aligns naturally with the temporal order in J. The remaining calculations are achieved in $O(1)$ by utilizing tensor operations, which take advantage of the enhanced computational power of modern processing units.

Secondly, instead of relying on solutions obtained through fixed-point equations, we employ an optimizer. The fundamental difference between the optimizer method and the fixed-point equation method lies in their approach to finding the optimum. The fixed-point method identifies an exact local optimum for a given parameter and can directly converge to a local optimum within approximately 100 iterations. In contrast, an optimizer explores the search space more extensively, aiming to approximate the global optimum by employing techniques such as momentum and adaptive algorithms. Since the optimizer moves the estimator incrementally towards the optimum, it requires a greater number of iterations.

Finally, due to the significant reduction in time complexity, we utilize random initialization and test multiple random seeds. In previous work, K-Means initialization was necessary because, without tensor operations, each iteration of the VEM step required approximately 2 minutes to complete. Consequently, bias information introduced by the K-means algorithm, a human-designed heuristic, was incorporated to complete the task more efficiently. However, our findings indicate that a fixed initialization via K-means does not significantly improve results; in fact, it may even lead to worse outcomes by causing the algorithm to become trapped in suboptimal local minima. Instead, we employ multiple trials of random initialization, which avoids introducing any biased information.

4.1 Implementation

The process of calculating the J function, as outlined in Algorithm 1, is a key component of our approach. This algorithm utilizes tensor operations to efficiently compute each term involved in J, significantly reducing computation time compared to traditional methods. By leveraging the `einsum` function for tensor manipulation, we are able to perform complex summations and multiplications in parallel, allowing for faster and more scalable calculations. The optimizer-based method for statistical clustering, described in Algorithm 2, incorporates softmax and sigmoid functions to enforce constraints on parameters during optimization, ensuring valid probability distributions. Additionally, an early stopping mechanism monitors improvements in the loss function, reducing unnecessary iterations and improving computational efficiency.

To implement our method, it is necessary to handle certain aspects of the original algorithm presented in previous work. Each parameter is subject to specific constraints. For instance, probability parameters must lie between 0 and 1, and the sums of π and τ over N or Q should equal 1. However, during the optimizer's descent process, these constraints might be violated if they are not explicitly enforced. To address this, we apply methods commonly used in deep learning, such as the softmax and sigmoid functions, to ensure that constraints are maintained throughout gradient descent. For example, for $\tau(i, q)$, the sum of probabilities across all latent states q for each node i must equal 1, so we apply the softmax function over the latent states q. Similarly, β must be constrained between 0 and 1, so we apply the sigmoid function. Consequently, we divide each parameter according to its constraints and apply the necessary transformations.

In previous implementations, the use of `float64` precision was essential due to issues related to underflow. Excessive multiplications of probability values during the calculation of J and in each VEM step caused values to become exceedingly small, often below 10^{-39}. This issue typically arises from the repeated multiplication of probabilities associated with latent variables, leading to numerical underflow. When underflow occurs, the VEM algorithm may encounter divisions of the form $0/0$, resulting in computational errors and instability. Although logarithmic transformations could theoretically address this, certain terms involve both summation and multiplication, making factorization infeasible. Clamping values to mitigate underflow partially solves the problem but introduces inaccuracies and negatively impacts performance.

By adopting sigmoid and softmax functions within our implementation, we are able to use `float32` precision, which provides considerable memory efficiency. The inverse of the sigmoid and softmax functions at 0 is well-defined, allowing us to avoid underflow issues without compromising the integrity of the calculations. Consequently, using `float32` reduces memory consumption while retaining computational accuracy, enabling the algorithm to operate more efficiently on modern hardware with limited memory resources.

Subsequent calculations of J follow the method presented in Section 2.1. The optimizer then updates τ and θ to maximize J. Similar to the VEM algorithm, τ and θ are not updated simultaneously for stability in convergence. A scheduler is employed, starting with a high learning rate that is gradually reduced as the updates proceed. The maximum number of iterations is fixed, and a stopping criterion is introduced. This stopping criterion tracks the number of times the improvement in J does not exceed a threshold within every 100 iterations. The process terminates if this count surpasses a predefined patience parameter.

To leverage tensor operations, large tensors are constructed to enable summation and multiplication in parallel. For example, $\phi_{ql}^{t}(Y_{ij})$ has distinct values for each $q, l \in \{1, \ldots, Q\}$, $i, j \in \{1, \ldots, N\}$, and $t \in \{1, \ldots, T\}$. Consequently, when computing ϕ, we construct a complete matrix of size $N \times Q \times Q \times N \times N$ by expanding the indices q, l, and t, along with the adjacency matrix. Using broadcasting techniques, the entire ϕ tensor can be computed in a single

operation. Additionally, we employ the `einsum` function to compute summation terms in parallel. For instance, the third term can be calculated by using `einsum` with two instances of `tau_marg` and summing over the upper triangular matrix. By doing so, all computations can be performed in parallel, though for the time series, this part follows $O(T)$ complexity.

Taking advantage of the computational efficiency enabled by tensor operations, our method can be implemented with multiple trials using random initialization. The random initialization draws values between 0 and 1, and the initial values are transformed back to the unrestricted real number domain using the inverse functions of softmax and sigmoid (log transformations for softmax and the inverse of sigmoid, respectively).

Algorithm 1 Calculation of J Function with tensor operation

Require: Initial $\tau\_init \in \mathbb{R}^{N\times Q}$, $\tau\_transition \in \mathbb{R}^{(T-1)\times N\times Q\times Q}$, $\alpha \in \mathbb{R}^{Q}$, $\pi \in \mathbb{R}^{Q\times Q}$, $\beta \in \mathbb{R}^{T\times Q\times Q}$
Require: Adjacency matrix $Y \in \mathbb{R}^{T\times N\times N}$
Ensure: Computed $J\_value \in \mathbb{R}$
  Generate τ_marg**:** ▷ $\tau\_marg \in \mathbb{R}^{T\times N\times Q}$
  Set initial result $\leftarrow \tau\_init$
  for $t = 0$ to $T-1$ **do**
    if $t == 0$ **then**
      $\tau\_marg[t,:,:] \leftarrow$ result
    else
      result $\leftarrow$ einsum$('ij, ijk \rightarrow ik', \text{result}, \tau_transition[t,:,:,:])$
      $\tau\_marg[t,:,:] \leftarrow$ result
    end if
  end for
  Create index tensors:
    $q\_indices \leftarrow$ Tensor of indices representing Q dimension
      $q\_indices \in \mathbb{R}^{T\times Q\times Q\times N\times N} \leftarrow$ Repeat Q indices across all dimensions
    $l\_indices \leftarrow$ Tensor of indices representing the latent states
      $l\_indices \in \mathbb{R}^{T\times Q\times Q\times N\times N} \leftarrow$ Repeat latent state indices across all dimensions
    $t\_indices \leftarrow$ Tensor of indices representing time steps
      $t\_indices \in \mathbb{R}^{T\times Q\times Q\times N\times N} \leftarrow$ Repeat time step indices across all dimensions
    $Y\_indices \leftarrow$ Tensor of indices representing adjacency matrix entries
      $Y\_indices \in \mathbb{R}^{T\times Q\times Q\times N\times N} \leftarrow$ Expand adjacency matrix indices across all dimensions
  $\phi\_values\_new \in \mathbb{R}^{T\times Q\times Q\times N\times N} \leftarrow \phi\_vectorized(q\_indices, l\_indices, t\_indices, Y\_indices, \beta)$
  $log\_phi\_values \in \mathbb{R}^{T\times Q\times Q\times N\times N} \leftarrow \log(\phi\_values\_new)$
  Calculate the first term:
  $term1 \leftarrow \sum(\tau\_init \times (\log(\alpha) - \log(\tau\_init))) \in \mathbb{R}$
  Calculate the second term:
  $term2 \leftarrow 0$
  for $t = 1$ **to** $T-1$ **do**
    $A \leftarrow$ einsum$('ij, ijk \rightarrow ijk', \tau_marg[t-1,:,:], \tau_transition[t,:,:,:]) \in \mathbb{R}^{N\times Q\times Q}$
    $B \leftarrow \log(\pi) \in \mathbb{R}^{Q\times Q}$ ▷ Logarithm of π
    Expand B to $\mathbb{R}^{N\times Q\times Q}$ by copying along N dimension
    $C \leftarrow \log(\tau\_transition[t,:,:,:]) \in \mathbb{R}^{N\times Q\times Q}$ ▷ Logarithm of $\tau\_transition$
    $term2 \leftarrow term2 + \sum(A \times (B - C))$ ▷ Sum of elementwise products
  end for
  Calculate the third term:
  $indices \leftarrow$ upper triangular indices of $N \times N$
  $D \leftarrow$ einsum$('aik, ajl \rightarrow aklij', \tau_marg, \tau_marg) \in \mathbb{R}^{T\times Q\times Q\times N\times N}$ ▷ Tensor product of τ_marg
  $term3 \leftarrow \sum(D \times log\_phi\_values)[:,:,:,indices[0], indices[1]] \in \mathbb{R}$ ▷ Sum of selected elements
  Sum all terms:
  $J\_value \leftarrow term1 + term2 + term3 \in \mathbb{R}$
  return J_value

Algorithm 2 Statistical Clustering using Optimizer

Require: Adjacency matrix Y
Require: Patience $P \in \mathbb{N}$
Require: Threshold $T \in \mathbb{R}^+$
Require: Learning rate $lr \in [0, 1]$, typically $lr = 0.1$
  Initialize $best\_loss \leftarrow -\infty$
  Initialize $no\_improve\_count \leftarrow 0$
  Define $threshold$ ▷ Set threshold for loss improvement
  Define $patience$ ▷ Set patience for early stopping
  Initialize $\tau_{\text{init}}^{\text{pre}} \in \mathbb{R}^{N \times Q}$
  Initialize $\tau_{\text{tran}}^{\text{pre}} \in \mathbb{R}^{(T-1) \times N \times Q \times Q}$
  Initialize $\alpha^{\text{pre}} \in \mathbb{R}^{Q}$
  Initialize $\pi^{\text{pre}} \in \mathbb{R}^{Q \times Q}$
  Initialize $\beta^{\text{pre}} \in \mathbb{R}^{T \times Q \times Q}$
  while not converged **do**
    for each iteration i in 1 to $num\_iterations$ **do**
      Reset gradients for optimizer$_\theta$
      Reset gradients for optimizer$_\tau$
      $\tau_{\text{init}} \leftarrow \text{Softmax}(\tau_{\text{init}}^{\text{pre}}, \text{dim} = 1)$
      $\tau_{\text{tran}} \leftarrow \text{Softmax}(\tau_{\text{tran}}^{\text{pre}}, \text{dim} = 3)$
      $\alpha \leftarrow \text{Softmax}(\alpha^{\text{pre}}, \text{dim} = 0)$
      $\pi \leftarrow \text{Softmax}(\pi^{\text{pre}}, \text{dim} = 1)$
      $\beta \leftarrow \text{Sigmoid}(\beta^{\text{pre}})$
      Compute loss $Loss \leftarrow -J(\tau_{\text{init}}, \tau_{\text{tran}}, \alpha, \pi, \beta, Y)$
      Backpropagate loss L
      Update parameters using optimizer$_\theta$
      Update parameters using optimizer$_\tau$
      if $iter \bmod 100 = 0$ **then**
        if $best\_loss - Loss < threshold$ **then**
          $no\_improve\_count \leftarrow no\_improve\_count + 1$ ▷ No significant improvement
        else
          $best\_loss \leftarrow current\_loss$ ▷ Update best loss
          $no\_improve\_count \leftarrow 0$ ▷ Reset no improvement counter
        end if
        if $no\_improve\_count \geq patience$ **then**
          print "Stopping early at iteration ", $iter$, " due to no improvement."
          break ▷ Exit the loop early
        end if
      end if
    end for
  end while
  return $\tau_{\text{init}}, \tau_{\text{tran}}, \alpha, \pi, \beta$

5 Synthetic experiments

Using this approach, we demonstrate in the following sections that our method, compared to previous work, yields comparable and, in some cases, superior results in synthetic experiments. We developed four models based on combinations of our method (optimizer) and the previous method (fixed-point equation), as well as on random initialization and K-Means initialization. Specifically, we conducted eight trials with random initialization and one trial with K-Means initialization, since the K-Means case produces consistent initial values. For a precise comparison, we share the initialization values and adjacency matrix across 100 simulated datasets. The iteration count for the optimizer is set to 5000 iterations for case (i) and 10000 iterations for case (ii). Since the previous method terminates quickly, the number of iterations is less relevant for those cases.

i) Our method with random initialization
ii) Our method with K-Means initialization
iii) Previous method with random initialization

iv) Previous method with K-Means initialization

We conducted experiments with varying numbers of time stamps and different values for the transition matrix. Specifically, we fixed the number of nodes at 100 and experimented with 5 and 10 time stamps, along with the following three transition matrices:

$$\pi_{\text{high}} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}, \quad \pi_{\text{medium}} = \begin{pmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{pmatrix}, \quad \pi_{\text{low}} = \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}$$

These three cases correspond to high, medium, and low group stability, respectively. In the first case, individuals are more likely to switch groups over time, which presents a greater challenge for initializing our algorithm. The stationary distribution for all three cases is $\alpha = (1/2, 1/2)$, meaning that both groups have similar proportions.

Table 1: Bernoulli parameter values in four cases, plus an affiliation example

Easiness	β_{11}	β_{12}	β_{22}
Low−	0.2	0.1	0.15
Low+	0.25	0.1	0.2
Medium−	0.3	0.1	0.2
Medium+	0.4	0.1	0.2
Medium with affiliation	0.3	0.1	0.3

Regarding the Bernoulli parameters β, we explored four different scenarios (shown in Table 1) representing various levels of difficulty, along with a specific example of affiliation. For this affiliation case, we note that the parameters are non-identifiable in the dynamic setting. The affiliation case meets the separability condition established for sparse planted partition models, which suggests that static reconstruction of groups is feasible (we classify this as a problem of medium difficulty).

For each combination of (π, β), we generated 100 datasets, estimated their parameters, clustered their nodes, and reported the results as boxplots of global and averaged ARI (Adjusted Rand Index) values analog to previous paper in Figure 1. In Figure 1, we observe that the problem becomes easier to solve when β and π values are more distinguishable. The global ARI, however, struggles to perform as well as the average ARI in more complex cases.

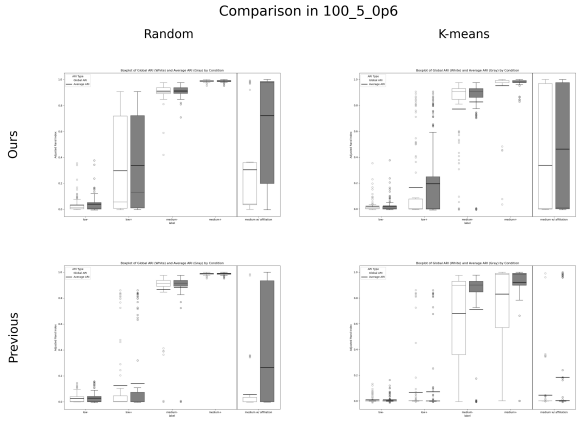
Figure 1 shows that our method with random initialization outperforms the previous method in all simulated cases, as evidenced by the mean scores of both global and average ARI, where our method consistently exceeds those of the previous method. Since the numerical method employed in the previous approach only finds a local optimum based on the initial point, our method with random initialization, which searches the parameter space more comprehensively, achieves better ARI scores.

Comparing cases (i) and (ii) with cases (iii) and (iv), it is evident that multiple trials with random initialization yield better performance than fixed K-Means initialization. The predefined τ values in the K-Means approach seem to introduce bias, hindering the discovery of more optimal solutions. This phenomenon appears in both our and the previous methods, suggesting that random initialization, without such biases, enhances performance when repeated across trials.

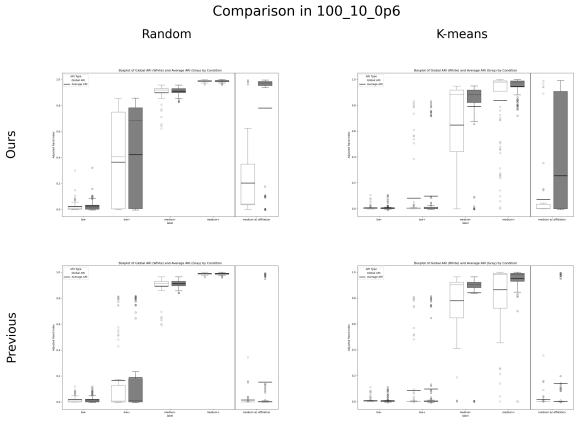
In cases involving the low-minus and low-plus configurations for (i) and (ii), the previous method struggles to find the optimal solution. However, our method demonstrates the ability to handle these challenging configurations. In the previous work, low-minus and low-plus configurations were deemed unsolvable; however, as seen in our low-minus cases, these scenarios can indeed be addressed by our approach.

When comparing results across different numbers of time stamps, we observe that 10 time stamps yield better performance than 5. Longer time stamps provide more information, aiding the inference process. However, in certain cases, such as low-plus with 10 time stamps, the previous method (case iv) sometimes performs worse than with 5 time stamps (comparison of (e)(iv) and (f)(iv)). The original paper attributes this to the instability of the marginal τ over extended time periods. In our experience, however, longer time stamps consistently lead to improved performance across all cases.

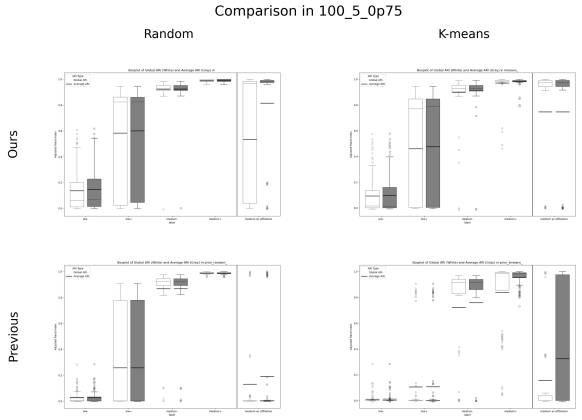
Compared to other methods, our method with random initialization exhibits fewer significant low outliers. Most low outliers result from either convergence to a local optimum or the inherent difficulty of the problem. Low outliers in the medium-minus configuration are significantly reduced in case (i) compared to other methods. This is because the optimizer has the ability to explore the parameter space more thoroughly, thereby increasing the likelihood of finding the global optimum rather than being confined to a local optimum.



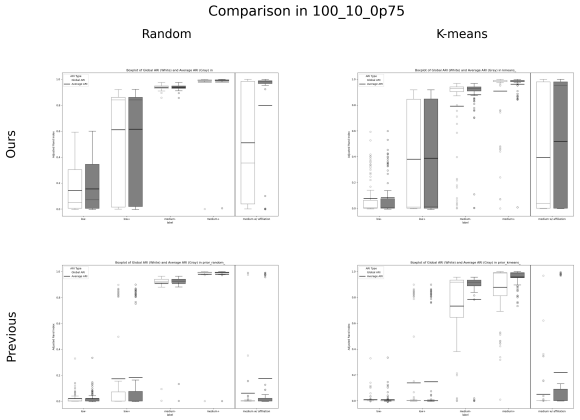
(a) 5 Time Stamps & Low Stability



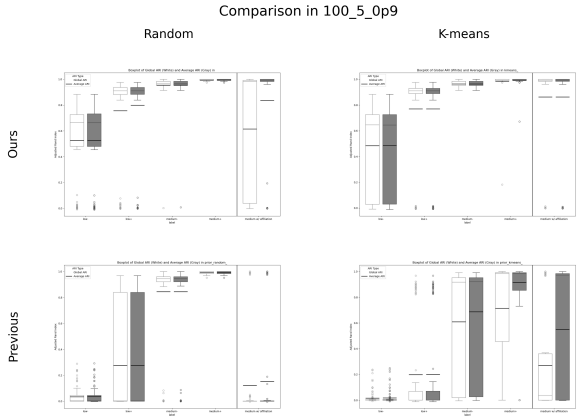
(b) 10 Time Stamps & Low Stability



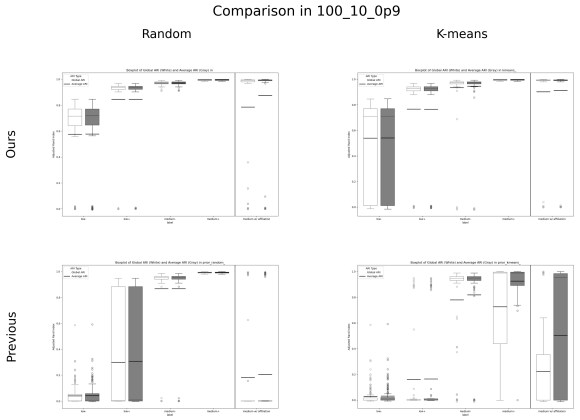
(c) 5 Time Stamps & Medium Stability



(d) 10 Time Stamps & Medium Stability



(e) 5 Time Stamps & High Stability



(f) 10 Time Stamps & High Stability

Figure 1: synthesis data

Another crucial aspect of our results is the time cost associated with the different methods. Without tensor operations, a `for` loop implementation would be required, leading to a substantial increase in computational time. The previous method, when implemented with tensor operations, requires approximately 0.08 seconds per one iteration for each VEM step (using Intel Xeon Gold 6348 CPU@2.60GHz(num:56)). In comparison, our method, also implemented with tensor

operations, requires approximately 0.03 seconds per one iteration(using Intel Xeon Gold 6348 CPU@2.60GHz(num:56)). This relatively small time per iteration allows us to perform 5000 iterations across 8 trials. The reduction in computational time enables the use of multiple trials with random initialization, making this approach feasible and affordable.

References

Catherine Matias and Vincent Miele. 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, 2017. ISSN 13697412, 14679868. URL https://www.jstor.org/stable/26773154.

Christophe Ambroise and Catherine Matias. New Consistent and Asymptotically Normal Parameter Estimates for Random-Graph Mixture Models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 74(1): 3–35, 10 2011. ISSN 1369-7412. doi:10.1111/j.1467-9868.2011.01009.x. URL https://doi.org/10.1111/j.1467-9868.2011.01009.x.