Supplementary Material of "Directed Acyclic Graph Structure Learning from Dynamic Graphs"

Shaohua Fan[1], **Shuyang Zhang**[1], **Xiao Wang**[1], **Chuan Shi**[1*]
[1]Beijing University of Posts and Telecommunications, China
{fanshaohua, sonyazhang, xiaowang, moyanhu, shichuan}@bupt.edu.cn

A Dataset Details

A.1 Intra-slice graph and inter-slice graph generation

Intra-slice graph $\mathcal{G}_{\mathbf{W}}$. Given a target mean degree k, we use either the Erdős-Rényi (ER) model [2] or the Barabási-Albert (BA) model [1] to generate intra-slice graphs. To ensure that the resulting graph is a DAG, we sample upper-triangular entries of the generated $\mathbf{W}$, and we then randomly permute the orders of rows and columns. The BA model generates the graph according to the "preferential attachment" mechanism in which nodes are added one by one. For each new node, one generates $k/2$ outgoing edges, in which the targets of these nodes are selected proportionally to their current degrees. This mechanism simulates a "rich get richer" effect that produces graphs with a power-law degree distribution at the end of the process. BA graphs are popular since they exhibit topological properties similar to real-world networks such as gene networks, social networks, and the internet.

Inter-slice graph $\mathcal{G}_{\mathbf{P}}$. We use ER model or Stochastic Block Model (SBM) [2] to generate inter-slice graphs. For ER model, its setting is the same as ER model used for intra-slice graph expect that we do not need to transform the graph into acyclic. For SBM model, we cluster the d variables into two blocks, and we set the probability of an inter-slice edge from i to j as p_{in} if i and j are in the same block, and it is set as p_{out} otherwise. We set p_{in} as 0.3 and p_{in} as 0.09 such that $p_{out}/p_{in} = 0.3$, so edges are more likely between two variables in the same cluster. In some real-world applications, the variables are expected to form as clusters, and SBMs could replicate this feature in our simulation experiments.

A.2 Synthetic Data Generating Process

Here we describe our synthetic data generation process in detail. We first generate intra-slice graph $\mathcal{G}_{\mathbf{W}}$, inter-slice graph $\mathcal{G}_{\mathbf{P}}$, and adjacency matrix $\mathbf{A}$ based on the methods described in App. A.1. Then based on these matrices, we generate the node features for each timestamp in Algorithm 1.

In Algorithm 1, the loop in line 3-14 is to generate p time-lagged base node features, which are used as the neighbors' features for generating the first timestamp for prediction. The loop in line 15-26 is to generate T steps node features. For each step, as formulated in line 23, each variable $ret[node]$ is generated based on the parents variable from contemporaneous parents variables $ret[W_{par}] * \mathbf{W}[W_{par}, node]$ and time-lagged parents variables $(\mathbf{A}^{(i-1)} * \mathbf{X}[i-1])[P_{par}^{(i-1)}] * \mathbf{P}^{(i-1)}[P_{par}^{(i-1)}, node] + \cdots + (\mathbf{A}^{(i-p)} * \mathbf{X}[i-p])[P_{par}^{(i-p)}] * \mathbf{P}^{(i-p)}[P_{par}^{(i-p)}, node]$. Hence, the variables of ret in each timestamp exhibit a casual order, which conforms with the definition of SVAR model.

*Corresponding Author.

Preprint. Under review.

Algorithm 1: Synthetic Data Generating Process

Input : Length of time-series: T; length of time-lagged influence: p; number of nodes: n; number of features: d;

Output : Node features of dynamic graph: $\{\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(T)}\}$;

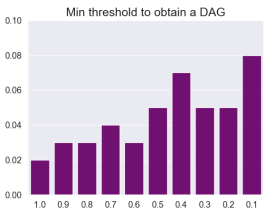
Initialization : Generate intra-slice graph $\mathcal{G}_{\mathbf{W}}$, inter-slice graph $\mathcal{G}_{\mathbf{P}}$ and corresponding matrices $\mathbf{W}$ and $\mathbf{P}$ with d variables; generate adjacency matrix $\mathbf{A}$ with n nodes.

$\mathcal{G}_{\mathbf{W}}^{sorted} \leftarrow topological\_graph\_sort(\mathcal{G}_{\mathbf{W}})$;
$\mathbf{X} \leftarrow [\,]$
for *1 to* p **do**
  ret$\leftarrow$ []
  for *node* $\in \mathcal{G}_{\mathbf{W}}$ **do**
    Append to ret[node] a list of Gaussian/Exponential randomly sampled list of size n;
  end
  for *node* $\in \mathcal{G}_{\mathbf{W}}^{sorted}$ **do**
    for *par* $\in$ *parents(node)* **do**
      ret[node] += ret[par] * W[par, node], where $\mathbf{W}[par, node]$ is the edge weight from par to $node$;
    end
  end
  Append ret to $\mathbf{X}$;
end
for i *in range(p, T)* **do**
  ret$\leftarrow$ []
  for *node* $\in \mathcal{G}_{\mathbf{W}}$ **do**
    Append to ret[node] a list of Gaussian/Exponential randomly sampled list of size n;
  end
  for *node* $\in \mathcal{G}_{\mathbf{W}}^{sorted}$ **do**
    $W_{par} = \mathcal{G}_{\mathbf{W}}^{Par}(node)$\\ parents of $node$ in graph $\mathcal{G}_{\mathbf{W}}$
    $P_{par}^{(i)} = \mathcal{G}_{\mathbf{P}^{(i)}}^{Par}(node)$ \\ parents of $node$ in graph $\mathcal{G}_{\mathbf{P}^{(i)}}$
    $ret[node] = ret[W_{par}] * \mathbf{W}[W_{par}, node] + (\mathbf{A}^{(i-1)} * \mathbf{X}[i-1])[P_{par}^{(i-1)}] * \mathbf{P}^{(i-1)}[P_{par}^{(i-1)}, node] + \cdots + (\mathbf{A}^{(i-p)} * \mathbf{X}[i-p])[P_{par}^{(i-p)}] * \mathbf{P}^{(i-p)}[P_{par}^{(i-p)}, node]$;
    \\$\mathbf{W}[W_{Par}, node]$ is the edge weight from W_{Par} to $node$; $\mathbf{P}^{(i)}[P_{Par}^{(i)}, node]$ is the edge weight from $P_{Par}^{(i)}$ to $node$;
  end
  Append ret to $\mathbf{X}$;
end
Return: $\mathbf{X}$

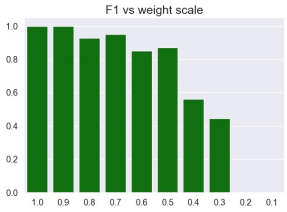
B Additional Experiments

B.1 Sensitivity of Weight Scale

We investigate the effect of weight scaling to the GraphNOTEARS algorithm in Figure 1. Particularly, we conduct experiments with $\mathbf{W}_{ij} \in \alpha \cdot [0.5, 2] \cup -\alpha \cdot [0.5, 2]$ with $\alpha \in \{1.0, 0.9, \ldots, 0.1\}$. On the left, we plot the smallest threshold $\tau_{\mathbf{W}}$ required to obtain a DAG for different scale α (as the overall DAG-ness is determined by $\mathbf{W}$, here we only study on $\mathbf{W}$). Overall, across different α, the variation in the smallest $\tau_{\mathbf{W}}$ required is minimal. The low values for the thresholds demonstrate that the algorithm recovers a DAG easily. On the right, the performance drops when using smaller value of α. The reason is that, with the signal to noise ratio (SNR) increases, it is hard to recover the DAGs from such noisy data.



(a) Smallest threshold $\tau_{\mathbf{W}}$ ensuring $\hat{\mathbf{W}}$ is a DAG.



(b) F1 score correspondings to different α

Figure 1: Varying weight scale $\alpha \in \{1.0, 0.9, \ldots, 0.1\}$ with $d = 20$ and $n = 500$ on $\mathbf{W}$ and $\mathbf{P}$ generated with ER graphs. The minimum $\tau_{\mathbf{W}}$ remains stable, while the F1-score of GraphNOTEARS decreases as expected as the SNR decreases.

References

[1] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[2] Mark Newman. *Networks*. Oxford university press, 2018.