# Learning semi-Markovian DAGs with flow-based VAE

**Dangchan Kim, Byungguk Kang, Jaeseok Kim, Minchan Kim**

December 4, 2023

**Abstract**

We propose a method to learn the structure of a semi-Markovian directed acyclic graph, which is a mixed graph containing both directed and bidirectional edges, using a flow-based variational autoencoder. The proposed method is based on the assumption that noise variables in the linear structural equation model can be considered as latent variables. To learn the structure of the mixed graph, we employ an inverse autoregressive flow to approximate the dependency structure of the noise variables and its prior distribution. We conducted experiments on simulated data by adjusting the number of nodes and the proportion of bidirectional edges. The code is available at `https://github.io/ddangchani/NFG-VAE`.

## 1 Introduction

Directed acyclic graphs (DAGs) are commonly used to represent causal relationships between variables [1]. However, recovering the causal structure from observational data is an NP-hard problem [2]. To overcome this problem, several methods have been proposed. NOTEARS [3] is a method that learns the structure of DAGs by minimizing the continuous approximation of the discrete constraint, enabling the learning of DAGs through continuous optimization.

Based on the continuous acyclicity constraint neural network-based approaches known as graph neural networks, have been proposed. [4, 5, 6, 7] The most representative model would be DAG-GNN [4], a method that learns the structure of DAGs by using a neural network that takes the adjacency matrix of the graph as an additional input. It utilizes a variational autoencoder (VAE) to learn DAGs by interpreting the noise variable in the linear structural equation model as a latent variable.

However, in some cases, the noise variables may not be independent. For example, in a linear structural equation model with correlated hidden variables, the noise variables may exhibit a dependency structure. In such cases, edges with correlation in the noise variable are considered bidirectional edges, and these models are referred to as semi-Markovian [8]. Since the prior distribution of the noise variable is assumed to be a standard Gaussian in DAG-GNN or other graph neural networks, it becomes challenging to learn the structure of semi-Markovian DAGs. Therefore, we propose a method to learn the full covariance matrix of the noise variable using a flow-based VAE.

## 2 Directed Acyclic Graphs and Mixed Graphs

### 2.1 Directed Acyclic Graphs

Here is the introduction of DAGs

### 2.2 Mixed Graphs

Here is the introduction of mixed graphs

### 2.3 Graph Learning

Here is the introduction of graph learning

## 3 Variational Autoencoders and Normalizing Flows

### 3.1 Variational Autoencoders

Here is the introduction of VAEs

### 3.2 Normalizing Flows

Normalizing flows are a method of transforming a simple distribution into a complex distribution [9]. Let $z$ be a random variable with a simple distribution $p(z)$, and $x$ be a random variable with a complex distribution $p(x)$. We can transform $z_0$ into $z_T$ using a series of invertible transformations $\{f_k\}_{k=1,\dots,T}$ as follows:

$$z_T = f_T \circ f_{T-1} \circ \cdots \circ f_1(z_0) \tag{1}$$

The probability density function of $x$ can be obtained by the change of variables formula:

$$p(z_T) = p(z_0) \left| \det \left( \frac{\partial f_T \circ f_{T-1} \circ \cdots \circ f_1(z_0)}{\partial z_0} \right) \right| \tag{2}$$

In VAE, the prior distribution of the latent variable $\mathbf{z}$ is typically assumed to be a standard Gaussian distribution or a multivariate Gaussian distribution with a diagonal covariance matrix. [10] However, to employ a more flexible prior distribution, we can utilize normalizing flows to transform the standard Gaussian distribution into a more complex distribution. This allows us to capture richer and more intricate patterns in the latent space, such as correlation between latent variables.

There are several types of normalizing flows, such as Householder flow [11], planar flow [9], etc. In this paper, we use the inverse autoregressive flow (IAF) [12], especially linear IAF. The linear IAF is defined as follows:

$$\mathbf{z_t} = \mu_\mathbf{t} + \sigma_\mathbf{t} \odot \mathbf{z_{t-1}}. \tag{3}$$

The linear IAF is the simplest case of IAF, which transforms a multivariate Gaussian with diagonal covariance to a multivariate Gaussian with full covariance using a single flow layer. The transformation is invertible and the determinant of the Jacobian is easily computable. To use linear IAF at VAE, we need to produce an extra output $\mathbf{L}(\mathbf{x})$ from the encoder network. The full-covariance Gaussian distribution is obtained by the following transformation:

$$\mathbf{z}_T = \mathbf{L}(\mathbf{x}) \cdot \mathbf{z}_0. \tag{4}$$

Note if we restrict the $\mathbf{L}(\mathbf{x})$ to be a lower triangular matrix with diagonal elements of ones, then the log-determinant of the Jacobian is zero, which is so-called volume preserving normalizing flow. [12]

With the linear IAF, the KL loss term can be written in closed form as follows:

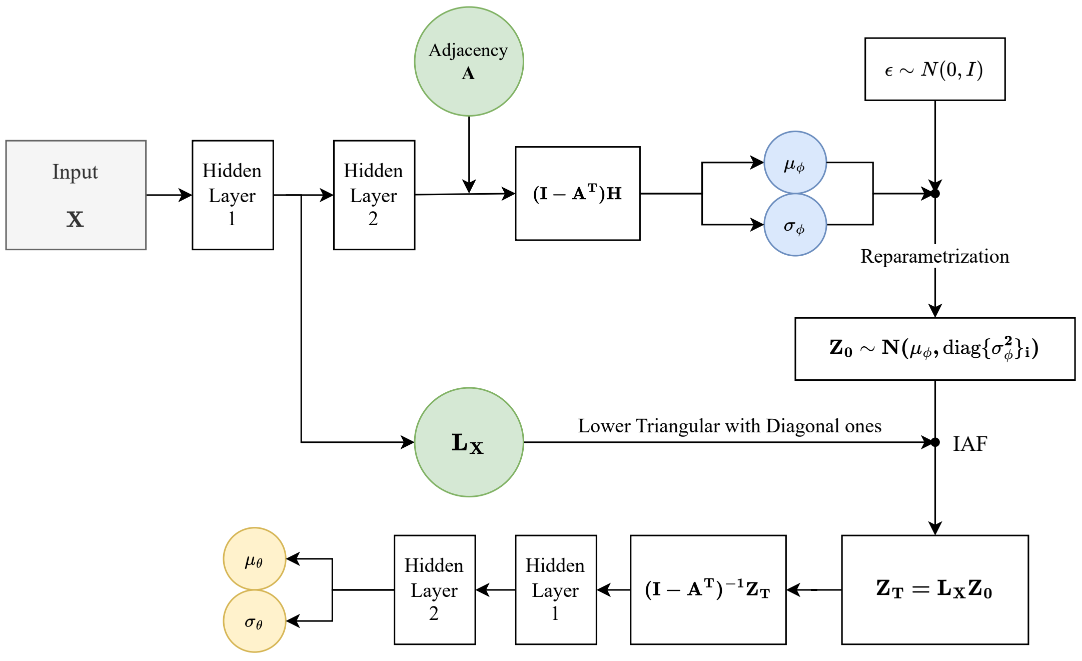
$$\begin{aligned} D_{KL}(q_\phi(\mathbf{z}_0|\mathbf{x}) \| p(\mathbf{z}_T)) &= \log q_\phi(\mathbf{z}_0|\mathbf{x}) - \log p(\mathbf{z}_T) \\ &= -\frac{1}{2}(\mathbf{z}_0 - \mu_\phi)^T \Sigma_\phi^{-1} (\mathbf{z}_0 - \mu_\phi) + \frac{1}{2}\mathbf{z}_T^T \mathbf{z}_T \end{aligned} \tag{5}$$

where $\mu_\phi$ and $\Sigma_\phi$ are the mean vector and covariance matrix obtained from the encoder network.

## 4 Method

### 4.1 Model

The overall architecture of our model is shown in Figure 1. The encoder network takes the input data matrix $\mathbf{X} \in \mathbb{R}^{B\times m\times 1}$ and adjacency matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ as inputs and outputs the mean vector $\mu_\phi$ and (log) variance $\sigma_\phi$ of the latent variable $\mathbf{z}_0$. The decoder network takes the latent variable $\mathbf{z}_T$ and the same adjacency matrix $\mathbf{A}$ used at encoder as inputs and outputs the mean vector $\mu_\theta$ and covariance matrix $\Sigma_\theta$ of the reconstructed data matrix $\mathbf{X}$. Also, during the training process the decoder variance $\sigma_\theta$ is fixed to 1. For the size of the hidden layer, we use 64. Note that the $\mathbf{L}_\mathbf{X}$ is an additional output from the encoder network. Since the initial output $\mathbf{L}_\mathbf{X}$ is not lower triangular matrix, we apply the lower triangular matrix transformation to $\mathbf{L}_\mathbf{X}$ to make it lower triangular matrix.



Adjacency A
$\epsilon \sim N(0, I)$
Input X
Hidden Layer 1
Hidden Layer 2
$(\mathbf{I} - \mathbf{A}^\mathbf{T})\mathbf{H}$
$\mu_\phi$
$\sigma_\phi$
Reparametrization
$\mathbf{Z_0} \sim \mathbf{N}(\mu_\phi, \mathrm{diag}\{\sigma_\phi^2\}_\mathbf{i})$
$\mathbf{L_X}$
Lower Triangular with Diagonal ones
IAF
$\mu_\theta$
$\sigma_\theta$
Hidden Layer 2
Hidden Layer 1
$(\mathbf{I} - \mathbf{A}^\mathbf{T})^{-1}\mathbf{Z_T}$
$\mathbf{Z_T} = \mathbf{L_X Z_0}$

Figure 1: Architecture of the proposed model.

### 4.2 Learning

Our model aims to learn the adjacency matrix of the directed acyclic graph. Instead of regarding the mixed graph as a cyclic graph, we consider them as DAG structure with correlated noise variables. Thus we can use the same optimization procedure as DAG-GNN, which uses the acyclicity constraint as a continuous approximation of the discrete constraint. The optimization procedure is minimizing the following loss function:

$$\mathcal{L}(A, W, \lambda) = -\mathcal{L}_{\text{ELBO}} + \tau\|A\|_1 + \lambda h(A) + \frac{c}{2}|h(A)|^2. \tag{6}$$

The second term is the L1 regularization term, which encourages sparsity of the adjacency matrix. We found that $\tau = 0.1$ works well for a node size of 50. For other node sizes, we use $\tau$ proportional to the inverse of the square of the node size. The third and fourth terms are the augmented Lagrangian terms. We gradually increase the value of $c$ during the training process, as a larger value of $c$ reduces the acyclicity constraint to zero [4].

## 5 Experiment

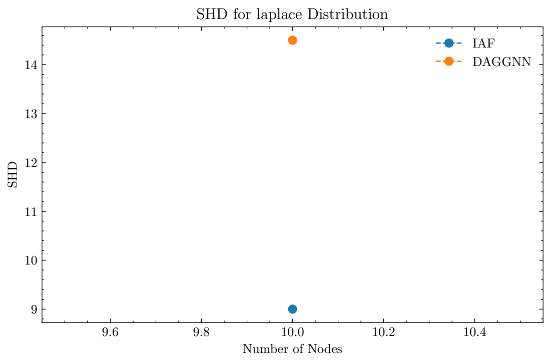
In this section, we conduct experiments on simulated data to evaluate the performance of our proposed method. We compare our method with the DAG-GNN [4] with random graph datasets. We used a thresholding value of extracting graph as 0.3 which is the same value used at DAG-GNN and NOTEARS.

**Datasets** Random graph datasets are generated using the following procedure. First, we generate a random directed acyclic graph using the Erdős–Rényi model. In Section 5.1, we consider the case where the noise variables follow not only an independent Gaussian distribution but also independent Laplace and exponential distributions. In Section 5.2, we conduct experiments where the noise variables follows a multivariate Gaussian distribution with a non-diagonal covariance matrix. We randomly select a given proportion of edges and make them bidirectional. Then, we generate a corresponding random covariance matrix and generate multivariate Gaussian data using that covariance matrix. In both sections, we generate 5000 samples for each graph. For the size of the graph, we use 10, 20, 30, and 50 nodes. For the proportion of bidirectional edges, we use 0.1, 0.3, 0.5, and 0.8.

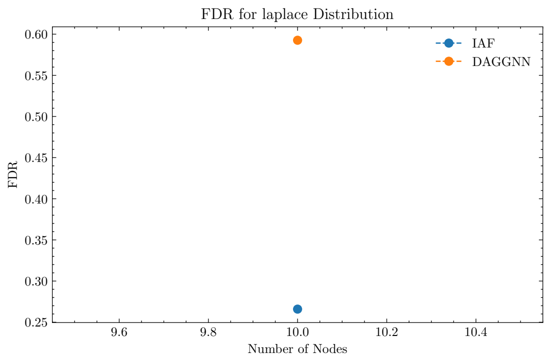
**Evaluation** We evaluate the performance of our method using two metrics: Structural Hamming Distance (SHD) and False Discovery Rate (FDR). SHD measures the number of edge additions, deletions, and reversals required to transform the estimated graph into the true graph. FDR represents the ratio of false positives to the total number of predicted edges. These metrics are calculated by comparing the estimated graph with the true graph, considering bidirectional edges. For each combination of the number of nodes and the proportion of bidirectional edges, we generate at least 5 random graphs and calculate the average metrics.

### 5.1 Independent Noise Case

We compare first the performance of our method with DAG-GNN in the case where the noise variables are independent. To be specific, we consider the case where the noise variables follow an independent Gaussian, Laplace, and exponential distribution. The results of the experiments are shown in Figure **??**, 2, and **??**. In all cases, our method outperforms DAG-GNN in terms of both SHD and FDR.



(a) SHD with Laplace noise

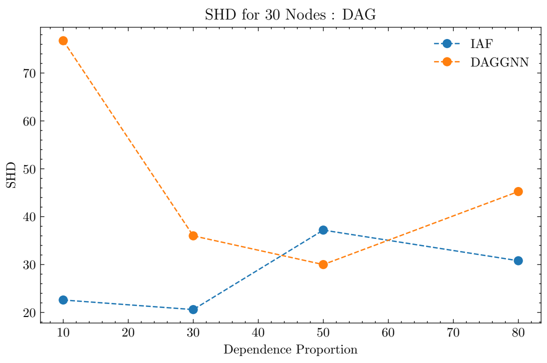


(b) FDR with Laplace noise

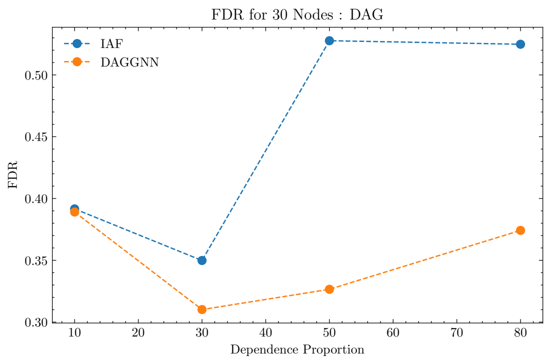
Figure 2: SHD and FDR with Laplace noise

### 5.2 Dependent Noise Case

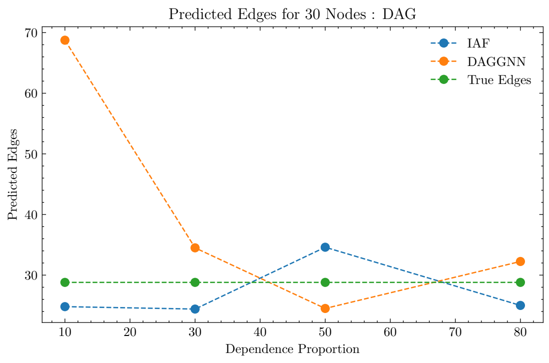
Firstly, we set the proportion of bidirectional edges to 0.5 and compared the performance of our method with that of DAG-GNN. The results, as depicted in Figure 3, demonstrate that our method surpasses DAG-GNN in terms of both SHD and FDR. Notably, with a node size of 20, our method exhibits stable



(a) SHD with dependence proportion 0.3



(b) FDR with dependence proportion 0.3



(c) Number of predicted edges

Figure 3: SHD, FDR, and number of predicted edges with dependence proportion 0.3

training, resulting in a significant difference in SHD and FDR. Figure 4 presents the estimated graphs produced by each method. The graph estimated by DAG-GNN (third column) contains considerably more edges than the true graph (first column), whereas the graph estimated by our method (second column) has a number of edges much closer to that of the true graph.

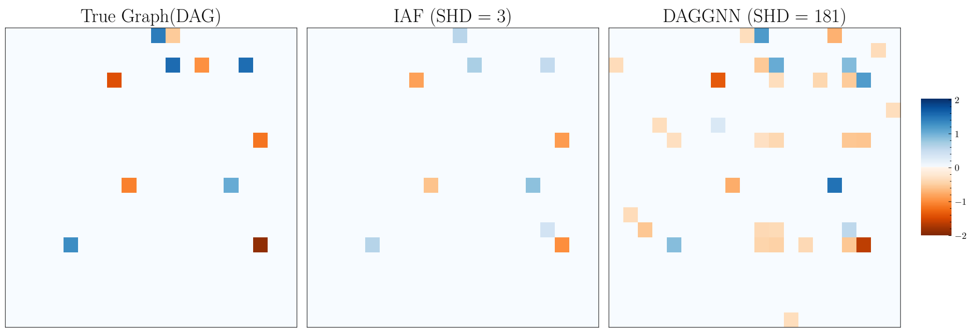
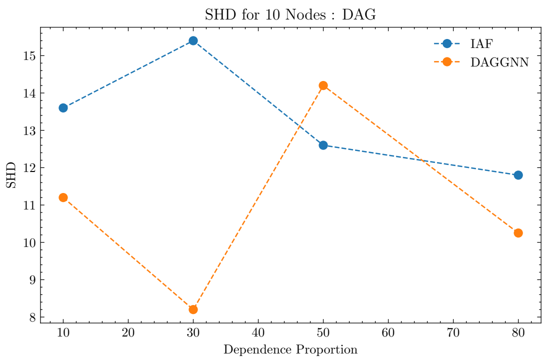
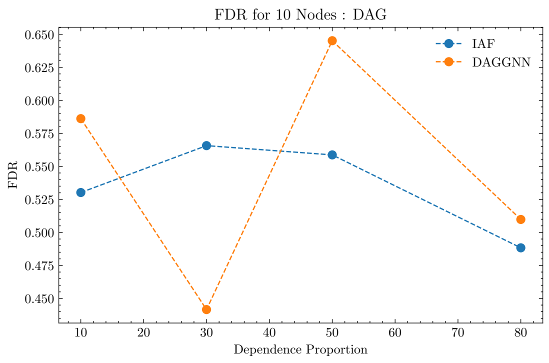


Figure 4: Example of estimated graphs with node size 20 and dependence proportion 0.5

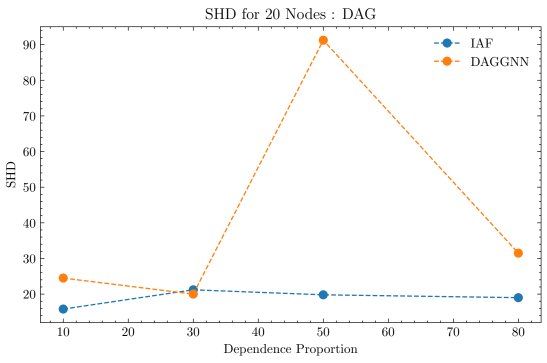
Next, we compare the performance of the two methods, each with varying proportions of bidirectional edges. The results, as illustrated in Figure 5, reveal that when the node size is 10, DAG-GNN surpasses our method in terms of SHD. However, when the node size is increased to 20, our method takes the lead, outperforming DAG-GNN in both SHD and FDR metrics.



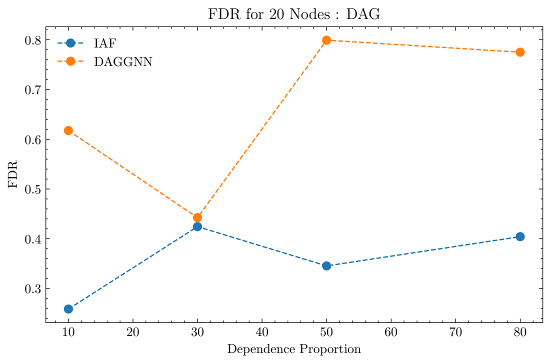
(a) SHD with node size 10



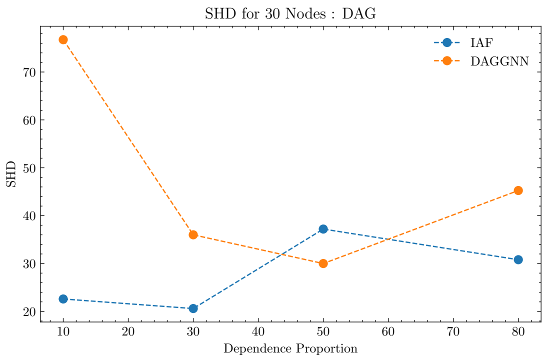
(b) FDR with node size 10



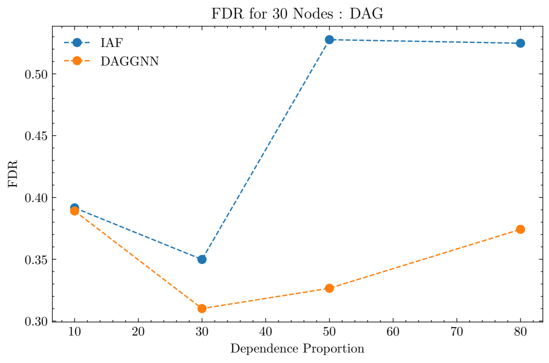
(c) SHD with node size 20



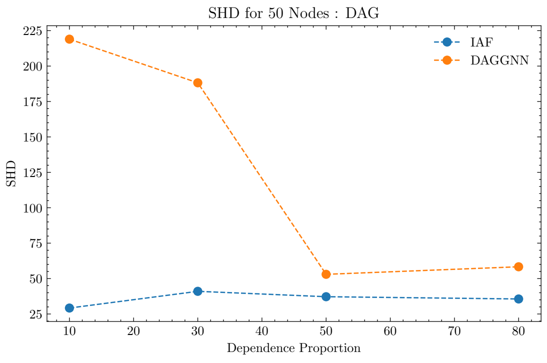
(d) FDR with node size 20



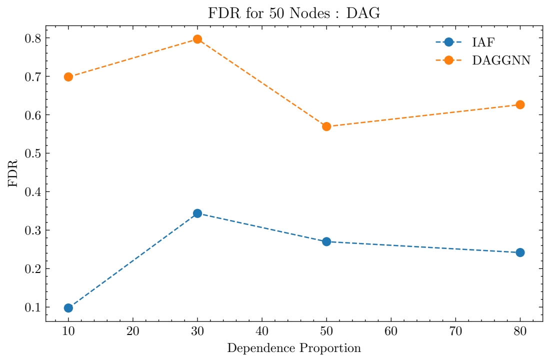
(e) SHD with node size 30



(f) FDR with node size 30



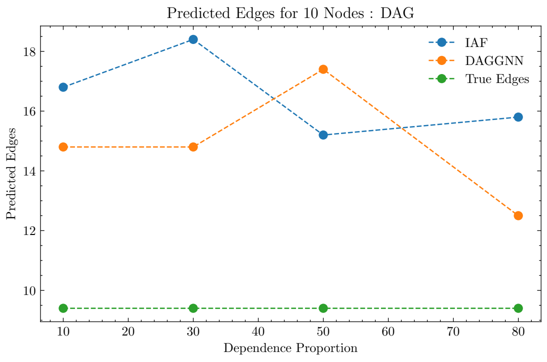
(g) SHD with node size 50



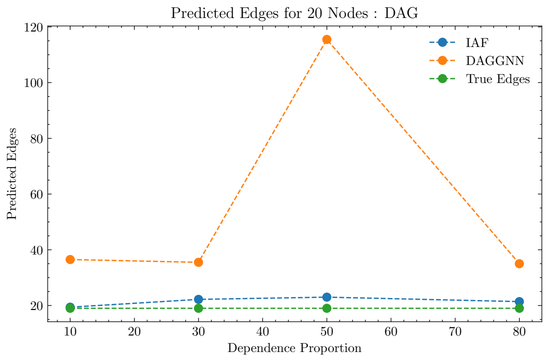
(h) FDR with node size 50

Figure 5: SHD and FDR with dependence proportion 0.1, 0.3, 0.5, and 0.8

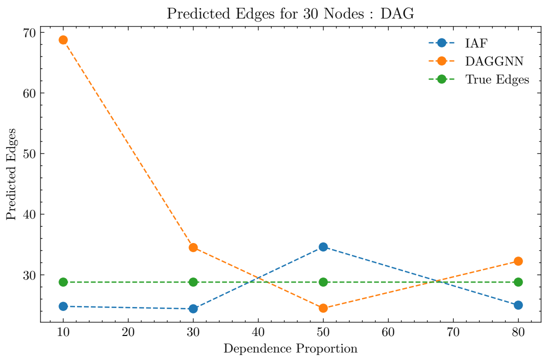
In terms of the number of predicted edges, we observe that our method produces a number of edges closer to the true number of edges than DAG-GNN, especially when the node size is 20. As shown in Figure 6, the number of predicted edges from both methods are similar when the node size is 10. However, when the node size is 20, 30 and 50, our method produces a number of predicted edges closer to the true number of edges than DAG-GNN. Considering that our method outperforms DAG-GNN in terms of SHD and FDR, we can conclude that our method produces a more accurate graph structure than DAG-GNN.



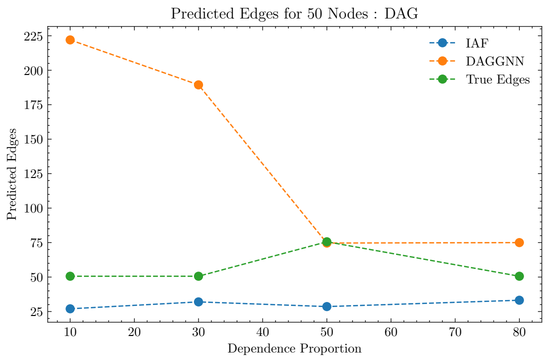
(a) Predicted edges with node size 10



(b) Predicted edges with node size 20



(c) Predicted edges with node size 30



(d) Predicted edges with node size 50

Figure 6: Number of predicted edges with dependence proportion 0.1, 0.3, 0.5, and 0.8

## 6 Conclusion

Here is the conclusion

## References

[1] J. Pearl, *Causality*. Cambridge University Press, 2 ed., 2009.

[2] D. M. Chickering, C. Meek, and D. Heckerman, "Large-sample learning of bayesian networks is np-hard," *CoRR*, vol. abs/1212.2468, 2012.

[3] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing, "Dags with no tears: Continuous optimization for structure learning," 2018.

[4] Y. Yu, J. Chen, T. Gao, and M. Yu, "Dag-gnn: Dag structure learning with graph neural networks," 2019.

[5] J. Li, T. Yu, J. Li, H. Zhang, K. Zhao, Y. Rong, H. Cheng, and J. Huang, “Dirichlet graph variational autoencoder,” 2020.

[6] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” 2016.

[7] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, “D-vae: A variational autoencoder for directed acyclic graphs,” 2019.

[8] I. Shpitser and J. Pearl, “Identification of joint interventional distributions in recursive semi-markovian causal models,” in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI’06, p. 1219–1226, AAAI Press, 2006.

[9] D. Rezende and S. Mohamed, “Variational inference with normalizing flows,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1530–1538, PMLR, 07–09 Jul 2015.

[10] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[11] J. M. Tomczak and M. Welling, “Improving variational auto-encoders using householder flow,” 2017.

[12] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, “Improved variational inference with inverse autoregressive flow,” in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.