Advancement of Structure Learning Methods for Time Series Data

Valen Potloff vpotloff@ucsd.edu
Tyler Gibbs tgibbs@ucsd.edu
Biwei Huang bih007@ucsd.edu
Babak Salimi bsalimi@ucsd.edu

Abstract

We examine the problem of learning graphical structures characterized by Directed Acyclic Graphs (DAGs) on time series data. DYNOTEARS adapts the methods used in NOTEARS to learn linear time series DAGs using a least squares score function. Recently, methods such as GOLEM improve the performance in the time-static linear case by deriving a score function via maximum likelihood. Further improvements have been made by DAGMA, adding support for learning non-linear relationships as well. We introduce two new methods, GOLEM-TS and DAGMA-TS, which adapt the findings from GOLEM and DAGMA to the time series case. We find that this adaption improves performance compared to DYNOTEARS.

Code: https://github.com/tylergibbs/dsc180ProjectB.git

1 Introduction

1.1 Structure Learning

Learning Directed Acyclic Graphs (DAGs) from data is a significant area of research in the field of machine learning due to it's applications many scientific fields. These constraints can be applied to a wide variety of situations, from proteins expression(Sachs et al. 2005), to general healthcare(Lucas, Gaag and Abu-Hanna 2004). Under certain conditions, these graphical structures may have causal interpretations. Learning these DAGs is an NP-hard problem Chickering (2000) primarily as a result of enforcing acyclically on the resultant graph. Methods for discovering DAGs generally are either constraint-based or score-based. Constraint based methods such as the PC algorithm rely on conditional independence tests and the faithfulness assumption to discover DAGs. Score-based methods approach the problem by optimizing a score function that quantifies the performance of a graph model based on data. The score-based approach allows the use of continuous optimization methods to learn a DAG represented by a weighted adjacency matrix usually notated as W.

1.2 Structure Learning SEM

In order to use a score-based approach to learn a DAG, we must define a Structural Equation Model (SEM). Let $W \in \mathbb{R}^{d\times d}$ represent the weighted adjacency matrix of a graph with d nodes. Let $X \in \mathbb{R}^{n\times d}$ represent the data matrix, where d is the size of the data vectors, and n is the number of data vectors, or observations. Let $Z \in \mathbb{R}^{n\times d}$ represent a noise matrix. We can formulate a (linear) SEM as follows:

$$X = XW + Z \tag{1}$$

A matrix W that satisfies (1) represents the weighted adjacency matrix of the directed graph that captures the dependency relationships between columns of X. Under further assumptions, the graphical structure of W may have causal interpretations.

1.3 Score Function

In order to learn W that satisfies (1), we must formulate a score function that captures the performance of W relating X to itself. It is easy to see that least-sqares loss between X and XW can quantify the performance of W. Therefore, the least-squares loss takes the form $l(W;X) = \frac{1}{2n}||X-XW||^2$. In addition to least-squares loss, we will introduce l_1-regularization on W so the learned graph is sparse. We can write our regularized score function as follows:

$$F(W) = l(W;X) + \lambda||W||_1 = \frac{1}{2n}||X - XW||^2 + \lambda||W||_1 \tag{2}$$

$l(W;X)$ need not be least-sqaures, but can be logistic loss, or other alternate formulations which will be discussed. The graph learned from minimizing $F(W)$ is not guaranteed to be

acyclic, and thus we must constrain the graph represented by W to be a DAG, or $G(W) \in \mathbb{D}$, where $\mathbb{D}$ is the space of all DAGs. Therefore, the full optimization problem is as follows:

$$\begin{aligned} &\min_{W \in \mathbb{R}^{d \times d}} F(W) \\ &\text{subject to } G(W) \in \mathbb{D} \end{aligned} \tag{3}$$

The central question is then: how do we enforce $G(W) \in \mathbb{D}$? The approaches we will discuss derive different forms of the function $h : \mathbb{R}^{d \times d} \to \mathbb{R}$ such that $h(W) = 0$ if and only if $G(W)$ is a DAG. The following sections will discuss three different approaches to the minimization procedure in (3): NOTEARS, GOLEM, and DAGMA.

1.4 NOTEARS

NOTEARS (Zheng et al. (2018)) uses the objective function with least-sqares loss and l_1-regularization shown in 2. The authors formulate an algebraic characterization of the acyclity constraint of the following form:

$$h(W) = tr(e^{W \circ W}) - d \tag{4}$$

where tr is the matrix trace, e^A is the matrix exponential of matrix A, and $\circ$ is the Hadamard (element-wise) product. The NOTEARS optimization problem then takes the following form:

$$\begin{aligned} &\min_{W \in \mathbb{R}^{d \times d}} F(W) = \frac{1}{2n} ||X - XW||^2 + \lambda ||W||_1 \\ &\text{subject to } h(W) = 0 \end{aligned} \tag{5}$$

The optimization problem in 5 is solved using the continuous optimization based L-BFGS-B (Bollapragada et al. (2018)) procedure, with the acycliyty constraint enforced using the augmented Lagrangian method (Burman, Hansbo and Larson (2022)). The NOTEARS method has been shown to produce results close to the ground truth graph.

1.5 GOLEM

According to the authors of GOLEM (Ng, Ghassami and Zhang (2021)), the so-called "hard" DAG constraint introduced by Zheng et al. (2018) in NOTEARS can be subject to numerical difficulties and issues with ill-conditioning as the penalty coefficent used in the augmented Lagrangian method must approach infinity in order to enforce acyclity. The parameters of the augmented Lagrangian method require careful-fine tuning in order to prevent this. In order to improve performance compared to NOTEARS, Ng, Ghassami and Zhang (2021) introduce GOLEM, which formulates the objective function $F(W)$ via maximum likelihood. Consequently, the GOLEM objective function only requires a "soft" acyclity constraint.

If we assume the nose vector Z in 1 is characterized by an independent Gaussian distribution with non-equal noise variances, the objective function derived via maximium likelihood is

as follows:

$$L_1(W;X) = \frac{1}{2}\sum_{i=1}^{d}\log(\sum_{k=1}^{n}(X_{ki} - X_k W_i^T)^2) - \log|\det(I-W)| \tag{6}$$

If we assume equal noise variances, the objective function becomes the following:

$$L_2(W;X) = \frac{d}{2}\log(\sum_{i=1}^{d}\sum_{k=1}^{n}(X_{ki} - X_k W_i^T)^2) - \log|\det(I-W)| \tag{7}$$

We can see that the above formulations are equivalent to the least squares loss introduced in NOTEARS, with an added term (called the logdet term). The authors of GOLEM point out that the logdet term helps to enforce acyclity, as "If a weighted matrix $B \in \mathbb{R}^{d\times d}$ represents a DAG, then $\log|\det(I-B)| = 0$". It is important to note that the previous proposition only goes one way. Namely, just because $\log|\det(I-B)| = 0$ doesn't mean that B is a DAG.

Given the objective funtion L_i ($i = 1$ if we assume non-equal noise variances, $i = 2$ if we assume equal noise variances), we can formulate the full minimization procedure as follows:

$$\min_{W\in\mathbb{R}^{d\times d}} S_i(W;X) = L_i(W;X) + \lambda_1||W||_1 + \lambda_2 h(W) \tag{8}$$

The algebraic characterization of acyclity ($h(W)$) is the same as the one found in NOTEARS (4). Notice how the GOLEM score function does not use the augmented Lagrangian method to enforce acyclity, but instead enforces it using a soft constraint. The authors of GOLEM show that this objective function has better performance compared to NOTEARS. GOLEM also allows us to make assumptions about the noise variances. The authors of Ng, Ghassami and Zhang (2021) note that the optimization of L_1 is subject to local solutions, so it is best to initialize W with the solution found via the minimization of L_2.

1.6 DAGMA

DAGMA (Bello, Argam and Revikumar (2023)) builds upon GOLEM and NOTEARS by translating the logdet term found in GOLEM into an exact acyclity charaterization, via the use of M-matricies. The authors of DAGMA define an M-matrix as follows:

Definition An M-matrix is a matrix $A \in \mathbb{R}^{dxd}$ of the form $A = sI - B$, where $B \geq 0$ and $s > \rho(B)$.

Where s is a scalar constant and $\rho(B)$ is the spectral radius of matrix B.

From this, the authors find the acyclicity constraint h^s_{ldet} that is defined as follows:

$$h^s_{ldet}(W) = -\log\det(sI - W\circ W) + d\log s \tag{9}$$

The authors of Bello, Argam and Revikumar (2023) state that h^s_{ldet} does not diminish cycles of any length, it has better behaved gradients, and h^s_{ldet} and its gradient are faster to

compute compared to other acyclity constraints. This makes the h^s_{ldet} acyclity constraint preferrable to 4.

The authors show that using log-least-squares loss, l_1-regularization, and h^s_{ldet} as a soft acyclity constraint improve performance compared to GOLEM and NOTEARS.

1.7 Learning Nonlinear Relationships

So far we have only discussed learning DAGs with linear edges captured by the SEM in 1. Zheng et al. (2019) formulate a generalized SEM of the following form:

$$X_j = f_j(X) + Z \tag{10}$$

In the case of a linear SEM, $f_j(X) = XW_j^T$. This general form allows us to consider the edges of a graph that represents the relationships between variables of X as an arbitrary function. For our purposes, we will only consider the case where edges are represented as Multi-layer Perceptrons (MLPs), or $f = MLP$. Since MLPs are considered to be universal function approximators (Hornik, Stinchcombe and White (1989)), the use of MLPs to capture edges in a graph allows us to learn relationships of an arbitrary form.

We are then left with two questions: 1) How do we find the graph from the form $f_j = MLP$?, and 2) How do we enforce acyclity? Zheng et al. (2019) show that f_j is independent of X_k if and only if $||\partial_k f_j||_2 = 0$, where ∂_k is the partial derivative with respect to X_k. We can then construct a matrix of the following form:

$$[W(f)]_{kj} = ||\partial_k f_j||_2 \tag{11}$$

The matrix $W(f) \in \mathbb{R}^{d\times d}$ captures the dependence between the columns of X, and therefore represents the weighted adjacency matrix, from which we can apply our acyclity constraint and recover a DAG.

What is $W(f)$ for the case of $f = MLP$? Let $A_j^{(1)}$ represent the weights of the first layer of the jth MLP (f_j). It can be shown that if $||\text{kth-column}(A_j^{(1)})||_2 = 0$, f_j is independent of X_k. Therefore, we can write the following:

$$[W(f)]_{kj} = ||\text{kth-column}(A_j^{(1)})||_2 \tag{12}$$

Using 11, we can enforce acyclity and recover the generated DAG. Furthermore, we can enforce sparsity on the resultant graph by adding l_1-regularization to $A_j^{(1)}$ for all j. We can therefore minimize an objective function with least-squares loss and enforce acyclity on $W(f)$ via the augmented Lagrangian method, or in the case of DAGMA, log least-squares loss and a soft acyclity constraint.

1.8 Time Series Data: DYNOTEARS

So far, we have only dealt with instantaneous relationships, failing to consider the influence of time lagged versions of X (X_{t-s}) on X (X_t). DYNOTEARS Pamfil et al. (2020) adapts the

findings of NOTEARS Zheng et al. (2018) to the case of time series data via the following:

Consider data matrix $X_t \in \mathbb{R}^{n\times d}$. Where X is the observation of X at time t. We can construct matrix $Y \in \mathbb{R}^{n\times pd}$ such that $Y = [X_{t-1}|X_{t-2}|...X_{t-p}]$, being a wide matrix containing time-lagged versions of X. Note that the time-lagged versions of X need not be sequential and may be at any point in time before t.

To create a SEM relating X and Y, define $W \in \mathbb{R}^{d\times d}$ to be the weighted adjacency matrix representing the relationship between columns of X. Define $A \in \mathbb{R}^{pd\times d}$ to be $A = [A_1^T|A_2^T|...A_p^T]^T$ where A_i represents the relation from X_{t-i} to X_t, or in other words the relation from X at a previous timestep to the X at the current timestep. With A representing the relation from all p timesteps to the current timestep. With a random noise matrix $Z \in \mathbb{R}^{n\times d}$, we can construct the SEM as follows: follows:

$$X = XW + YA + Z \tag{13}$$

The authors of Pamfil et al. (2020) find W and A using methods from NOTEARS. Namely, using least-sqaures loss, l_1-regularization, and the augmented lagrangian hard acyclity constraint on W from NOTEARS. There is no need to enforce acyclity on A as the weights from A only go forward in time and thus do not contribute to cycles.

The objective function of DYNOTEARS suffers the same problems as NOTEARS due to the hard acyclity constraint. Additionally, DYNOTEARS only considers linear relationships, and does not take into account non-linear relationships as in Zheng et al. (2019). We seek to improve the performance of DYNOTEARS by adapting improvments from GOLEM and DAGMA to the time-series case. Additionally, we introduce a method for finding non-linear relationships of time-series data.

2 Methods

We introduce two new methods for learning DAGs on time-series data: **DAGMA-TS** and **GOLEM-TS**. GOLEM-TS uses the objective function found in Ng, Ghassami and Zhang (2021) to learn a graph with instantaneous and time-lagged relationships with a soft acyclity constraint. DAGMA-TS uses the Bello, Argam and Revikumar (2023) soft acyclity constraint to learn instantaneous and time-lagged relationships. Additonally, we adapt DAGMA-TS to the nonlinear case in Zheng et al. (2019) with the use of MLPs.

2.1 Alternate formulation of time-series SEM

In order to adapt the DYNOTEARS SEM to the methods of GOLEM and DAGMA, we must modify the following SEM:

$$X = XW + YA + Z$$

Define n as number of data vectors, d as the dimensionality of the data, and p as the autoregressive order. Therefore $X \in \mathbb{R}^{n \times d}$ represents the data matrix, $Y \in \mathbb{R}^{n \times pd}$ represents the data matrix of time-lagged versions of X, $W \in \mathbb{R}^{d \times d}$ represents the time static weights of the graph, $A \in \mathbb{R}^{pd \times d}$ represents the time lagged weights of the graph, and $Z \in \mathbb{R}^{n \times d}$ represents the additive noise matrix. Let $d' = (p+1)d$, $A' = [W^T | A^T]^T \in \mathbb{R}^{d' \times d}$, and $Y' = [X|Y] \in \mathbb{R}^{n \times d'}$. We can rewrite the SEM as follows:

$$X = Y'A' + Z \tag{14}$$

2.2 DAGMA-TS

Given 14, we can use the methods of DAGMA to learn A', and apply the acyclity constraint to only the W component of A'. We can also learn a generalized SEM of the form $X_j = f_j(Y')$. In our case, we only consider $f = MLP$ to learn nonlinear edge weights. Concretely, the optimization problem of (linear) DAGMA-TS is the following:

$$\min_{A' \in \mathbb{R}^{d \times d}} \frac{1}{2n} log(||X - Y'A'||^2) + \lambda_1 ||A'||_1 + \lambda_2 h^s_{ldet}(W) \tag{15}$$

In the non-linear case of DAGMA-TS, we use the same procedure as in Zheng et al. (2019) and Bello, Argam and Revikumar (2023) to recover the generated DAG and enforce acyclity.

2.3 GOLEM-TS

Given the SEM in 14, one may think that we can find a function L that represents the log-likelihood of the data given A'. However, this approach is fruitless, due to the fact that A' is a rectangular matrix with more rows than columns and no matrix of that form has a right inverse. Instead, we will transform the matrix A' via the following:

Define $E \in \mathbb{R}^{d' \times pd}$, and $B = [A'|E] \in \mathbb{R}^{d' \times d'}$. We can then redefine the SEM as the following:

$$Y' = Y'B + Z \tag{16}$$

Since B is a square matrix, we can apply the methods of GOLEM to learn B. E represents weights going backwards in time, which we must not consider to be a possibility. In order to prevent GOLEM from learning weights backwards in time, E must be zero-valued. To enforce this, we constrain the objective function such that $||E|| = 0$.

We will skip the full derivation of the objective function for GOLEM-TS. For a detailed proof of the following objective, see Ng, Ghassami and Zhang (2021). Following the same steps as GOLEM but for the SEM in 16, we get the following:

$$L_1(B;X) = \frac{1}{2} \sum_{i=1}^{d'} \log(\sum_{k=1}^{n} (Y_{ki} - Y_k B_i^T)^2) - \log|\det(I - B)| \tag{17}$$

The above objective is the same as 6, but with Y instead of X and B instead of W. We find better performance when replacing the least squares component of 17 with the least squares component from DYNOTEARS, resulting in the following:

$$L_1(B;X) = \frac{1}{2}\sum_{i=1}^{d}\log(\sum_{k=1}^{n}(X_{ki} - Y_k A_i^T)^2) - \log|\det(I-B)| \tag{18}$$

And assuming equal noise variances:

$$L_2(B;X) = \frac{d}{2}\log(\sum_{i=1}^{d}\sum_{k=1}^{n}(X_{ki} - Y_k A_i^T)^2) - \log|\det(I-B)| \tag{19}$$

So the full optimization problem of GOLEM-TS is the following:

$$\min_{B\in\mathbb{R}^{d'\times d'}} S_i(B;X) = L_i(B;X) + \lambda_1||A||_1 + \lambda_2 h(W) + \lambda_3||E||_1 \tag{20}$$

Notice how we have seperate l_1-regularization terms for the different components of B, this is because we want $||E||_1 = 0$, but we just want A to be sparse, meaning that the regularization factor for E (λ_3) will be much bigger than that of A (λ_1). In 20, when $i = 1$, the model is the non-equal variance version, or **GOLEM-TS-NV**, and is **GOLEM-TS-EV** when $i = 2$.

2.4 Experiments

The following section details the different models and experimental setups used to test our methods, post processing techniques, hyperparameter tuning of all our new methods, and implementation details.

2.4.1 Experimental Setup

We introduced various methods in this paper. **DAGMA-TS** has a linear version (which we will just call **DAGMA-TS**) and a non-linear (MLP) version, or **DAGMA-TS-NL**. **GOLEM-TS** also has two versions, one assuming equal noise variance, or **GOLEM-TS-EV**, and one assuming non-equal noise variances, or **GOLEM-TS-EV**. Overall, we have introduced for distinct methods: **DAGMA-TS**, **DAGMA-TS-NL**, **GOLEM-TS-EV**, and **GOLEM-TS-NV**. We will compare all of these methods, and use **DYNOTEARS** (Pamfil et al. (2020)) as our baseline.

To compare the performance between our methods and DYNOTEARS, we will generate various Erdős-Rényi (ER) ground-truth graphs using the same time-series sampling procedure as in Pamfil et al. (2020). Since we have developed methods for both linear and non-linear edges, we will generate ground truth graphs with both linear and MLP generated edges. The MLPs for each edge have 1 hidden layer with a sigmoid activation and linear output activation. We will test our linear models (DYNOTEARS, GOLEM-TS-EV, GOLEM-TS-NV, DAGMA-TS) on the linear graphs, and ALL of our models including DYNOTEARS on the non-linear graphs.

For both the linear and non-linear graphs, we will fix our autoregressive order (p) to 1. Following Pamfil et al. (2020), we will generate graps of degree 2 and 4, labeled *ER2* and *ER4* respectively, with the number of nodes $d \in \{5, 10, 20, 50, 100\}$. For each graph, we will simulate a dataset following the same procedure as Pamfil et al. (2020), generating datasets of size $n \in \{50, 500\}$, with different types of additive noise (Gaussian-EV, Gaussian-NV, Exponential, and Gumbel).

For each combination of graph and model, we will repeat the process 10 times to compute average performance. We compare the performance of our generated DAG to the ground truth DAG based on Structural Hamming Distance (SHD), True Positive Rate (TPR), False Discovery Rate (FDR), False Positive Rate (FPR), and Runtime measured in seconds.

2.4.2 Post Processing

Due to the numerical stability introduced by the acyclity constraint, post proccessing of the learned graph is required before we can compare it to the ground truth DAG. Most previous approaches discussed so far perform a thresholding step, where all absolute edge weights below a certain threshold are set to zero. In the various approaches discussed, this value can range from 0.01 to 0.3. For all of our own methods, we set this value $\omega = 0.2$, while for DYNOTEARS we keep it the same at $\omega = 0.01$.

To ensure that the generated graph is a DAG, an additional thresholding step is performed, where if the graph is not a DAG, the smallest edges are removed until the graph is acyclic.

2.4.3 Hyperparameter Tuning

In order to ensure optimal performance, we performed 5-fold cross validation to find the optimal values of hyperparameters for each method.

For DYNOTEARS, we did not perform any tuning as we felt the hyperparameters found in Pamfil et al. (2020) were sufficent.

For both verions of GOLEM-TS, we found that $\lambda_1 = 0.01$, $\lambda_2 = 1.0$, and $\lambda_3 = 5.0$ resulted in optimal performance.

For DAGMA-TS (linear), we found the optimal hyperparameters to be $\lambda_1 = 0.05$ and $\lambda_2 = 0.01$. For DAGMA-TS-NL, we found that $\lambda_1 = 0.02$ and $\lambda_2 = 0.005$ to be optimal. We also found that 1 hidden layer of size 20 achieved the desired performance without sacrificing runtime.

For GOLEM-TS-EV, we train for 50,000 epochs, with early stopping for when the training loss bottoms out. For GOLEM-TS-NV, we first train for a maximum of 20,000 epochs using the equal variances objective function, and then train for a maximum of 30,000 epochs using the non-equal variances objective function, with early stopping. For DAGMA-TS, we use the same training procedure described in Bello, Argam and Revikumar (2023), and therefore train for the same number of epochs.

2.4.4 Implementation Details

For DYNOTEARS, we use the implementation referenced in Pamfil et al. (2020), which was implemented in PyTorch Paszke et al. (2017). All methods introduced here were implemented in PyTorch. GOLEM-TS and DAGMA-TS follow the same respective optimization procedures as shown in their original time-static implementations. For more details, we refer the reader to Ng, Ghassami and Zhang (2021) and Bello, Argam and Revikumar (2023).

Due to our methods being implemented from scratch in PyTorch, we were able to create models that support GPU acceleration, drastically decreasing the runtime of our experiments. DYNOTEARS' off the shelf implementation does not support GPU acceleration, so we run this on a CPU, the same way it was done in Pamfil et al. (2020). Our methods were run on a Nvidia GeForce GTX 1080ti, while DYNOTEARS experiments were performed on an 8-core Intel CPU with 16 GB of memory.

3 Results

The following section details the results of the experiments described in the previous section. For full results, look to B.

3.1 Linear Case

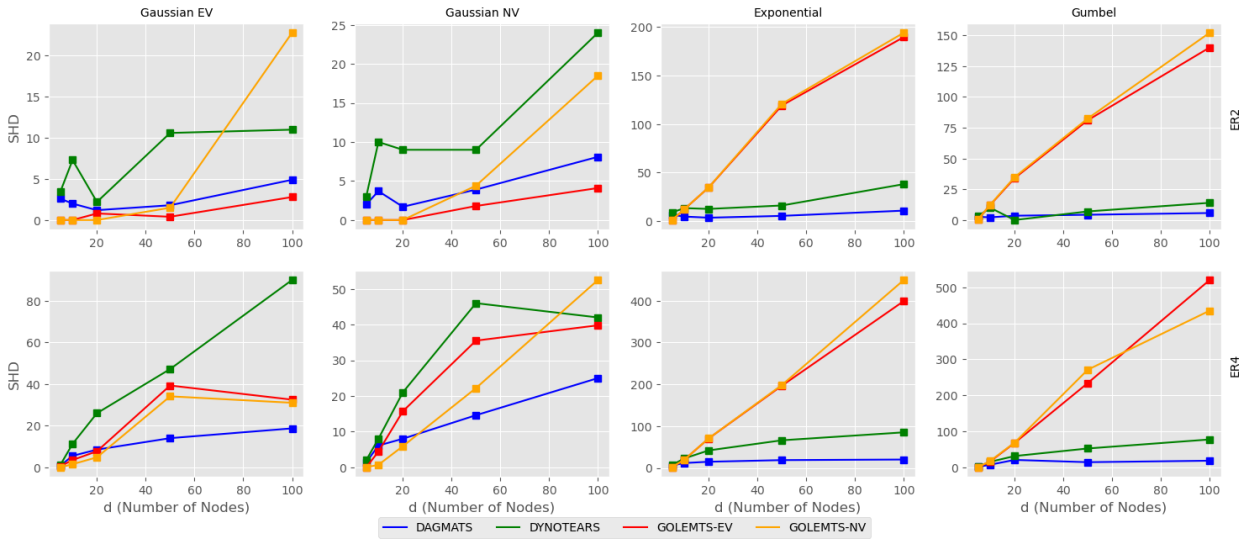

Figure 1: Structural Hamming Distance (SHD) for linear ground truth graphs with $n = 500$. Lower is better. DAGMA-TS appears to have the best performance. It also seems that the methods from GOLEM-TS are ill-conditioned for exponential and gumbel noise.

3.2 Non-linear Case

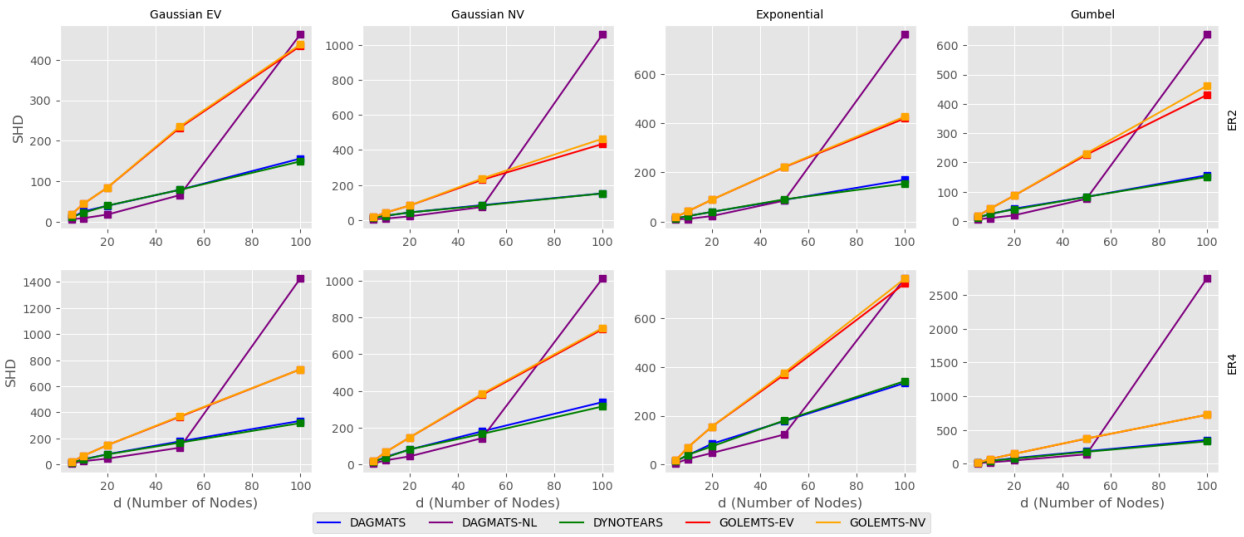

Figure 2: Structural Hamming Distance (SHD) for non-linear ground truth graphs with $n = 500$. Lower is better. DAGMA-TS-NL seems to have the best performance, except for when $d = 100$.

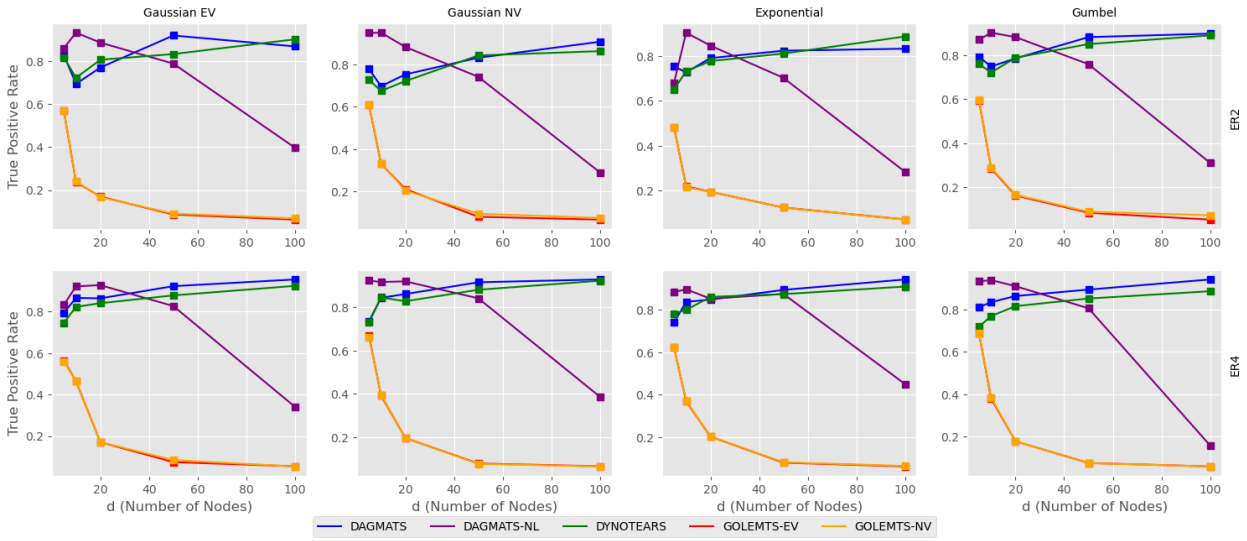

Figure 3: True Positive Rate (TPR) for non-linear ground truth graphs with $n = 500$. Higher is better. DAGMA-TS-NL seems to have the best performance, except for when $d = 100$.

4 Applications

This section details our applications of our methods to real world datasets, specifically S&P 100 Stock Prices.

4.1 S&P 100 Dataset

Following Pamfil et al. (2020), we use our best performing method, **DAGMA-TS** learn both instantaneous and time-lagged relationships between the daily closing prices of S&P 100 tickers. Following DYNOTEARS, we take the log difference between daily closing prices, and center the data so that the mean is zero. For the time-series component, we considered an autoregressive order of $p = 1$, taking into account the previous day's normalized returns.

Similar to DYNOTEARS, the weights of time-lagged relationships were negligible, meaning past price movement has no indication of future movement, reinforcing the efficent market hypothesis (Fama (1970)), similar to DYNOTEARS, it was found that there is an increased concentration of relations within sectors, which we have shown by ordering the tickers on the graph by their respective industries in 6. It appears that the graph learned by DAGMA-TS has less edges compared to the one generated by DYNOTEARS, suggesting that DAGMA-TS learns only meaningful edges.

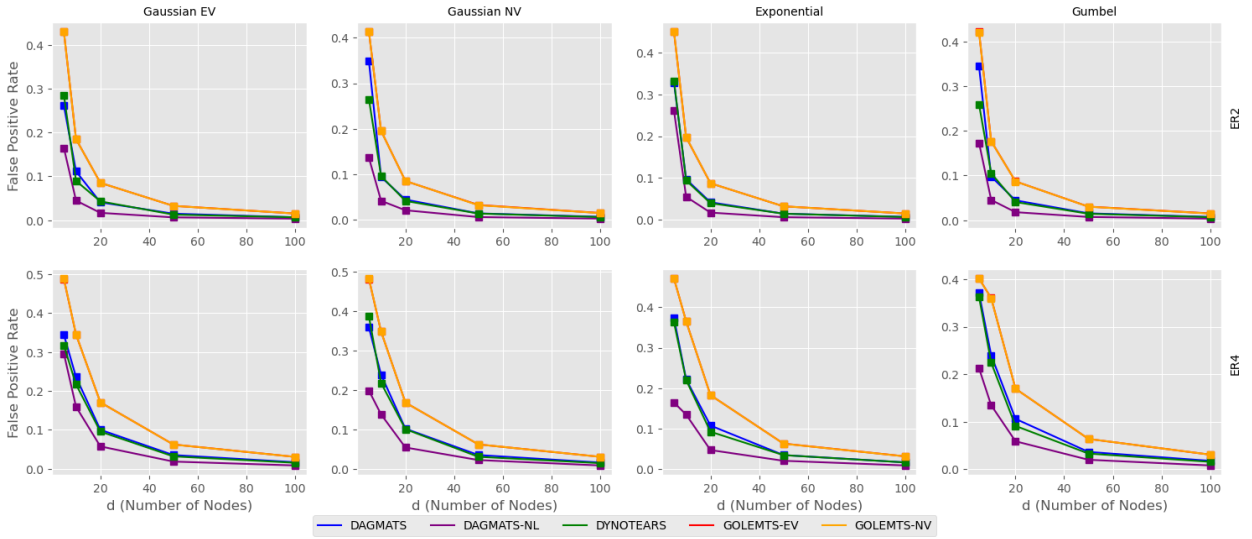

Figure 4: False Positive Rate (FPR) for non-linear ground truth graphs with $n = 500$. Lower is better. DAGMA-TS-NL seems to have the best performance overall, suggesting that compared to DYNOTEARS, DAGMA-TS-NL produces sparser graphs with a low rate of false positives.

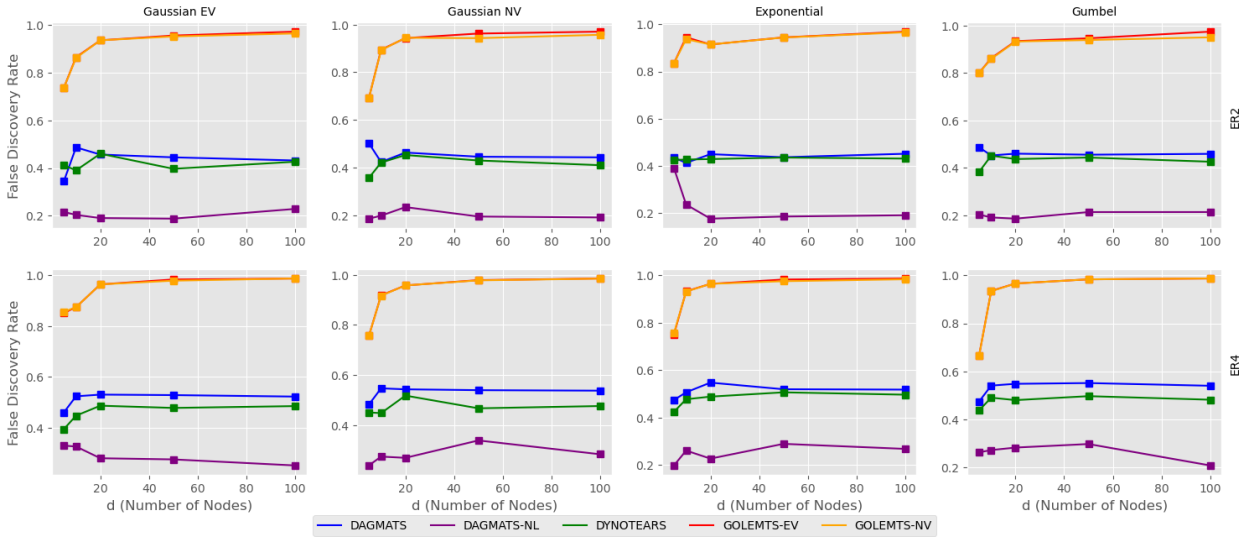

Figure 5: False Discovery Rate (FDR) for non-linear ground truth graphs with $n = 500$. Lower is better. DAGMA-TS-NL seems to have the best performance overall, suggesting that compared to DYNOTEARS, DAGMA-TS-NL produces sparser graphs with a low rate of false positives.

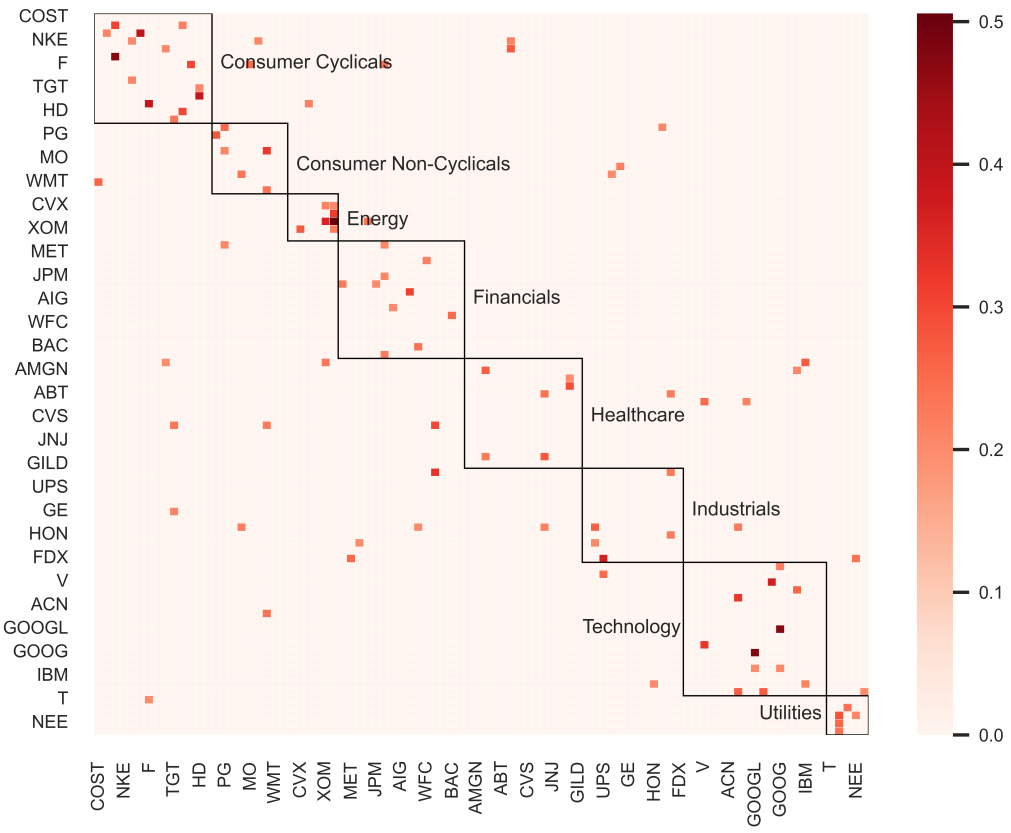

Figure 6: Estimated instantaneous graph of the S&P 100 Dataset. Time-lagged weights were found to be negligible or zero.

5 Conclusion

In this paper, we have shown how we can take advancements in structure learning from Bello, Argam and Revikumar (2023) and Ng, Ghassami and Zhang (2021) and apply them to the case of learning instantaneous and time-lagged relations introduced in Pamfil et al. (2020). We have introduced four new methods: **GOLEM-TS-EV**, **GOLEM-TS-EV**, **DAGMA-TS**, and **DAGMA-TS-NL**. The two methods of GOLEM-TS allow us to learn instantaneous and time-lagged edges using a soft acyclity constraint, and allow us account for the possibility of non-equal noise variances. DAGMA-TS also allows us to use a soft, better conditioned acyclity constraint, as well as learn non-linear edges with the use of MLPs.

In 1, we can see that all of our methods achieve lower structural hamming distance compared to DYNOTEARS. However, in the case of exponential and gumbel noise, while DAGMA-TS improves performance, the methods of GOLEM-TS perform significantly worse, suggesting that GOLEM-TS is ill-conditioned for non-gaussian noise.

For the linear case, we also examine the performance of all methods when the datset is small, in this case $n = 50$. See A 6. All of our methods perform worse than DYNOTEARS, which suggests that our methods are ill-conditioned for small datasets. Although encountering such small datasets is unlikely, it is still an important consideration when deciding which method to use.

It is important to note that in the linear case, all of our methods have significantly lower runtime than DYNOTEARS (A 5 and A 10). This is likely due to our methods taking advantage of GPU acceleration, while DYNOTEARS does not have that capability.

In the non-linear case, we compare DAGMA-TS-NL to the linear DYNOTEARS, as well as our other linear models. While DAGMA-TS-NL outperforms all of our methods and DYNOTEARS in terms of SHD when $d \leq 50$, the performance is significantly worse when $d = 100$ (2). This could be due to the possibility that the traning procedure and hyperparameters for DAGMA-TS-NL may need to be modified for larger values of d. However, DAGMA-TS-NL acheives lower False Positive and False Discovery Rates than all other methods (4, 5), suggesting that DAGMA-TS-NL finds sparser graphs with high accuracy compared to DYNOTEARS and other linear methods. The results are the same for smaller datasets when $n = 50$ (A 16, A 17, A 18, A 19,). For larger datasets ($n = 500$), DAGMA-TS-NL has higher runtime than DYNOTEARS (A 15), but for smaller datasets, the results are mixed (A 20).

Overall, we conclude that, in the linear case, DAGMA-TS is the optimal choice for learning instantaneous and time-lagged relationships from data, but falls short on smaller datasets. In the non-linear case DAGMA-TS-NL is the optimal choice for learning non-linear instantaneous and time-lagged relationships from data.

6 Discussion

In this section, we discuss limitations of our methods, and potential areas of future work.

Static Structure As pointed out in Pamfil et al. (2020), our methods assume the underlying structure of the time-series data is static throughout the entire time series. Future work may examine the relaxation of these assumptions, allowing the structure to change over time according to the data.

Noise In all of our models, we assume the noise is additive in nature. Future work may exaimine the use of different noise models. Additonally, the methods of GOLEM-TS assume Gaussian noise, and do not perform well when that is not the case. Future work may derive maximum likelihood objective functions assuming different types of noise.

Maximum Likelihood Derivation For GOLEM-TS, we were unable to derive a maximum likelihood objective function for the SEM shown in 14, and instead had to create a square matrix B to make our SEM compatible with the methods outlined in Ng, Ghassami and Zhang (2021). Future work may examine the derivation of a maximum likelihood objective function from 14, or alternate ways of modifying 14 such that it is compatible with Ng, Ghassami and Zhang (2021).

References

Bello, Kevin, Byron Argam, and Pradeep Revikumar. 2023. "DAGMA: Learning DAGs via M-matricies and a Log-Determinant Acyclicity Characterization."

Bollapragada, Raghu, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. 2018. "A Progressive Batching L-BFGS Method for Machine Learning."

Burman, Erik, Peter Hansbo, and Mats G. Larson. 2022. "The augmented Lagrangian method as a framework for stabilised methods in computational mechanics."

Chickering, David. 2000. "Learning Bayesian Networks is NP-Complete." *Networks* 112. [Link]

Fama, Eugene F. 1970. "Efficient Capital Markets: A Review of Theory and Empirical Work." *The Journal of Finance* 25 (2): 383–417. [Link]

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer feedforward networks are universal approximators." *Neural Networks* 2 (5): 359–366. [Link]

Lucas, Peter J., Linda C. Gaag, and Ameen Abu-Hanna. 2004. "Bayesian Networks in biomedicine and health-care." *Artificial Intelligence in Medicine* 30: 201–214. [Link]

Ng, Ignavier, AmirEmad Ghassami, and Kun Zhang. 2021. "On the Role of Sparsity and DAG Constraints for Learning Linear DAGs."

Pamfil, Roxana, Nisara Sriwattanaworachai, Shaan Desai, Philip Pilgerstorfer, Paul

Beaumont, Konstantinos Georgatzis, and Bryon Aragam. 2020. “DYNOTEARS: Structure Learning from Time-Series Data.”

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. “Automatic differentiation in PyTorch.”

Sachs, Karen, Omar Perez, Dana Pe’er, Douglas Lauffenburger, and Garry Nolan. 2005. “Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data.” *Science (New York, N.Y.)* 308 : 523–9. [Link]

Zheng, Xun, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. 2018. “DAGs with NO TEARS: Continuous Optimization for Structure Learning.”

Zheng, Xun, Chen Dan, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. 2019. “Learning Sparse Nonparametric DAGs.”

Appendices

B Additional Results

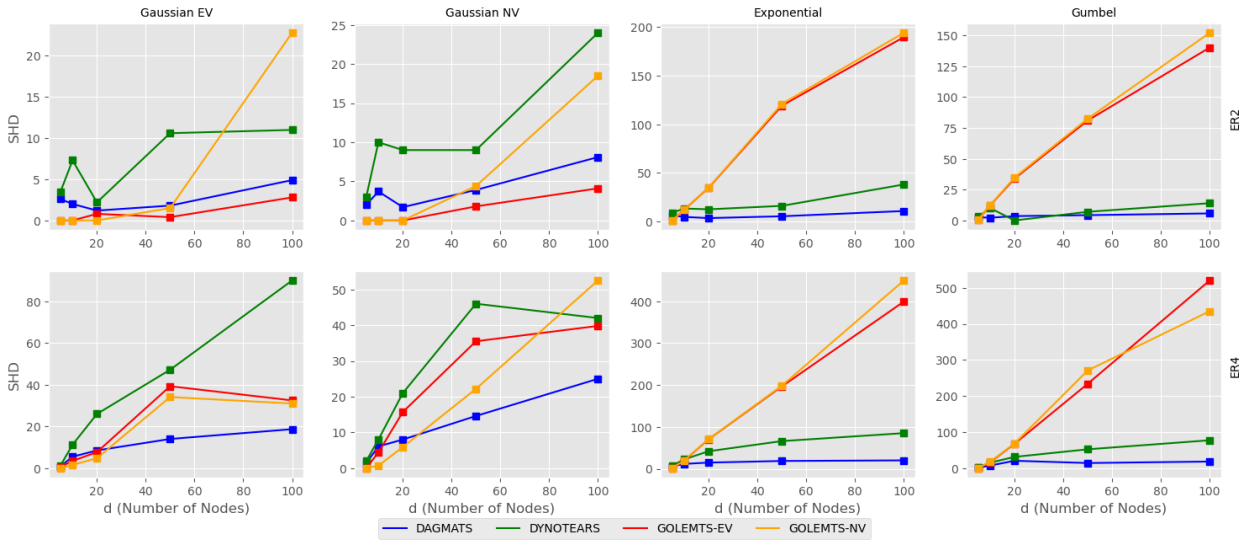

Figure A 1: Structural Hamming Distance (SHD) for linear ground truth graphs when $n = 500$. Lower is better.

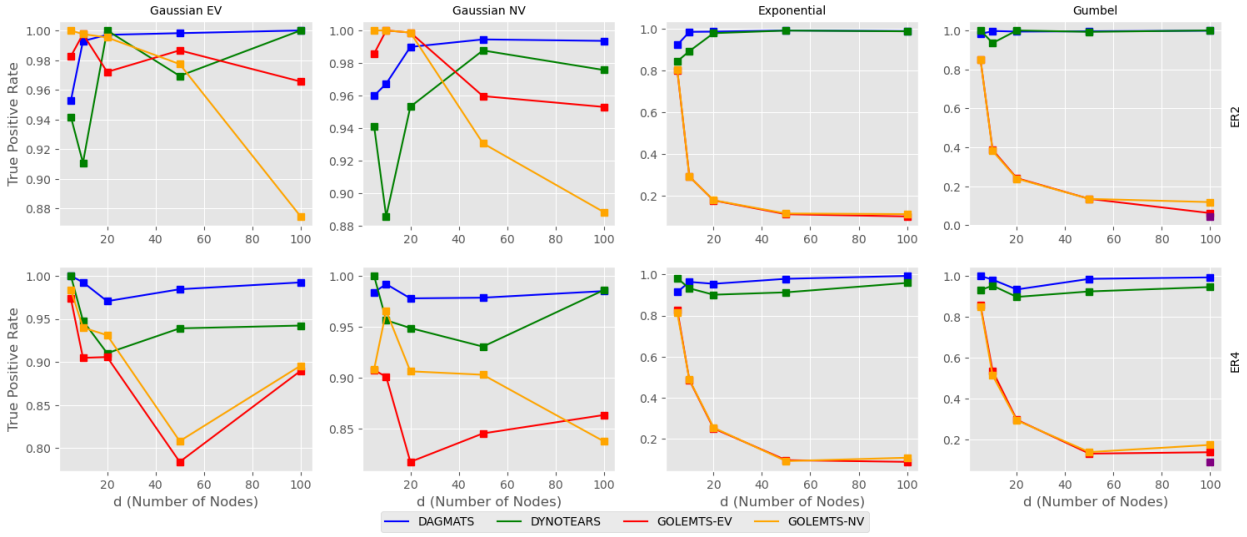

Figure A 2: True Positive Rate (TPR) for linear ground truth graphs when $n = 500$. Higher is better.

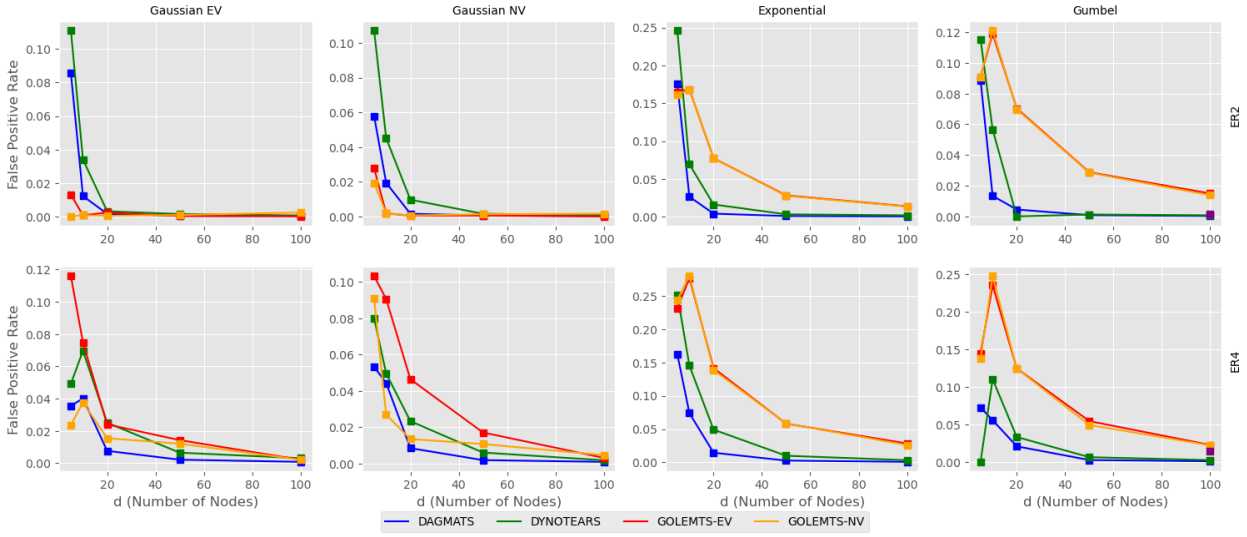

Figure A 3: False Positive Rate (FPR) for linear ground truth graphs when $n = 500$. Lower is better.

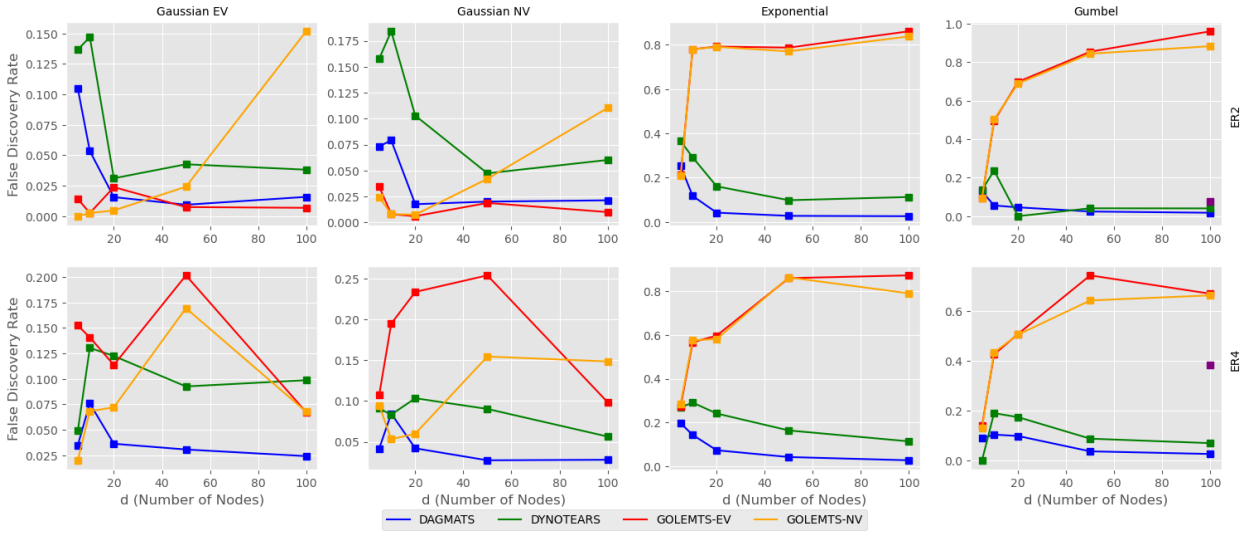

Figure A 4: False Discovery Rate (FDR) for linear ground truth graphs when $n = 500$. Lower is better.

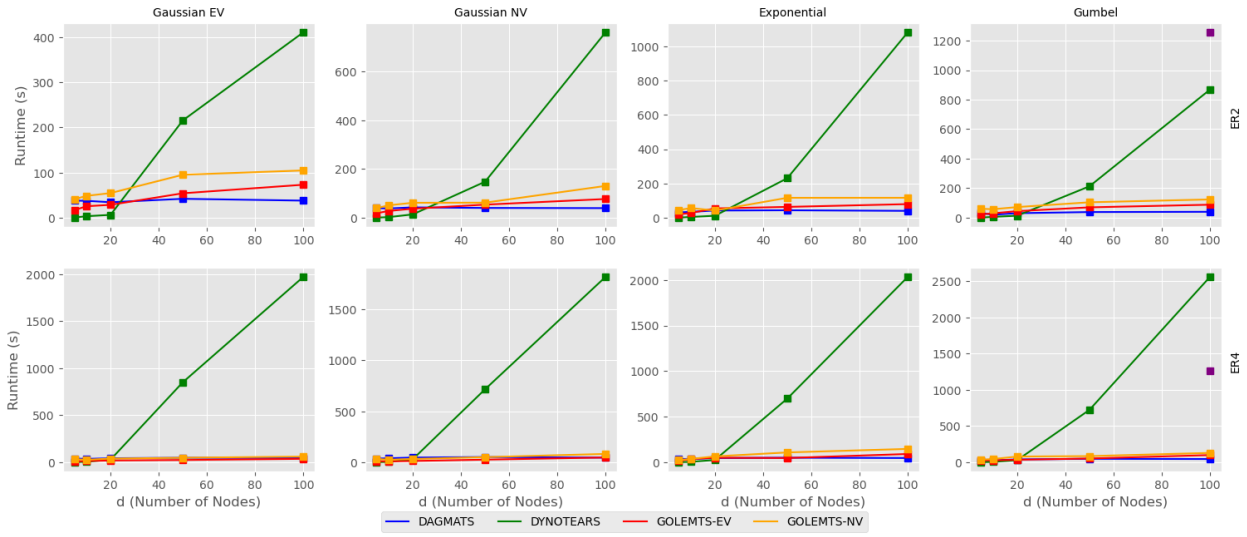

Figure A 5: Runtime (seconds) for linear ground truth graphs when $n = 500$. Lower is better.

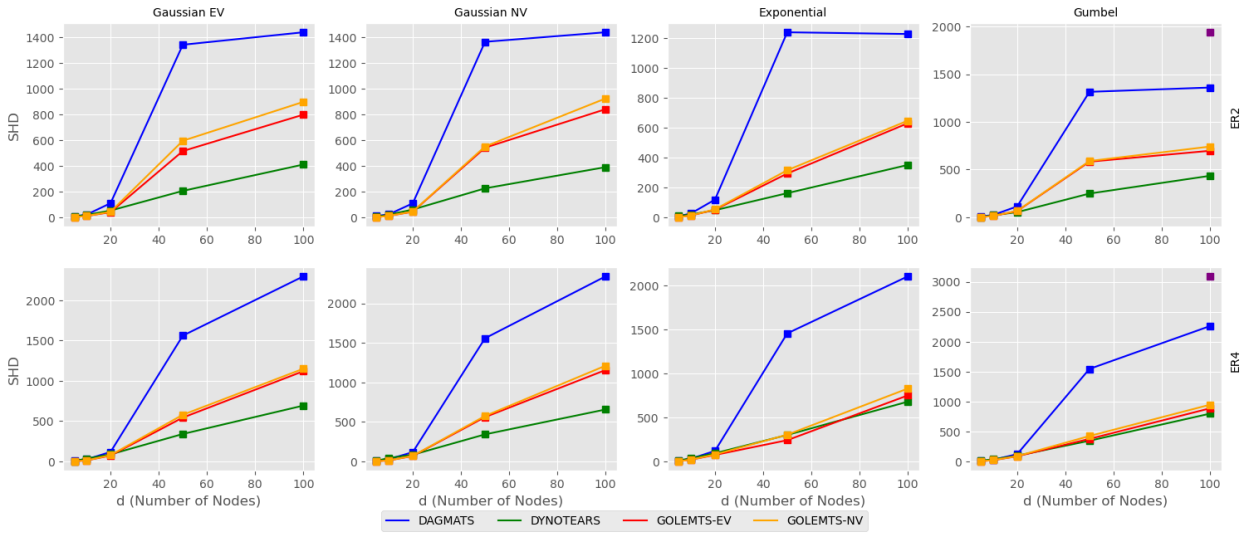

Figure A 6: Structural Hamming Distance (SHD) for linear ground truth graphs when $n = 50$. Lower is better.

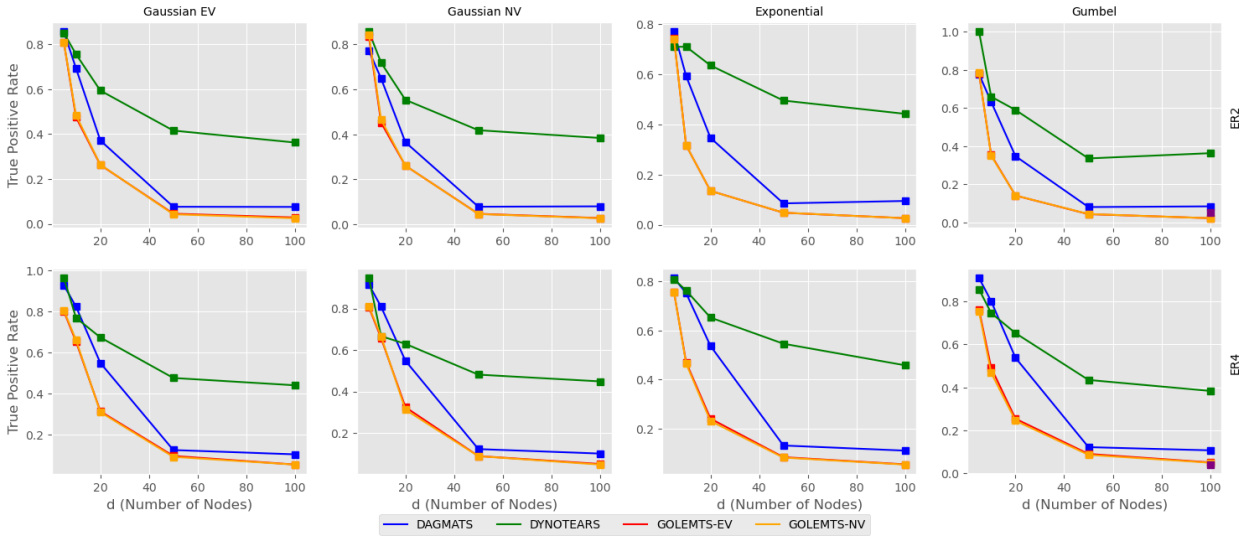

Figure A 7: True Positive Rate (TPR) for linear ground truth graphs when $n = 50$. Higher is better.

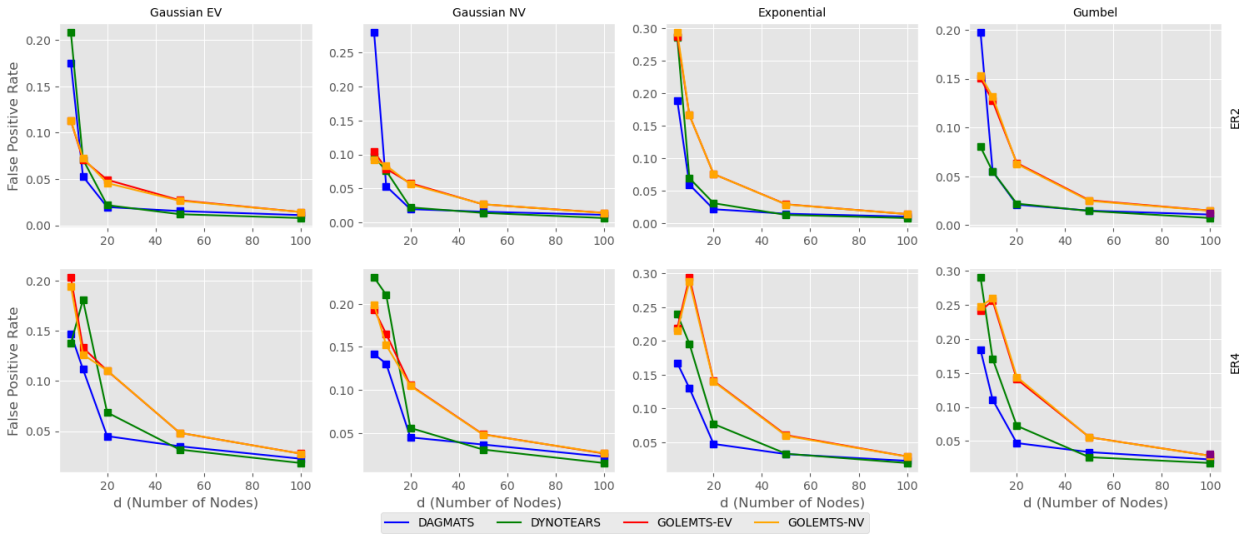

Figure A 8: False Positive Rate (FPR) for linear ground truth graphs when $n = 50$. Lower is better.

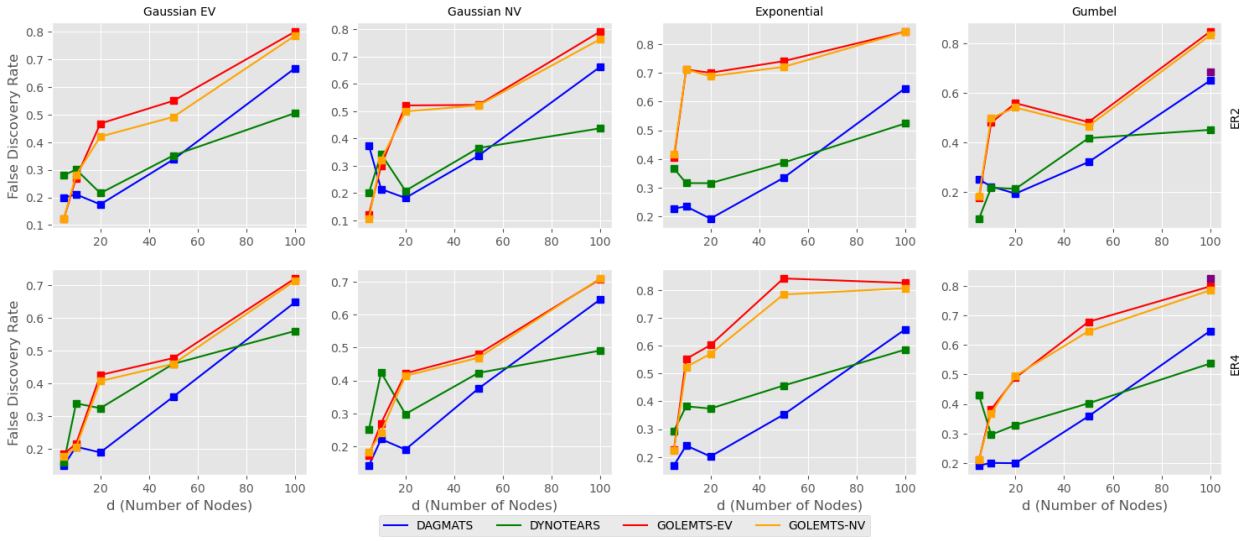

Figure A 9: False Discovery Rate (FDR) for linear ground truth graphs when $n = 50$. Lower is better.

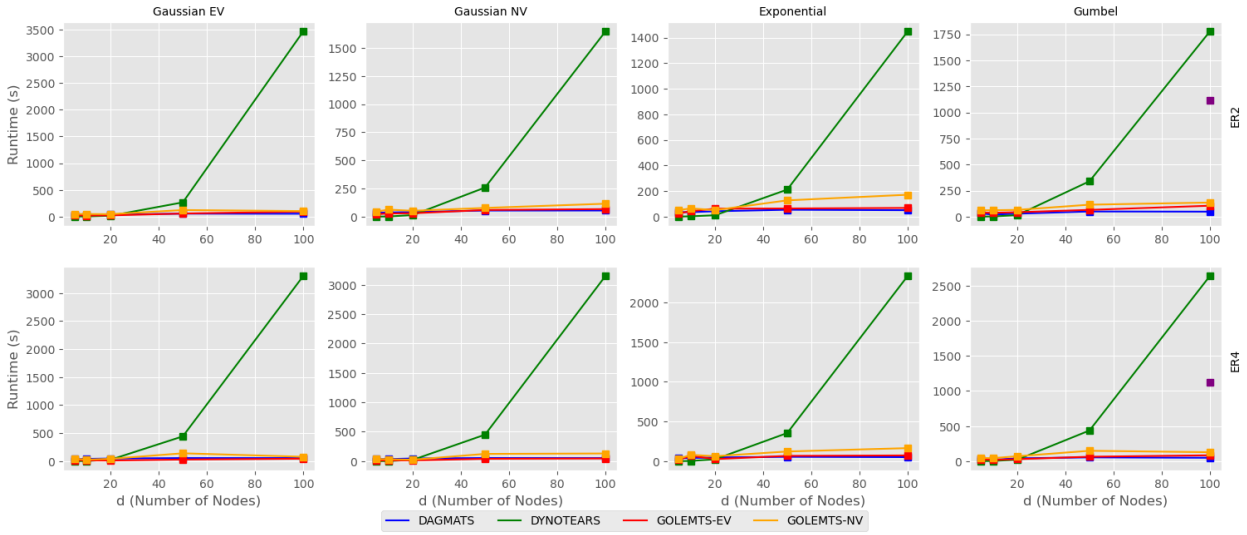

Figure A 10: Runtime (seconds) for linear ground truth graphs when $n = 50$. Lower is better.

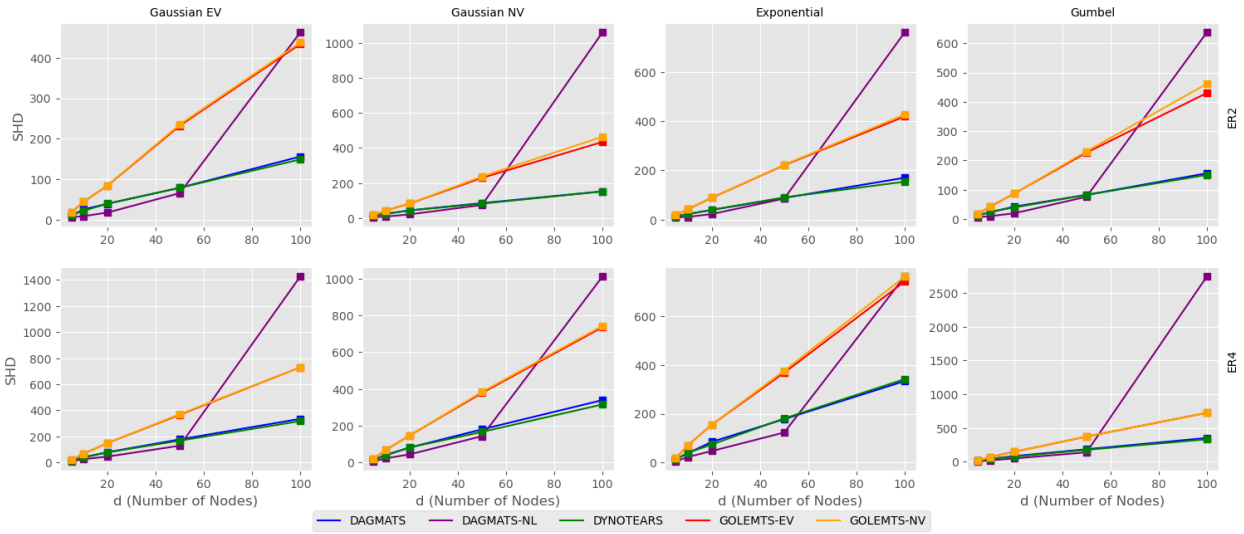

Figure A 11: Structural Hamming Distance (SHD) for nonlinear ground truth graphs when $n = 500$. Lower is better.

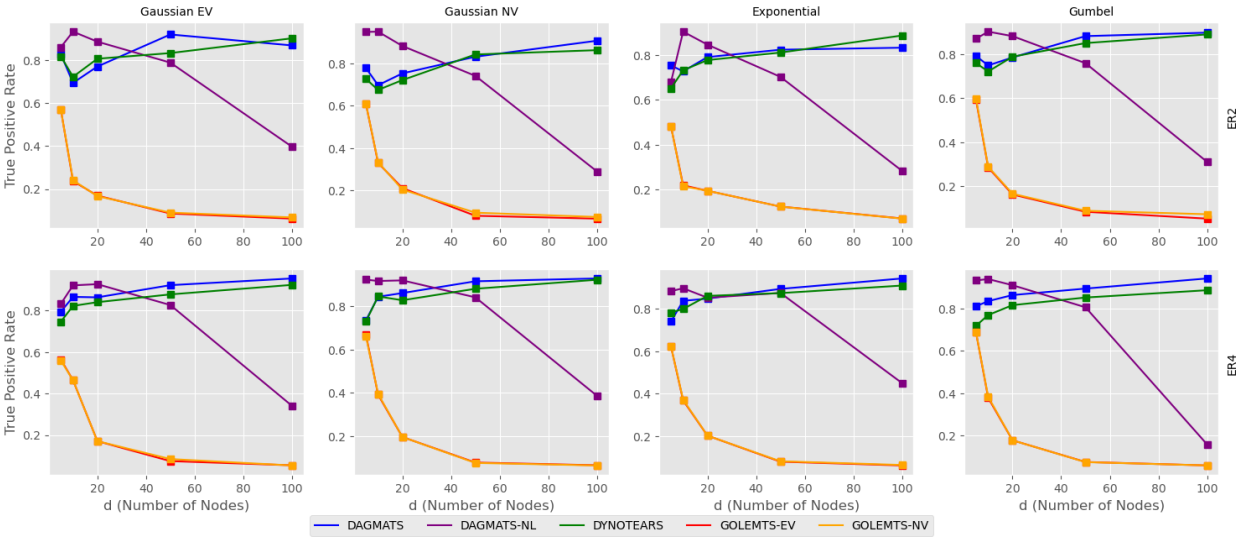

Figure A 12: True Positive Rate (TPR) for nonlinear ground truth graphs when $n = 500$. Higher is better.

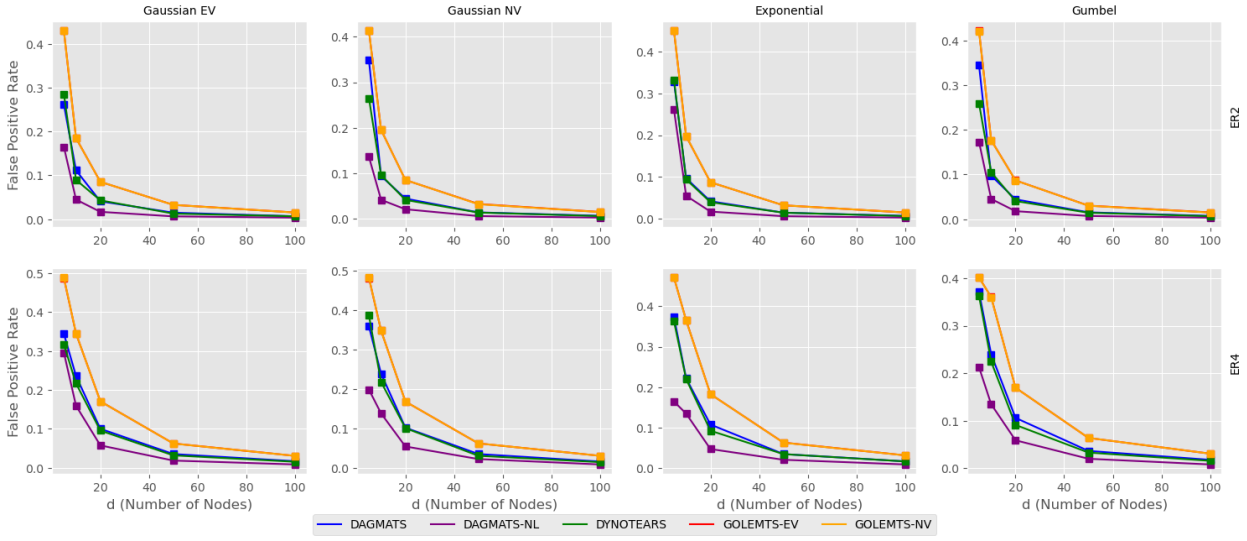

Figure A 13: False Positive Rate (FPR) for nonlinear ground truth graphs when $n = 500$. Lower is better.

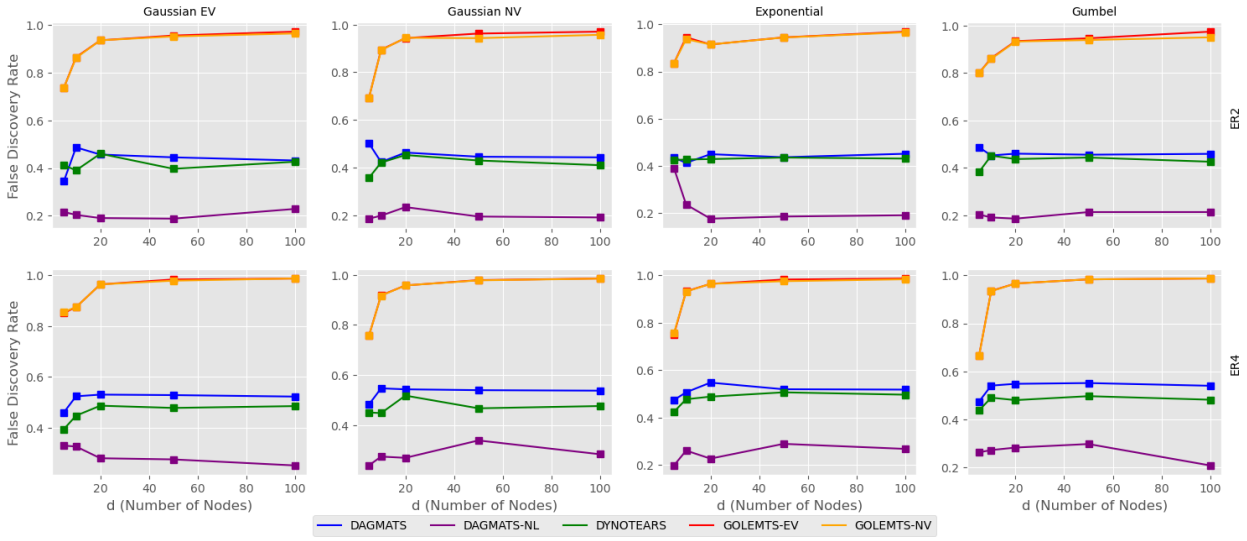

Figure A 14: False Discovery Rate (FDR) for nonlinear ground truth graphs when $n = 500$. Lower is better.

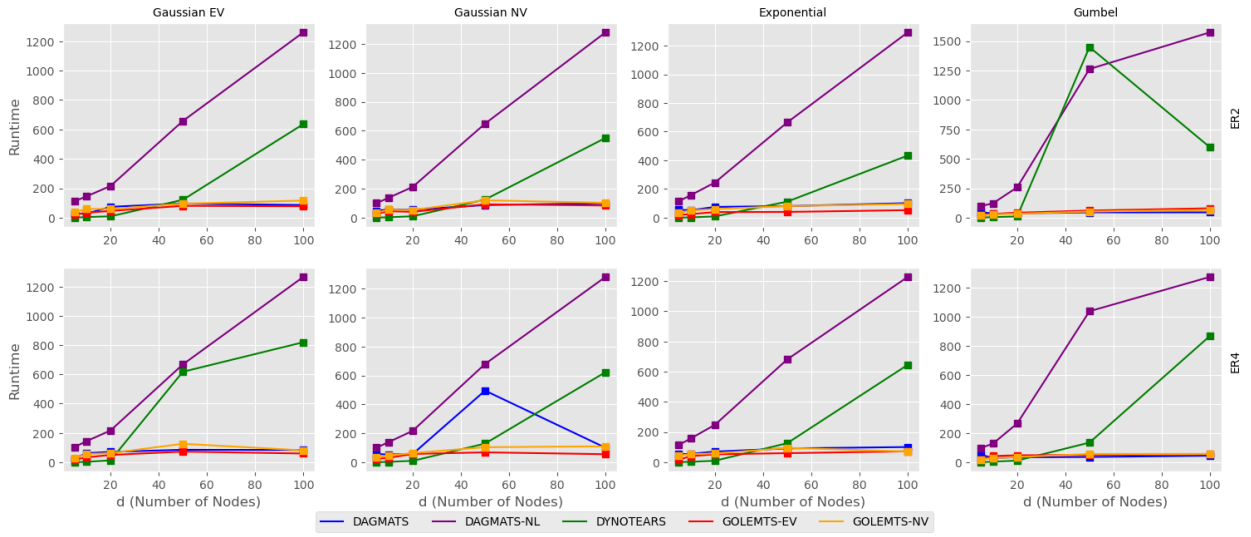

Figure A 15: Runtime (seconds) for nonlinear ground truth graphs when $n = 500$. Lower is better.

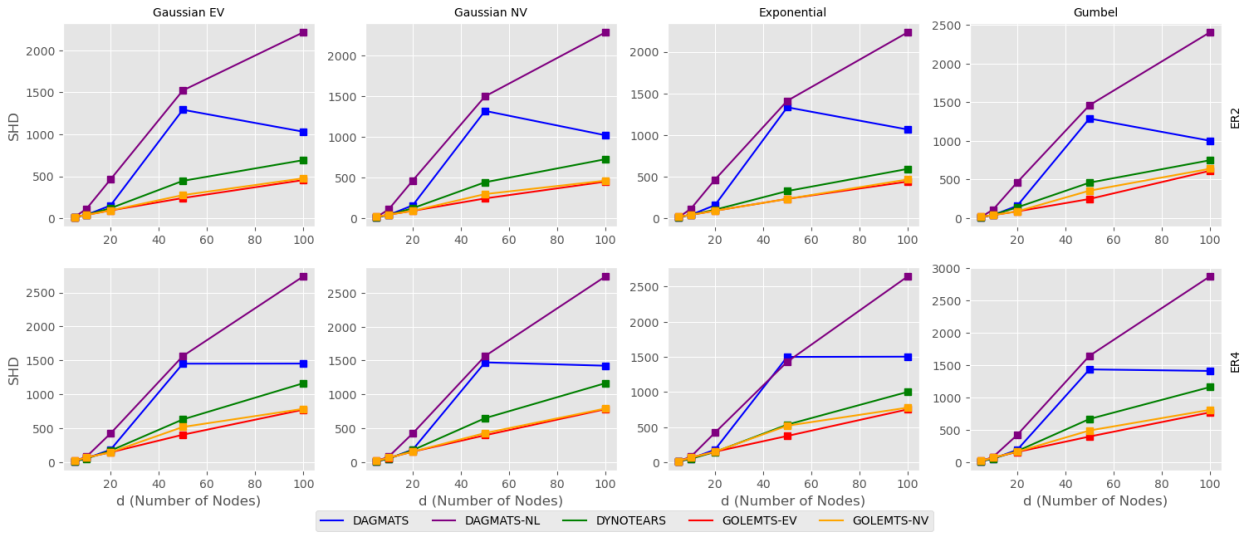

Figure A 16: Structural Hamming Distance (SHD) for nonlinear ground truth graphs when $n = 50$. Lower is better.

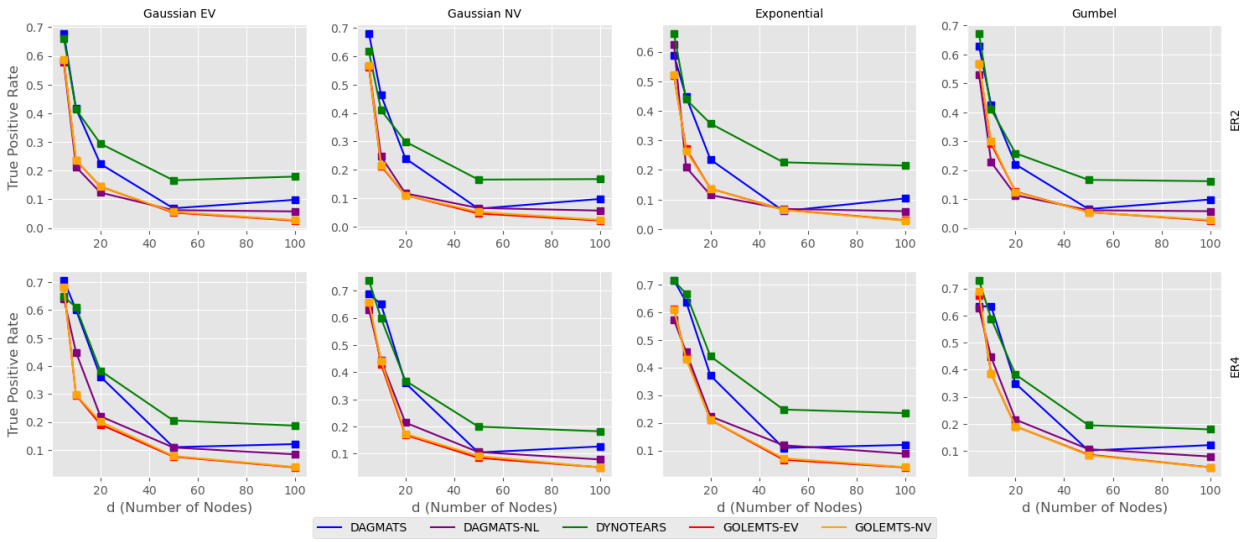

Figure A 17: True Positive Rate (TPR) for nonlinear ground truth graphs when $n = 50$. Higher is better.

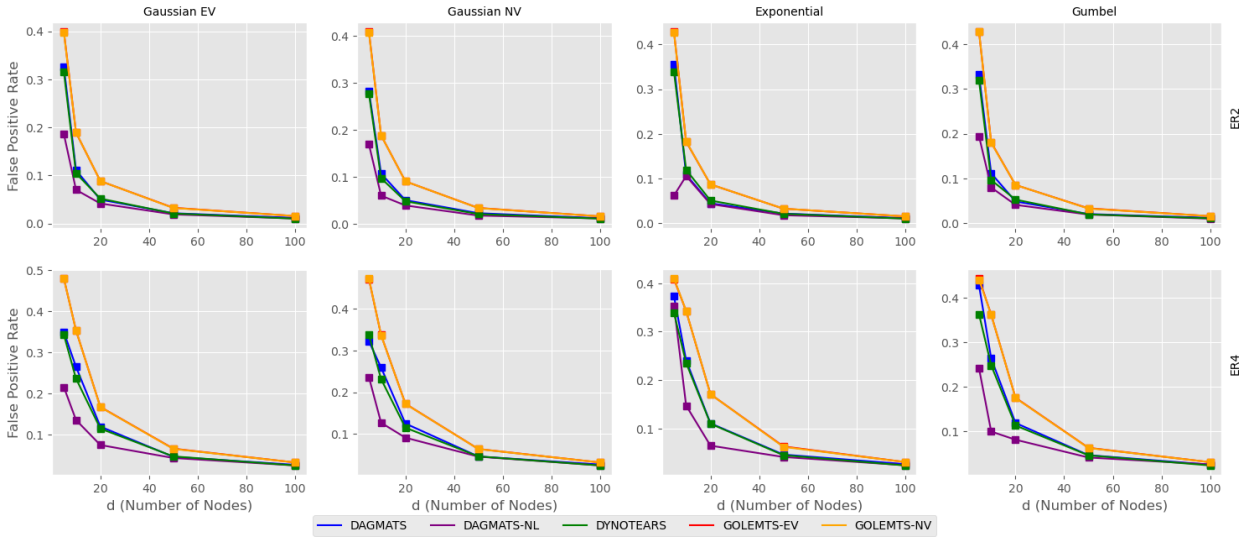

Figure A 18: False Positive Rate (FPR) for nonlinear ground truth graphs when $n = 50$. Lower is better.

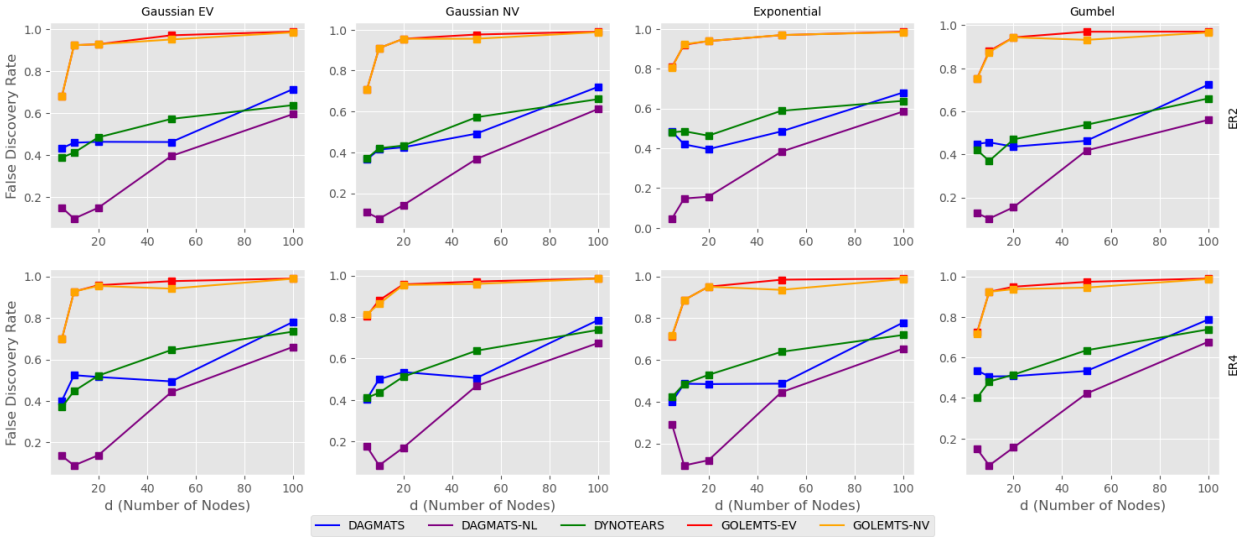

Figure A 19: False Discovery Rate (FDR) for nonlinear ground truth graphs when $n = 50$. Lower is better.

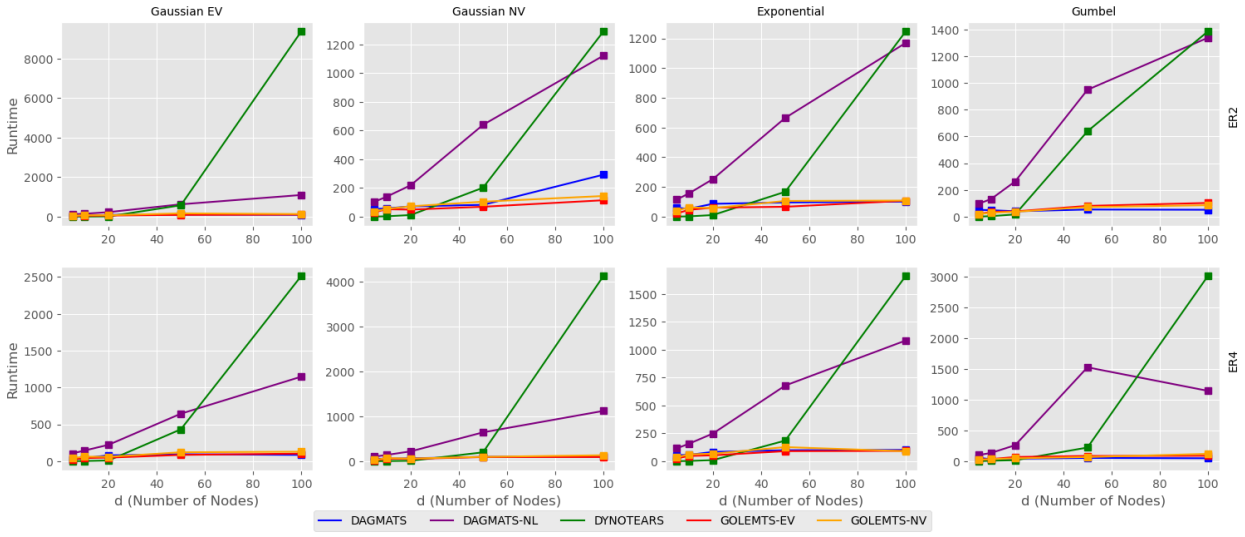

Figure A 20: Runtime (seconds) for nonlinear ground truth graphs when $n = 50$. Lower is better.