Supplementary Materials for DYNOTEARS: Structure Learning from Time-Series Data

Roxana Pamfil[1*], **Nisara Sriwattanaworachai**[1*], **Shaan Desai**[1†], **Philip Pilgerstorfer**[1], **Paul Beaumont**[1], **Konstantinos Georgatzis**[1], **Bryon Aragam**[2]
[1]QuantumBlack, a McKinsey company
causal@quantumblack.com
[2]University of Chicago
bryon@chicagobooth.edu

[*] *These authors contributed equally.*

A Comparison of one-stage and two-stage algorithms

It is possible to minimize the DYNOTEARS objective using either a one-stage algorithm (see Section 2.2) or a two-stage algorithm (see Section 2.3). The two formulations give nearly identical results when the number of samples exceeds the number of variables (i.e., when $n \gg dp$). However, the two-stage algorithm runs somewhat faster, so it should be the preferred option in cases where there is sufficient data.

The difference between the two implementations becomes noticeable especially when the number of samples is below the number of variables. In such cases, estimating the reduced-form VAR from Equation (9) leads to overfitting. In particular, when $n < dp$, we are solving an underdetermined system, so the residual is $\mathbf{e} = 0$ and we cannot get a meaningful estimate of $\mathbf{W}$. One might resort to regularization by imposing an ℓ_1 penalty to enforce the sparsity of $\mathbf{B}_i$ in Equation (9). As $\mathbf{A}_i = \mathbf{B}_i(\mathbf{I} - \mathbf{W}) = \mathbf{B}_i - \mathbf{B}_i\mathbf{W}$, we see that the sparsity of $\mathbf{B}_i$ does not translate directly to the sparsity of $\mathbf{A}_i$. We also observe empirically that $\mathbf{A}_i$ is denser than $\mathbf{B}_i$ in most cases. As a result, one should use a larger regularization parameter in the two-stage setting compared to the one-stage setting. However, to prevent error propagation, it is preferable to estimate $\mathbf{W}$ and $\mathbf{A}$ simultaneously via the combined loss from (5).

B Numerical experiments

B.1 Alternative algorithms

The first algorithm that we use for benchmarking is based on an approach that Murphy proposed in Murphy (2002). His idea was to learn intra-slice and inter-slice structures independently; the former task reduces to a static structure-learning problem, and the latter can be viewed as a feature-selection problem. We use static NOTEARS (with ℓ_1 regularization) to estimate $\mathbf{W}$, and we use Lasso regression (which incorporates ℓ_1 regularization by definition) to estimate $\mathbf{A}$. This provides a more appropriate comparison to DYNOTEARS than the original setup (Friedman et al., 1998; Murphy, 2002). Note that learning $\mathbf{W}$ and $\mathbf{A}$ independently in this way is not equivalent to the two-step formulation of DYNOTEARS from Section 2.3; in the latter case, we apply NOTEARS to the residuals $\mathbf{e}$, rather than to the original data $\mathbf{X}$. Our variant of Murphy's method has the same hyperparameters as DYNOTEARS: two regularization parameters $\lambda_{\mathbf{W}}$ and $\lambda_{\mathbf{A}}$, and two thresholds $\tau_{\mathbf{W}}$ and $\tau_{\mathbf{A}}$ for the weights.

The second algorithm is the SVAR estimation method from Hyvärinen et al. (2010), a time-series version of the LiNGAM algorithm from Shimizu et al. (2006). It follows the two-step approach from Section 2.3 of first estimating a reduced-form VAR model and then applying LiNGAM to the residuals. With the assumption of non-Gaussian errors, the resulting model is identifiable (Hyvärinen et al., 2010; Shimizu et al., 2006). The two hyperparameters of the time-series version of LiNGAM are the weight thresholds $\tau_{\mathbf{W}}$ and $\tau_{\mathbf{A}}$, which we include for comparability with the other algorithms. (The authors of Hyvärinen et al., 2010 did not have this thresholding as part of their method.)

The third algorithm in our experiments is tsGFCI (Malinsky and Spirtes, 2018), a time-series extension of the Greedy Fast Causal Inference (GFCI) algorithm (Ogarrio et al., 2016). Both GFCI and tsGFCI are hybrid algorithms that rely on conditional-independence tests and on local changes to a graph to incrementally improve the BIC score. These algorithms work in settings with latent variables and return a partial ancestral graph (PAG). We define heuristics to extract adjacency matrices $\mathbf{W}$ and $\mathbf{A}$ from the output PAG (see Appendix B.2 for details). When there is ambiguity in whether an edge is present or not, we treat

[†] Contributed during an internship at QuantumBlack.

tsGFCI as favorably as possible. One important parameter in tsGFCI is the "penalty discount"; larger values of this parameter increase the BIC penalty and thus result in sparser output graphs. In our experiments on simulated data, we find that setting the penalty discount between 2 and 4 produces output graphs that are closest to the ground truth. Our simulations from Section 3.2 and Appendix B.7 use a value of 2 (which is also the default value).

B.2 Interpreting a PAG as a DAG

The tsGFCI algorithm returns a partial ancestral graph (PAG), which one cannot immediately compare to a ground-truth DAG. Thus, we developed a set of rules to convert the PAG output to a DAG, making sure to do so in a manner that favors tsGFCI. The rules are as follows:

- If an edge is directed (i.e., A $\rightarrow$ B in the PAG), then we treat it as a directed edge in the DAG.
- If an edge in the PAG is either directed or it indicates the presence of a latent factor (i.e., A $\circ\!\!\rightarrow$ B), then we check whether the directed edge exists in the ground truth graph and assume that tsGFCI made the correct choice.
- If two nodes are related through a latent variable (i.e., A $\longleftrightarrow$ B in the PAG), then we disregard the edge.
- If the edge is ambiguous (i.e., A $\circ\!\!-\!\!\circ$ B), then we assume that tsGFCI made the correct choice; in other words, we check whether A $\rightarrow$ B, B $\leftarrow$ A, or A is not connected to B in the ground-truth DAG and we assume that tsGFCI made the same choice.

Using these rules, we pick the outcomes most favorable for tsGFCI in ambiguous cases. This implies, in particular, that our results slightly overstate the performance of tsGFCI on simulated data.

B.3 Hyperparameter selection

For our simulations with $n = 500$, we apply a small amount of regularization, $\lambda_{\mathbf{W}} = \lambda_{\mathbf{A}} = 0.05$, for both DYNOTEARS and NOTEARS + Lasso. Because the number of samples exceeds the number of variables, performance is not particularly sensitive to the amount of regularization. For all algorithms except tsGFCI, we apply the weight thresholds $\tau_{\mathbf{W}} = 0.3$ and $\tau_{\mathbf{A}} = 0.1$. We set $\tau_{\mathbf{W}} = 0.3$ to be consistent with the experiments for static NOTEARS from Zheng et al. (2018), and we set $\tau_{\mathbf{A}} = 0.1$ using an analogous heuristic. (We experimented with other threshold values, and the relative ranking of the algorithms was largely the same.) For tsGFCI, we set the penalty discount to 2; recall discussion from Appendix B.1.

Regularization becomes more important for $n = 50$, so we set $\lambda_{\mathbf{W}} = \lambda_{\mathbf{A}} = 0.2$ for DYNOTEARS and NOTEARS + Lasso. We set $\tau_{\mathbf{W}} = 0.3$ and $\tau_{\mathbf{A}} = 0.2$. We keep the penalty discount at 2 for tsGFCI, as experimentation with other values did not yield superior results.

Although we did not attempt to optimize hyperparameters for our experiments on simulated data, our work on the S&P100 and the DREAM4 datasets (see Section 4) indicates ways in which one can estimate these parameters through cross-validation.

B.4 Data generation process

We provide more details about the data generation process that we use in our numerical experiments from Section 3.

Intra-slice model As in Zheng et al. (2018), we use either the Erdős–Rényi (ER, Newman, 2018) model or the Barabási–Albert (BA, Barabási and Albert, 1999) model to generate intra-slice graphs given a target mean degree k (which counts both incoming and outgoing edges). In the ER model, one samples edges using i.i.d. Bernoulli trials. To ensure that the resulting graph is a DAG, we sample lower-triangular entries of $\mathbf{W}^{\text{bin}}$ in this way, and we then permute the rows and columns to randomize node order. By setting the probability of each Bernoulli trial to $k/(d-1)$, the expected mean degree in the resulting graph is k, as desired. The BA model (Barabási and Albert, 1999) relies on a "preferential attachment" mechanism to generate growing networks in which nodes are added one by one. For each new node, one generates $k/2$ outgoing edges. The targets of these edges are selected at random from the existing nodes, proportionally to their current degrees. This mechanism encapsulates a "rich get richer" effect that produces, at the end of the process, graphs with a power-law degree distribution. The BA model thus produces graphs that mirror the wide degree distributions that are common in many real-world networks. By construction, this formulation of the BA model generates DAGs. As for ER models, we permute the rows and columns of the resulting adjacency matrix so that nodes are not a priori sorted in the topological order.

To go from an unweighted to a weighted DAG, we follow Zheng et al. (2018) and we sample weights uniformly at random from $[-2.0, -0.5] \cup [0.5, 2.0]$.

Inter-slice model We use two models to generate inter-slice graphs. One is a directed ER model in which we sample entries of the binary adjacency matrix $\mathbf{A}^{\text{bin}}$ using i.i.d. Bernoulli trials with probabilities k/d. This choice implies that the mean in-degree of nodes at time

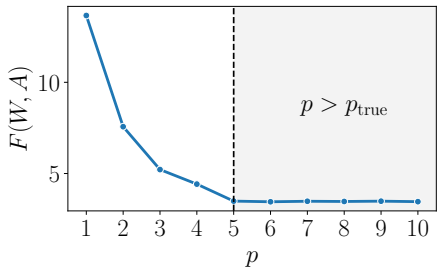
t is equal to pk. The other model is a simplified type of the so-called stochastic block model (SBM, Newman, 2018). In our simulations, we partition the d variables into two blocks, and we assume that the probability of an inter-slice edge from i to j is equal to p_{in} if i and j are in the same block, and it is equal to p_{out} otherwise. We select p_{in} and p_{out} such that $p_{\text{out}}/p_{\text{in}} = 0.3$ (so edges are more likely between two variables in the same cluster) and such that the expected mean in-degree of nodes at time t is pk. In applications to real data we expect some clustering of the variables and of their causal effects, and we rely on SBMs to replicate this feature in our simulation experiments.

Given a binary inter-slice adjacency matrix $\mathbf{A}^{\text{bin}}$, we sample edge weights uniformly at random from a specified interval, which we allow to depend on p. More precisely, we sample edge weights from slice $t-p$ to slice t from $[-0.5\alpha, -0.3\alpha] \cup [0.3\alpha, 0.5\alpha]$, where $\alpha = 1/\eta^{p-1}$ with $\eta \geq 1$. The weight decay parameter η therefore reduces the influence of variables that are farther back in time from the current time slice. (Of course, there is no such penalty when $\eta = 1$.) We incorporate this parameter into our data-generation process because it replicates a time-decay scenario that we expect to encounter in real-world applications. It also allows us to determine whether DYNOTEARS can learn edge weights across multiple scales, ideally without having to specify different thresholds for each matrix $\mathbf{A}_i$ ($i \in \{1, \dots, p\}$).

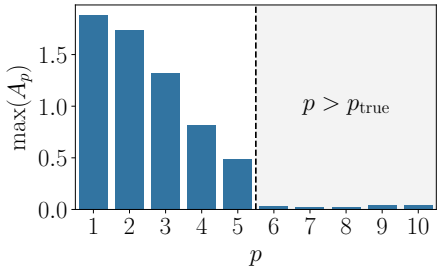
Once we have $\mathbf{W}$ and $\mathbf{A}$, we use the SEM from (3) to generate a data matrix $\mathbf{X}$ of size $n \times d$. The noise term $\mathbf{Z}$ in (3) is a matrix of i.i.d. random variables. Following Zheng et al. (2018), we use normal and exponential distributions with tuneable scale parameters (set to 1 by default) for these random variables.

B.5 Autoregressive order

In Section 3.2, we assumed that the correct value of the autoregressive order p is given. This is rarely the case in applications, so it is useful to be able to estimate p from data. In Figure B.1, we indicate two such potential diagnostics for a simulated dataset with $p_{\text{true}} = 5$. In Figure B.1a, we see that the objective function decreases as we run DYNOTEARS for increasing values of p. However, the objective values plateau once $p > p_{\text{true}}$, as the increasingly complex models do not yield better fits to the data. For real-world data, where there is no single "true" value for p, one can also look at plateaus in the BIC score. An alternative method for selecting p is to look at the magnitude of the weights in the estimated inter-slice matrices, as we do in Figure B.1b. For $p > p_{\text{true}}$, $\mathbf{A}_p$ does not contain entries that are significantly above 0 in magnitude. Thus, in cases when p is unknown, one can keep increasing p until the entries of $\mathbf{A}_p$ become negligible.



(a) Objective value as a function of p



(b) Largest absolute value in $\mathbf{A}_p$ as a function of p

Figure B.1: Results for fitting a model using different values of p to data with $p_{\text{true}} = 5$. (a) The objective value $F(\mathbf{W}, \mathbf{A})$ plateaus for $p > p_{\text{true}}$. (b) The estimated edge weights in $\mathbf{A}_p$ are close to 0 for $p > p_{\text{true}}$.

B.6 Running times

We provide some illustrative running times for different values of d in Figure B.2. These running times also depend on the density of the underlying graph and on the distribution of edge weights, although it is difficult to quantify the precise relationship. The speed of DYNOTEARS and NOTEARS + LiNGAM is heavily dependent on the values of the regularisation parameters, with larger values resulting in faster running times.[1]

Although tsGFCI and LiNGAM run significantly faster than DYNOTEARS, we believe that the gain in accuracy from using the latter makes it worthwhile even in cases with hundreds of variables. As a reminder, running DYNOTEARS on the S&P100 dataset (which has $d \approx 100$) takes a few minutes on a typical laptop.

[1] For the DREAM4 datasets, the CPU times are approximately 0.1 minutes, 1.5 minutes, and 60 minutes, respectively, for $\lambda_{\mathbf{W}} \in \{0.1, 0.01, 0.001\}$ and $\lambda_{\mathbf{A}} = 0.01$.

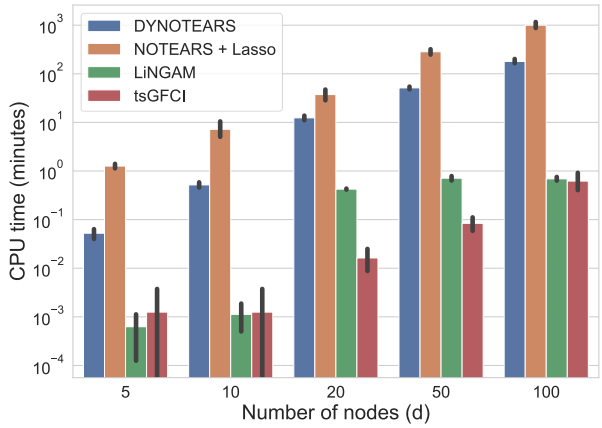


Figure B.2: CPU times for the simulations from Figure B.3 (Gaussian noise and $n = 500$). Running times are comparable for Exponential noise and for $n = 50$.

B.7 Additional results

Additional ground-truth graphs The relative performance of different algorithms varies as we change the density of the ground-truth graphs. In Figure B.3, we show the F1 scores for simulated data with Gaussian noise, $n = 500$, four choices of intra-slice graphs (columns), and four choices of inter-slice graphs (rows). The performance of tsGFCI is especially sensitive to changes in graph densities, with a notable drop in F1 scores when the intra-slice graph is ER4.

Additional performance metrics In Figures B.4 to B.7, we plot four additional performance metrics to complement the F1 scores from Figure 3 in the main part of the paper. The metrics are standard and are defined as follows:

- True positive rate (TPR): number of correctly-identified edges divided by the number of edges in the ground-truth graph.
- False discovery rate (FDR): number of incorrectly-identified edges divided by the number of edges in the estimated graph.
- Structural Hamming distance (SHD): number of changes (i.e., edge removals, edge additions, and edge reversals) required to go from one (unweighted) graph to another.
- Frobenius norm (FRO) of the difference between two weighted matrices (i.e., between a ground-truth adjacency matrix and an estimated matrix).

Note that the Frobenius norm does not apply to the tsGFCI algorithm, which only returns unweighted edges.

For $n = 500$, DYNOTEARS generally outperforms the other algorithms for both Gaussian (Figure B.4) and exponential noise (Figure B.5); there are some exceptions to this when the number of variables is small, $d \in \{5, 10, 20\}$. For $n = 50$ (see Figures B.6 and B.7), NOTEARS + Lasso and LiNGAM output estimated graphs that are significantly denser than the ground truth. As a result, while these two algorithms have large TPRs, their overall performance is not competitive due to a large number of false positives.

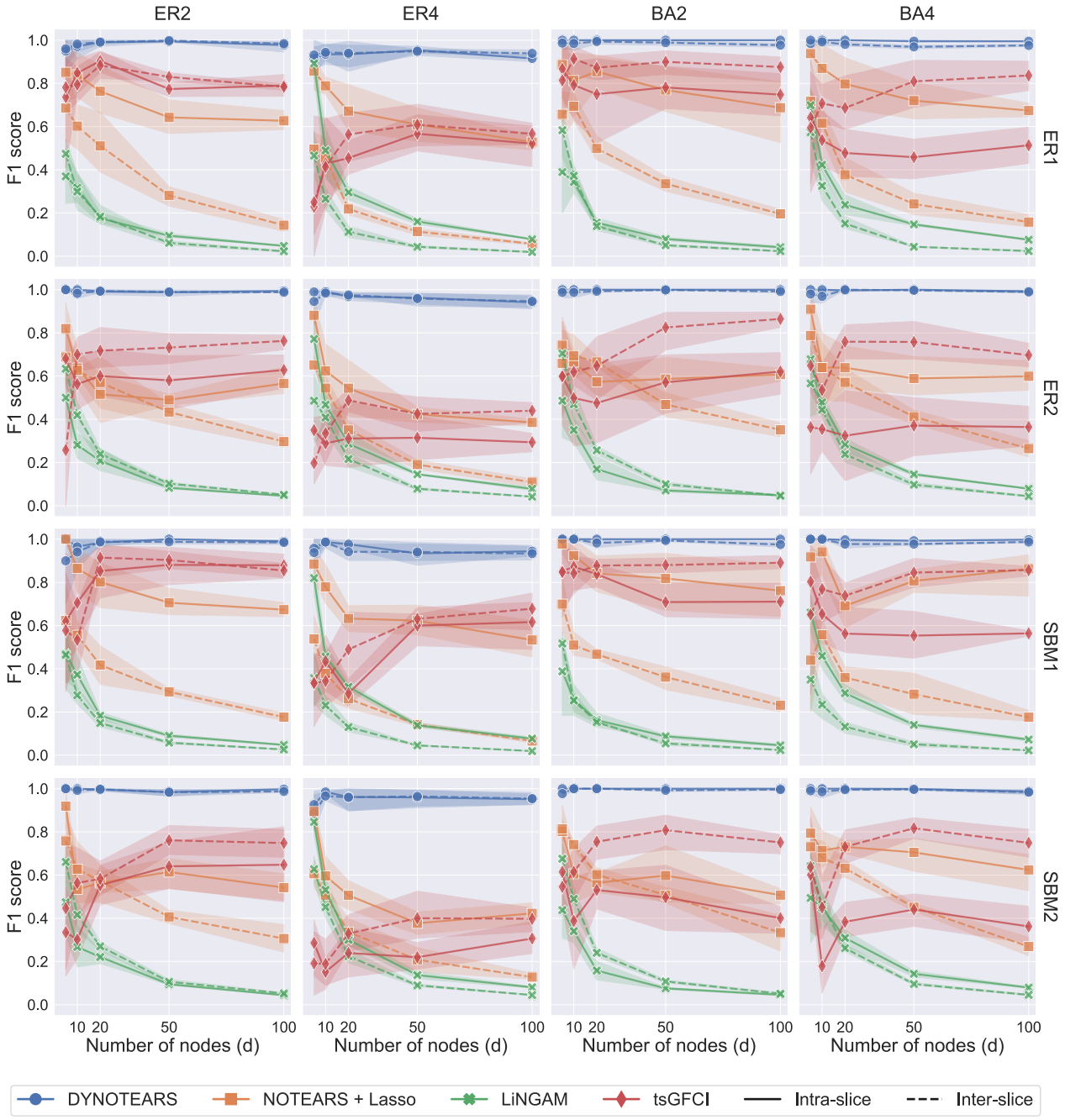
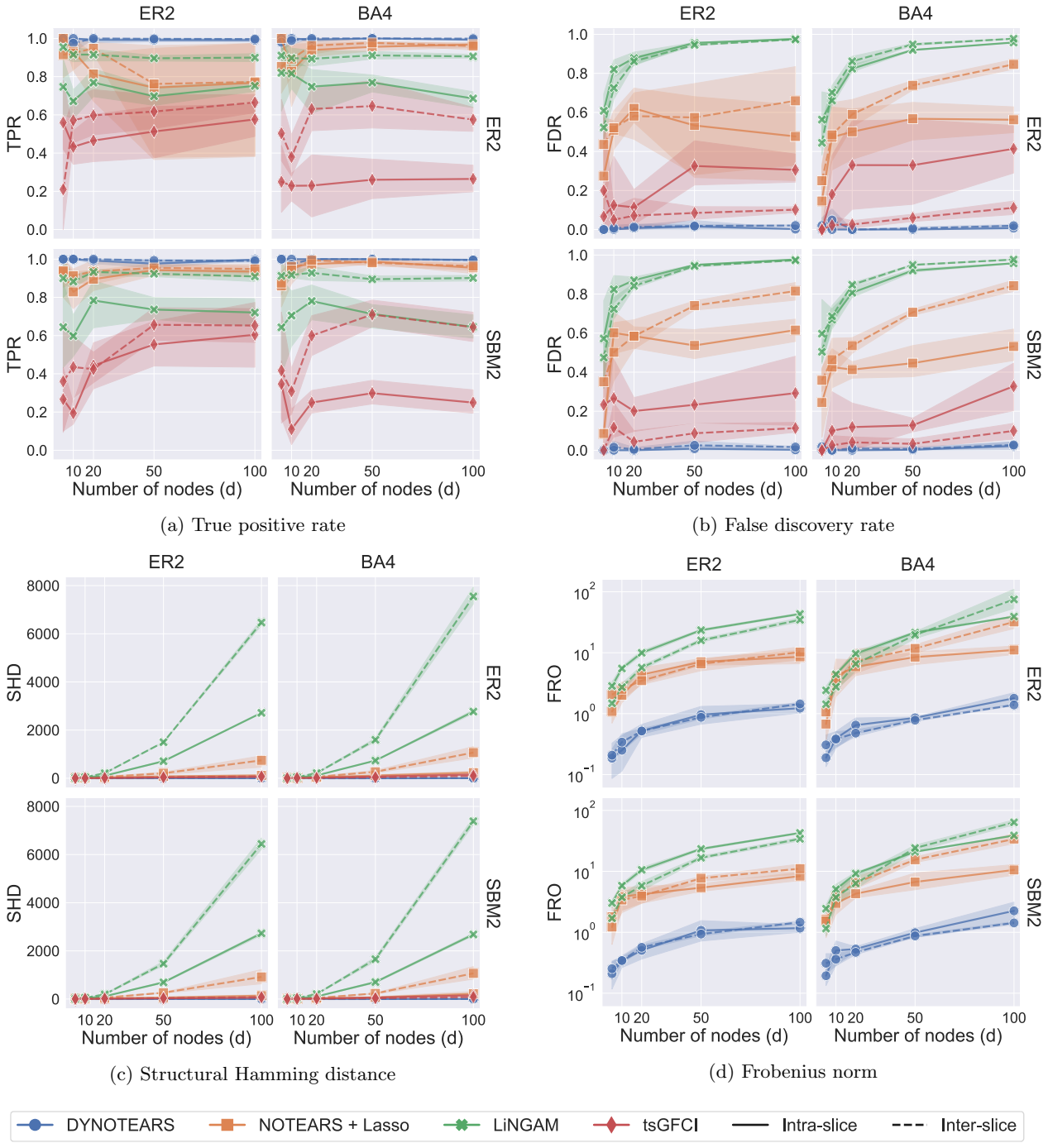


Figure B.3: F1 score for $n = 500$, $p = 1$, $d \in \{5, 10, 20, 50, 100\}$, Gaussian noise, and different choices of intra-slice graphs (columns) and inter-slice graphs (rows). Each marker indicates the mean performance across 5 algorithms runs (each on a different simulated dataset).



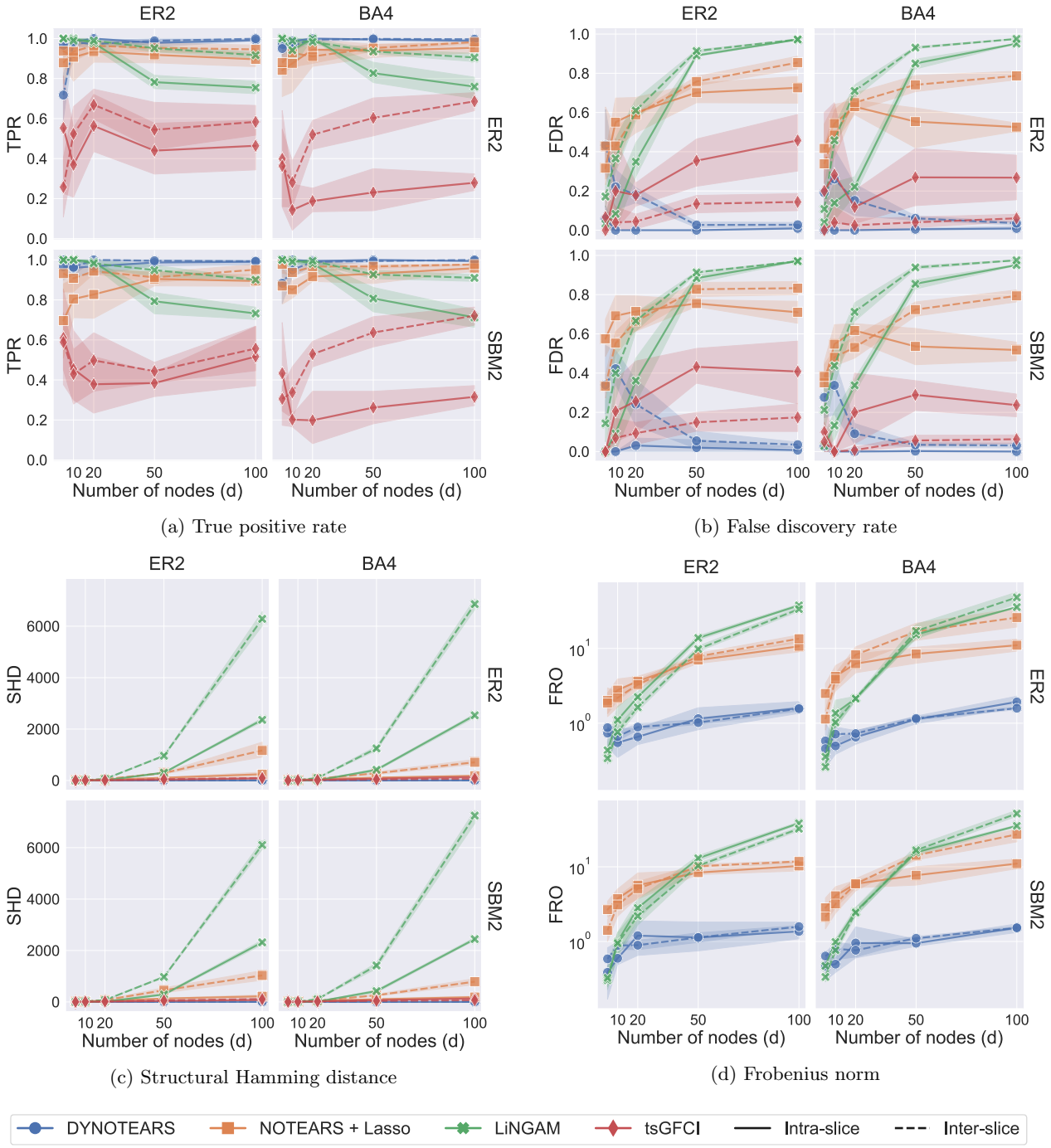
(a) True positive rate

(b) False discovery rate

(c) Structural Hamming distance

(d) Frobenius norm

Figure B.4: Results for $n = 500$, Gaussian noise. Each panel corresponds to a different performance metric.



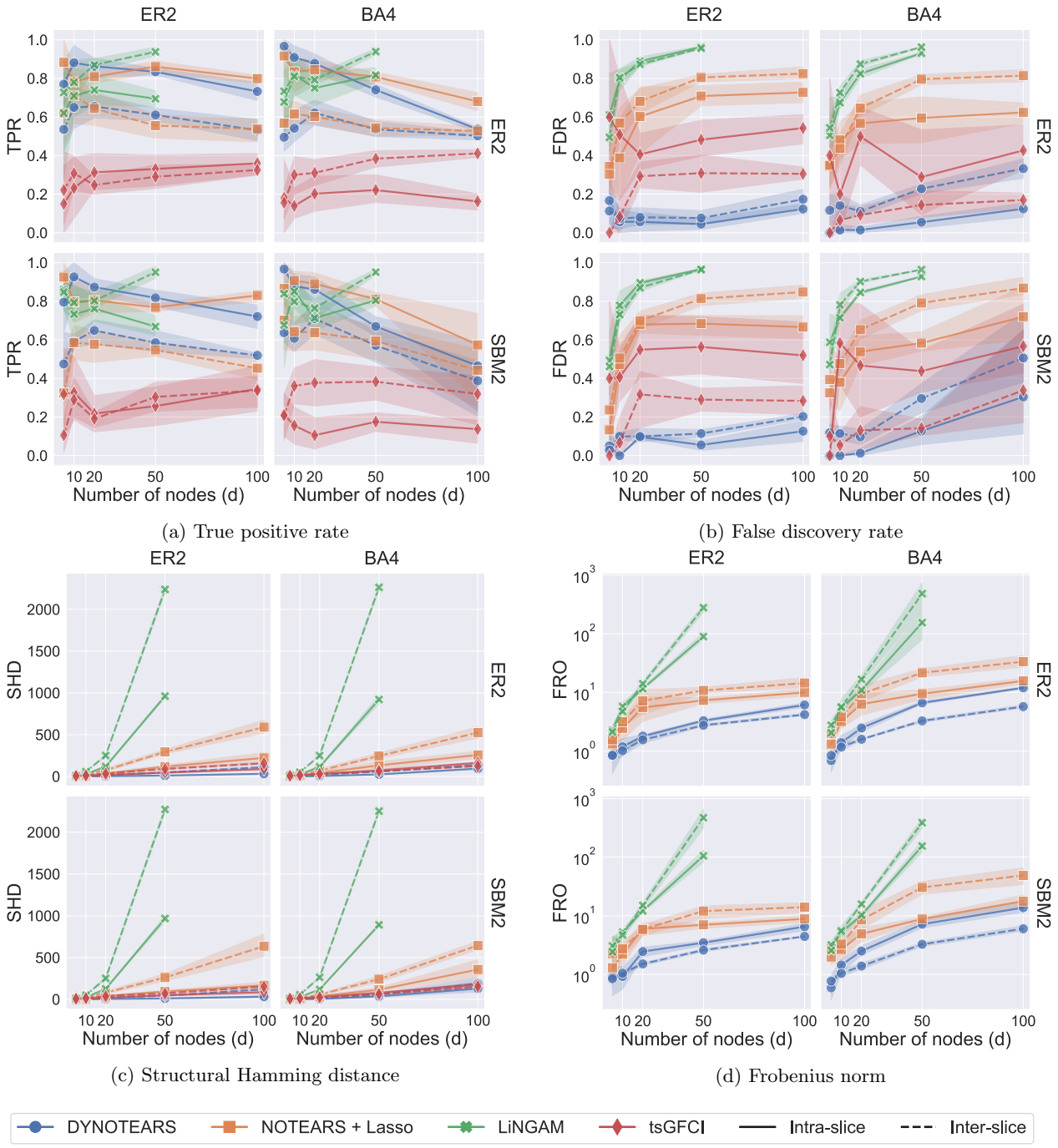
(a) True positive rate

(b) False discovery rate

(c) Structural Hamming distance

(d) Frobenius norm

Figure B.5: Results for $n = 500$, exponential noise. Each panel corresponds to a different performance metric.



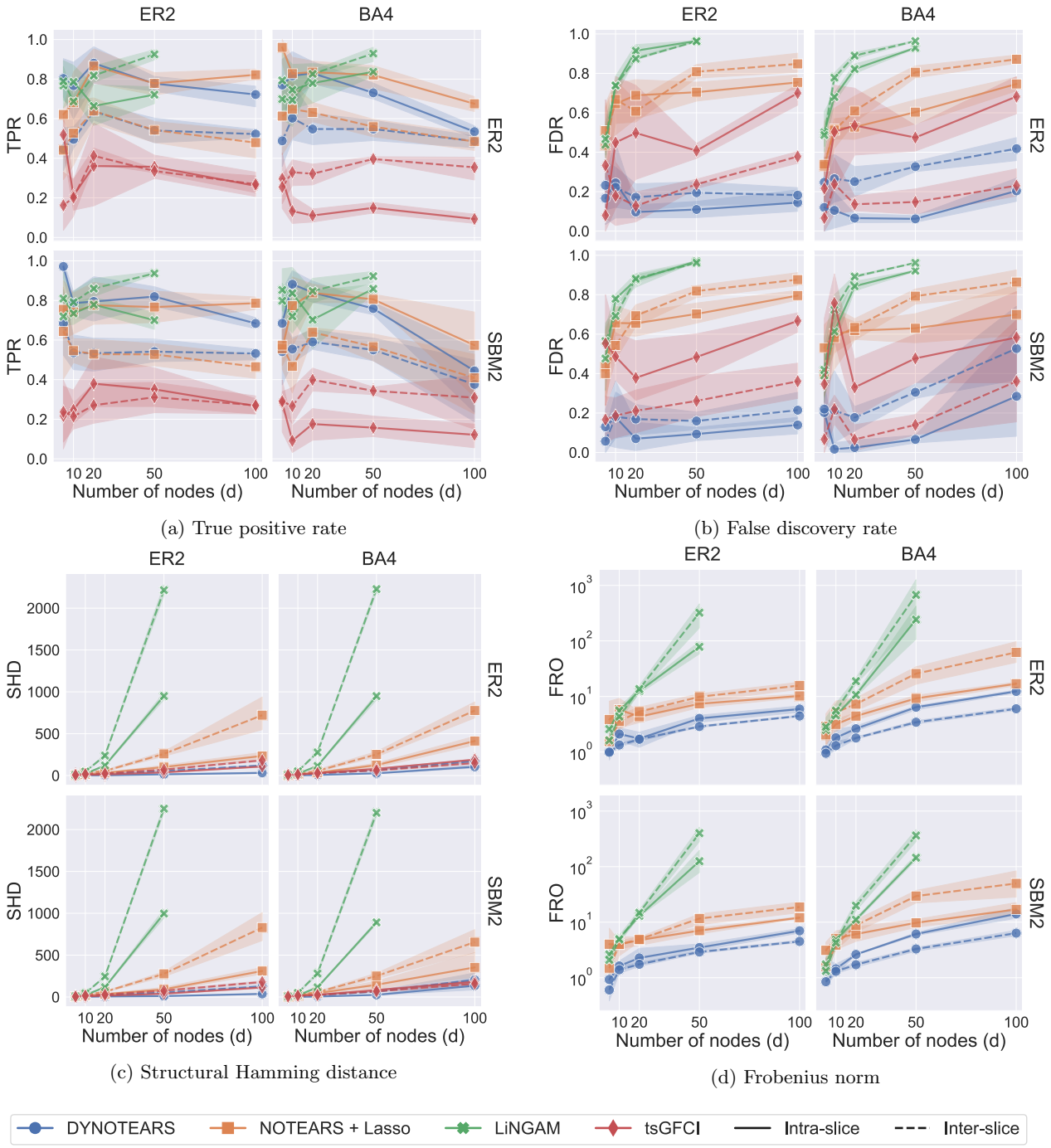
(a) True positive rate

(b) False discovery rate

(c) Structural Hamming distance

(d) Frobenius norm

Figure B.6: Results for $n = 50$, Gaussian noise. Each panel corresponds to a different performance metric. LiNGAM does not work for $n < d$, so the corresponding results are missing from the plots.



(a) True positive rate

(b) False discovery rate

(c) Structural Hamming distance

(d) Frobenius norm

Figure B.7: Results for $n = 50$, exponential noise. Each panel corresponds to a different performance metric. LiNGAM does not work for $n < d$, so the corresponding results are missing from the plots.

C S&P100 application

C.1 Parameter selection

We hold out the last 400 trading days as a validation set and we discard 2 data points between the validation and trainining sets. This roughly corresponds to a 33%/66% split. We report the Frobenius norm across a range of parameter values in Figure C.1. Note that the multiples of 3 are used as half-way points in the log-10 space and added to create a finer grid. However, we find that the surface is fairly smooth and we show these values for completeness. We select $\lambda_{\mathbf{W}} = 0.1$ and $\lambda_{\mathbf{A}} = 0.1$, as they correspond to the smallest values in the grid.

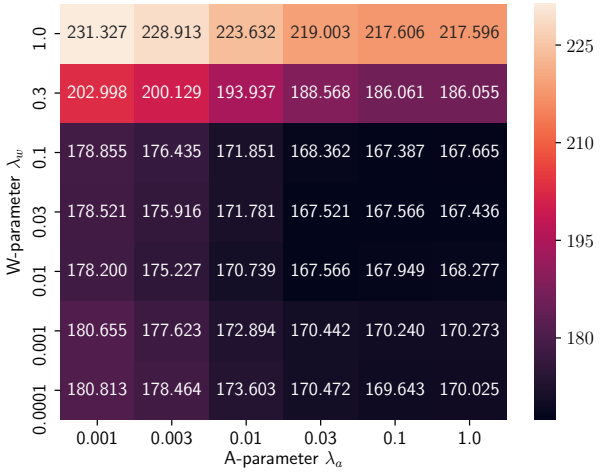


Figure C.1: Heatmap of validation loss using the Frobenius norm for the S&P100 dataset across a range of $\lambda_{\mathbf{A}}$ and $\lambda_{\mathbf{W}}$ parameters. Note that $\lambda_{\mathbf{W}} = 1$ and $\lambda_{\mathbf{A}} = 1$ correspond to zero-matrices for $\mathbf{W}$ and $\mathbf{A}$ and the loss is equal to the Frobenius norm of the dataset, 217.596. The loss for static NOTEARS with $\lambda_{\mathbf{W}} = 0.1$ is 167.784.

D DREAM4 application

D.1 Cross validation

We use 10-fold cross validation to select the regularization parameters $\lambda_{\mathbf{W}}$ and $\lambda_{\mathbf{A}}$. The number of folds is a natural choice, as each dataset consists of 10 separate time series evaluated at 21 time steps for the same 100 genes. We evaluate performance on the validation set using the root mean squared error (RMSE). We find that for sufficiently low $\lambda_{\mathbf{A}}$, the RMSE is not sensitive to the values of $\lambda_{\mathbf{W}}$ (see Figure D.1). A plausible explanation is that our data consist of time steps separated at intervals of 50, but the underlying process is at a slower scale; the model therefore predominantly captures lagged inter-slice directed edges. As such, we set $\lambda_{\mathbf{W}}$ to the largest value in the range of optimal RMSE to enforce sparsity and simplicity. We apply cross validation separately for each of the 5 datasets in DREAM4, and we use the optimal parameters in each case to compute the average AUPR and AUROC.

D.2 Comparison to other methods

In tables D.1 and D.2, we compare the performance of DYNOTEARS to that of other methods. We obtain performance metrics for other algorithms from Lu et al. (2019).

The performance of our method can be compared to other solvers. Note that our model returns two matrices, $\mathbf{W}$ and $\mathbf{A}$, which can be interpreted as learning fast-acting ($\mathbf{W}$) and slow-acting ($\mathbf{A}$) influences. As such, we combine the two matrices by an element-wise sum to generate our final weight matrix. The results are summarized in Table D.1 and Table D.2. Note that there are missing values in the table because they were not initially reported.

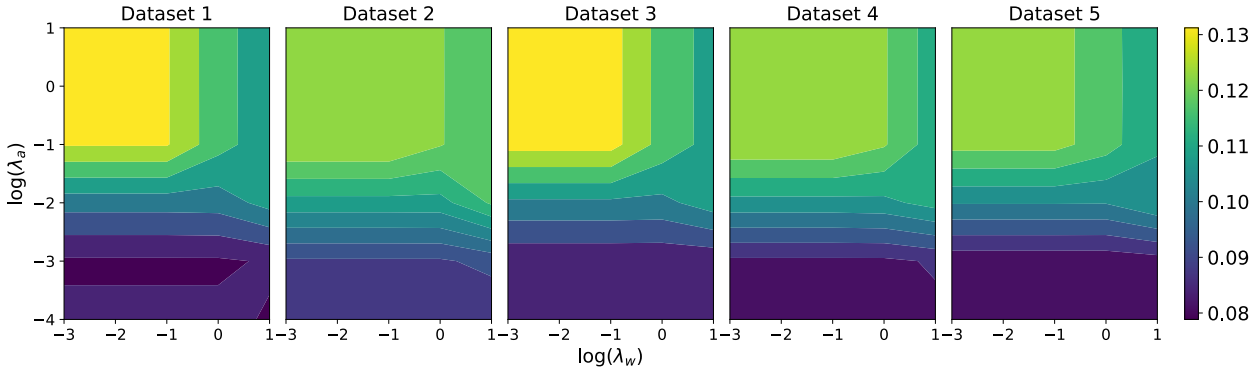


Figure D.1: Heatmaps of cross-validation RMSE for the 5 DREAM4 datasets across a range of $\lambda_{\mathbf{A}}$ and $\lambda_{\mathbf{W}}$ parameters.

Algorithm	Method Type	Average AUROC	STD AUROC	Network 1	Network 2	Network 3	Network 4	Network 5	Overall Rank	DBN Rank
DYNOTEARS	**DBN**	**0.664**	**0.047**	**0.748**	**0.612**	**0.634**	**0.674**	**0.653**	**8**	**2**
Ebdbnet	DBN	0.643							11	4
G1DBN	DBN	0.676	0.030	0.680	0.640	0.680	0.660	0.720	6	1
ScanBMA	DBN	0.657							10	3
VBSSMa	DBN	0.624	0.060	0.590	0.560	0.590	0.670	0.710	13	5
VBSSMb	DBN	0.618	0.060	0.560	0.570	0.620	0.640	0.700	14	6
Jump3	DT	0.720	0.040	0.770	0.670	0.740	0.680	0.740	2	
CSIc	GP	0.610	0.030	0.650	0.560	0.630	0.610	0.600	15	
CSId	GP	0.728	0.010	0.740	0.710	0.720	0.740	0.730	1	
GP4GRN	GP	0.686	0.040	0.720	0.620	0.700	0.700	0.690	4	
ARACNE	MI	0.558	0.010	0.560	0.540	0.560	0.550	0.580	18	
CLR	MI	0.678	0.030	0.700	0.630	0.710	0.670	0.680	5	
MRNET	MI	0.672	0.030	0.680	0.630	0.710	0.660	0.680	7	
TSNI	ODE	0.566	0.030	0.550	0.550	0.600	0.540	0.590	17	
BETS	VAR	0.688	0.060	0.780	0.650	0.640	0.700	0.670	3	
Enet	VAR	0.662	0.050	0.730	0.620	0.620	0.670	0.670	9	
GCCA	VAR	0.584	0.020	0.600	0.570	0.600	0.580	0.570	16	
LASSO	VAR	0.643							11	

Table D.1: AUROC scores of 18 structure-learning algorithms on the DREAM4 gene-expression dataset. Values for methods other than DYNOTEARS are from Lu et al. (2019).

Algorithm	Method Type	Average AUPR	STD AUPR	Network 1	Network 2	Network 3	Network 4	Network 5	Overall Rank	DBN Rank
DYNOTEARS	**DBN**	**0.173**	**0.041**	**0.235**	**0.110**	**0.177**	**0.188**	**0.155**	**4**	**1**
Ebdbnet	DBN	0.043							21	6
G1DBN	DBN	0.110	0.010	0.110	0.100	0.130	0.100	0.110	8	2
ScanBMA	DBN	0.101							9	3
VBSSMa	DBN	0.086	0.020	0.080	0.050	0.110	0.100	0.090	12	5
VBSSMb	DBN	0.096	0.030	0.090	0.060	0.120	0.120	0.090	11	4
dynGENIE3	DT	0.198	0.050	0.220	0.140	0.250	0.220	0.160	2	
GENIE3	DT	0.072	0.020	0.050	0.060	0.100	0.060	0.090	14	
Jump3	DT	0.182	0.050	0.260	0.110	0.190	0.170	0.180	3	
CSIc	GP	0.070	0.040	0.130	0.030	0.070	0.070	0.050	16	
CSId	GP	0.208	0.030	0.260	0.170	0.220	0.200	0.190	1	
GP4GRN	GP	0.162	0.050	0.220	0.100	0.160	0.210	0.120	6	
ARACNE	MI	0.046	0.010	0.030	0.040	0.060	0.040	0.060	20	
CLR	MI	0.072	0.020	0.050	0.060	0.110	0.060	0.080	14	
MRNET	MI	0.068	0.020	0.040	0.060	0.100	0.060	0.080	18	
tl-CLR	MI	0.168	0.050	0.180	0.110	0.240	0.150	0.160	5	
Inferelator	ODE	0.069	0.010	0.063	0.071	0.075	0.073	0.062	17	
TSNI	ODE	0.026	0.010	0.020	0.030	0.030	0.020	0.030	23	
BETS	VAR	0.128	0.020	0.160	0.100	0.130	0.140	0.110	7	
Enet	VAR	0.098	0.020	0.120	0.080	0.100	0.110	0.080	10	
GCCA	VAR	0.050	0.020	0.040	0.040	0.070	0.070	0.030	19	
LASSO	VAR	0.073							13	
OKVAR-Boost	VAR	0.034	0.020	0.050	0.050	0.030	0.020	0.020	22	

Table D.2: AUPR scores of 23 structure-learning algorithms on the DREAM4 gene-expression dataset. Values for methods other than DYNOTEARS are from Lu et al. (2019).

References

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Friedman, N., Murphy, K., and Russell, S. (1998). Learning the structure of dynamic probabilistic networks. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 139–147. Morgan Kaufmann Publishers Inc.

Hyvärinen, A., Zhang, K., Shimizu, S., and Hoyer, P. O. (2010). Estimation of a structural vector autoregression model using non-Gaussianity. *Journal of Machine Learning Research*, 11(May):1709–1731.

Lu, J., Dumitrascu, B., McDowell, I. C., Jo, B., Barrera, A., Hong, L. K., Leichter, S. M., Reddy, T. E., and Engelhardt, B. E. (2019). Causal network inference from gene transcriptional time series response to glucocorticoids. *bioRxiv*, 587170.

Malinsky, D. and Spirtes, P. (2018). Causal structure learning from multivariate time series in settings with unmeasured confounding. In *Proceedings of 2018 ACM SIGKDD Workshop on Causal Discovery*, pages 23–47.

Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley. AAI3082340.

Newman, M. E. J. (2018). *Networks*. Oxford University Press.

Ogarrio, J. M., Spirtes, P., and Ramsey, J. (2016). A hybrid causal search algorithm for latent variable models. In *Conference on Probabilistic Graphical Models*, pages 368–379.

Shimizu, S., Hoyer, P. O., Hyvärinen, A., and Kerminen, A. (2006). A linear non-Gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(Oct):2003–2030.

Zheng, X., Aragam, B., Ravikumar, P. K., and Xing, E. P. (2018). DAGs with NO TEARS: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems 31*, pages 9472–9483.