Discrete Learning Of DAGs Via Backpropagation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recently continuous relaxations have been proposed in order to learn directed acyclic graphs (DAGs) by backpropagation, instead of combinatorial optimization. However, a number of techniques for fully discrete backpropagation could instead be applied. In this paper, we explore this direction and propose DAG-DB, a framework for learning DAGs by Discrete Backpropagation, based on the architecture of Implicit Maximum Likelihood Estimation (I-MLE) [1]. DAG-DB performs competitively using either of two fully discrete backpropagation techniques, I-MLE itself, or straight–through estimation [2, 3].

1 Introduction

Aim. Directed acyclic graphs (DAGs) occur in a wide range of contexts, including project management, version control systems, evolutionary biology, and Bayesian networks. Bayesian networks have been a particularly popular subject for machine learning, including for the problem considered in this paper, learning a Bayesian network's DAG from data generated by the Bayesian network. This paper aims to learn DAGs using fully discrete backpropagation, avoiding the relaxation from discrete to continuous variables which is a common part of non-combinatoric DAG learning [for example, 4–16].

Bayesian networks. A Bayesian network associates a DAG with a random variable which has components indexed by the DAG nodes. A directed edge $i \longrightarrow j$ between nodes i and j represents a dependency of the random variable's j component on its i component. Discussion of Bayesian networks and DAGs may be found in textbooks [e.g. 17–19]; to establish our terminology, appendix A gives a very brief outline.

Contribution. Our contribution is to show that backpropagation methods which retain the fully discrete nature of a directed acyclic graph can be used to predict DAGs from data. We avoid the common approach of 'relaxing' digraph adjacency matrices from a binary to a real matrix. Instead, we sample discrete variables probabilistically and use methods of backpropagating that do not relax the variables to be continuous. No ethical issues are raised.

2 Related work and background

Learning DAGs from data. Most ways of learning DAGs from data can classed as combinatoric methods [reviewed in 20] or continuous optimisation methods [reviewed in 21]. Combinatoric methods may themselves be labelled as constraint–based, identifying the graph from conditional independence testing, and score–based, searching the space of possible graphs using a score function to evaluate search–results [20]. The PC algorithm [22] is a well–known constraint–based method, with a number of variant algorithms, for example PC-Stable [23] which reduces dependency on arbitrary node ordering. An example of a score–based approach is the Fast Greedy Equivalence Search (FGES)

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

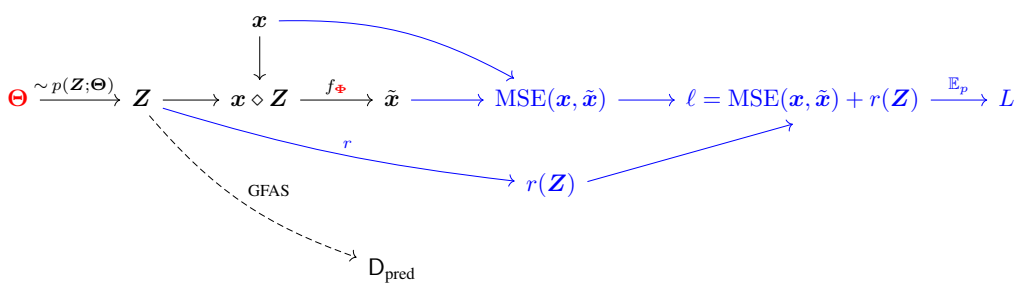

Figure 1: Overall DAG-DB architecture, including learnable parameters and loss calculation.

algorithm [24], which is an optimised version of the Greedy Equivalence Search (GES) [25, 26]. The development of NOTEARS in 2018 [4] for linearly–generated data signalled the start of widespread development of continuous optimisation approaches, including the GOLEM method [10] which is generally even more successful than NOTEARS for linear data with, for example, Gaussian noise. Many other continuous methods have recently been developed [for example, 5–9, 11–16] and the method presented in this paper, which uses gradient descent, may be viewed as related to these. However, continuous methods generally adopt some approach of 'relaxing' the discrete property of an edge being present or absent into a continuous variable, whereas our DAG-DB approach maintains edges as fully discrete, binary objects, even during our method's training phase. Our approach might be termed a 'probabilistic relaxation', and another example of this can be found in the SDI [27] framework, which focuses on contexts where data may be generated by interventions during training, whereas, here, our focus is on pre–generated 'observational' data.

Discrete backpropagation. A number of methods have also been developed to allow backpropagation without transforming discrete variables to be continuous in training. We take four such methods as examples. Straight–through estimation (STE) [2, 3] essentially ignores the discrete nature of the variables, and, as we will see, can be surprisingly effective. Score–function estimation (SFE) [28] uses the elementary rules of dfferentiation to estimate the gradient of an expectation by sampling with respect to a probability distribution. 'Blackbox' differentiation (BB) [29] is a way of adjusting continuous inputs to a combinatorial solver in order to mimic gradient descent for the discrete variable. Implicit maximum likelihood estimation (I-MLE) [1] incorporates noise into BB to handle discrete random variables. This allows combinatorial solvers to be used for maximum likelihood estimation. I-MLE also makes use of both STE and a seminal result by Domke [30]. We note that SDI [27], highlighted above, uses a particular Bernoulli–based SFE-like form of backpropagation [31].

Perturb–and–MAP sampling. We use the technique of perturb–and–MAP (P&M) sampling of random variables [32]. This approximates a parametrized distribution, say $p(\boldsymbol{Z};\boldsymbol{\Theta})$, with a random matrix variable $\boldsymbol{Z}$ and continuous parameter matrix $\boldsymbol{\Theta}$, as

$$p(\boldsymbol{Z};\boldsymbol{\Theta}) \sim \mathrm{MAP}(\boldsymbol{\Theta} + \tau\boldsymbol{\Psi}), \tag{1}$$

where each element of the noise matrix $\boldsymbol{\Psi}$ is i.i.d. sampled from a standard one–dimensional distribution, $\tau > 0$ is a temperature, and $\mathrm{MAP}(\boldsymbol{\Theta} + \tau\boldsymbol{\Psi})$ denotes the most likely value of $\boldsymbol{Z}$ with respect to $p(\boldsymbol{Z};\boldsymbol{\Theta} + \tau\boldsymbol{\Psi})$. P&M is useful when sampling $p(\boldsymbol{Z};\boldsymbol{\Theta})$ directly is intractable or expensive, and the MAP solver is cheap.

Maximum DAG. Our random variable $\boldsymbol{Z}$ will represent the adjacency matrix of a directed graph. To find a DAG from that digraph, we will use an approximate solution to the 'maximum DAG' problem. This considers a weighted digraph (each edge is assigned a real number) and the challenge is to remove edges from the digraph to form a DAG, whilst maximising the sum of the remaining edge–weights. We found it most efficient, and necessary as the number of digraph nodes rose toward 100, to use an approximate maximum DAG solver, 'Greedy Feedback Arc Set' (GFAS) [33], our Python implementation being adapted from (more optimised) Java code [34, 35].

3 Method

Framework. Figure 1 gives an overview of the DAG-DB framework. Suppose we wish to learn a DAG with d nodes from a dataset $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ of n data points $\boldsymbol{x} \in \mathbb{R}^d$. Let $\mathcal{R} \subset \mathbb{R}^{d\times d}$ and $\mathcal{B} \subset \{0,1\}^{d\times d}$ be the sets of zero–diagonal, respectively real and binary, matrices. A learnable vector $\boldsymbol{\Theta} \in \mathcal{R}$ parametrizes a exponential family distribution $p(\boldsymbol{Z};\boldsymbol{\Theta})$,

$$p(\boldsymbol{Z};\boldsymbol{\Theta}) = \exp\left(\langle \boldsymbol{Z},\boldsymbol{\Theta}\rangle_{\mathrm{F}}/\tau - \mathrm{A}(\boldsymbol{\Theta})\right), \tag{2}$$

where $\boldsymbol{Z} \in \mathcal{Z} \subseteq \mathcal{B}$ is a discrete matrix, $\tau > 0$ is a temperature, and $\mathrm{A}(\boldsymbol{\Theta})$ normalizes $p(\boldsymbol{Z};\boldsymbol{\Theta})$ to sum to one. Recognising the matrix form of $\boldsymbol{\Theta}$ and $\boldsymbol{Z}$, eq. (2) uses the Frobenius scalar product,

$$\langle \boldsymbol{Z},\boldsymbol{\Theta}\rangle_{\mathrm{F}} \coloneqq \mathrm{tr}\big(\boldsymbol{Z}^{\mathsf{T}}\boldsymbol{\Theta}\big) = \sum_{\mathrm{ij}} \boldsymbol{Z}_{\mathrm{ij}}\boldsymbol{\Theta}_{\mathrm{ij}}. \tag{3}$$

The matrix $\boldsymbol{Z}$ is interpreted as the adjacency matrix of a directed graph with, because of its zero diagonal, no self–loops. In the forward pass, samples $\boldsymbol{Z}^{(s)}$, $s = 1, ..., S$, from $p(\boldsymbol{Z};\boldsymbol{\Theta})$ are taken using the P&M sampling outlined in section 2. For $\mathcal{Z} = \mathcal{B}$, the MAP solver sets matrix elements of $\mathrm{MAP}(\boldsymbol{\Theta})$ to be one if $\boldsymbol{\Theta} > 0$ and zero otherwise. We found empirically that it is best to use the standard logistic distribution to generate the noise $\boldsymbol{\Psi}$ in eq. (1). This can also be theoretically justified as a good choice because of the binary nature of $\boldsymbol{Z}$'s matrix elements [32].

As is common for continuous optimisation methods, DAG learning proceeds by solving the problem of predicting values of $\boldsymbol{x}$ components $\boldsymbol{x}_{\mathrm{j}}$ from its values $\boldsymbol{x}_{\mathrm{i}}$ at 'parent' nodes with edges $\mathrm{i} \longrightarrow \mathrm{j}$. As set out in appendix C, this is ensured by the 'graphification' operation $\boldsymbol{x} \diamond \boldsymbol{Z}$ in the context of the linear map $f_{\boldsymbol{\Phi}}$. Elsewhere, this technique usually employs a weighted adjacency matrix with real elements [see, e.g., 4, 8, 10]; but we use the binary adjacency matrix $\boldsymbol{Z}$ without such a relaxation. To the extent we relax the problem, this is by treating the digraph probabilistically via the distribution $p(\boldsymbol{Z};\boldsymbol{\Theta})$, which, in practice, becomes concentrated around the most likely adjacency matrix.

The parameters $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$ are trained by feeding data points $\boldsymbol{x} \in \boldsymbol{X}$ into the model batchwise. The mean–squared error between $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$ is added to a regularizing function $r(\boldsymbol{Z})$ to give a loss ℓ associated with the sample $\boldsymbol{Z}$. The empirical mean of these losses, over the samples $\boldsymbol{Z}$ and the batch members $\boldsymbol{x}$, gives the batch loss L. Our learnt parameters $\boldsymbol{\Phi}$ and $\boldsymbol{\Theta}$ are then updated by backpropagation: while for $\boldsymbol{\Phi}$ this is standard backpropagation, a technique needs to be chosen for discrete backpropagation from $\boldsymbol{Z}$ to $\boldsymbol{\Theta}$. We found I-MLE [1] and straight–through estimation [2, 3] to be useful techniques for such backpropagation; we had less success here with score–function estimation [28] and 'blackbox' differentiation [29]. Appendix B gives details of I-MLE and straight–through estimation as used in DAG-DB.

After the final epoch of training, the GFAS algorithm is applied. We take the single most probable directed graph $\boldsymbol{Z} = \mathrm{MAP}(\boldsymbol{\Theta})$, and then find its maximum DAG when weighted by $\boldsymbol{\Theta}$. Tests for 30 nodes and fewer, for which we could use an exact maximum DAG solver we built using MiniZinc [36, 37], suggested that the solutions of our solver were in practice exact, or close to exact, due, most likely, to regularizing (see below) $\boldsymbol{Z}$ towards being a DAG. Because of computational cost, for all but one of our experiments, we use GFAS only at evaluation after the final epoch.

Regularization. DAG-DB can employ both functional and constraint regularization. Functional regularization is based on that of NOTEARS [4], and provided by the function

$$r(\boldsymbol{Z}) = \rho_{\mathrm{DAG}}\, r_{\mathrm{DAG}}(\boldsymbol{Z}) + \rho_{\mathrm{sp}}\, r_{\mathrm{sp}}(\boldsymbol{Z}), \tag{4}$$

where $\rho_{\mathrm{DAG}}, \rho_{\mathrm{sp}} > 0$ are strength coefficients. For the DAG-regularizer r_{DAG}, we use the binary version of NOTEARS' DAG regularizer,

$$r_{\mathrm{DAG}}(\boldsymbol{Z}) = \left[\mathrm{tr}\left(\exp \boldsymbol{Z}\right) - d\right]^2, \tag{5}$$

which has been shown [4, prop. 2] to vanish if and only if $\boldsymbol{Z}$ is the adjacency matrix for a DAG. The second regularizer, r_{sp}, is an L^1-regularizer promoting sparsity by summing $\boldsymbol{Z}$'s matrix elements. The other form of regularization is to constrain $\boldsymbol{Z}$ to within a proper subset $\mathcal{Z} \subsetneq \mathcal{B}$. We optionally introduce a maximum size of digraph, constraining $\boldsymbol{Z}$ to have no more than a given maximum number M of non–zero entries (edges). It is here that P&M sampling becomes particularly helpful: with no such constraint, we could have used Bernoulli sampling instead of P&M; but, with a maximum size, there is no tractable direct sampling method. For a maximum size M, the MAP solver identifies the M biggest elements of $\boldsymbol{\Theta}$, but drops any that are not positive. Matrix elements of $\mathrm{MAP}(\boldsymbol{\Theta})$ corresponding to the resulting index set have value one, and the remainder have value zero.

Table 1: Metrics for selected models for the Sachs [38] observational dataset of 853 data points. Best metric scores are in bold. See the text and appendices for discussion.

Model		$\mathrm{SHD_c}$	$\mathrm{prec_c}$	$\mathrm{rec_c}$	pred. size
STE_84	MAX_SIZE = 23	20	0.158	0.176	19
	MAX_SIZE = 8	15	0.600	0.176	5
IMLE_None	mean	12.7	0.869	0.255	5.0
	median	13	**1.000**	0.235	5
GOLEM		**11**	**1.000**	0.353	6
NOTEARS		**11**	0.467	**0.412**	15
FGES		**11**	0.750	0.353	8
PC		**11**	0.750	0.353	8

4 Experiments

Overview. As detailed in appendix D.3, we identified useful hyperparameters settings, STE_84, using STE with a maximum size 84, and IMLE_None, using I-MLE with no maximum size constraint. We then conducted ablation experiments, and tests on linearly–generated synthetic data, and on a real dataset. The main metric we use is $\mathrm{SHD_c}$, often normalised to $\mathrm{nSHD_c}$, complemented by a precision measure $\mathrm{prec_c}$ and, sometimes also by a recall measure $\mathrm{rec_c}$. Appendix D.2 gives more details on these metrics, as part of supplementary material on the experiments in appendix D.

Ablation experiments. We performed further ablation experiments to identify the relative role of the regularizers, with results shown in appendix D.4. We found that r_{DAG}, and, the appropriate setting or not of a maximum size regularizer, both have a notable good effect. However, the presence of a sparsity regularizer r_{sp}, although almost always helpful, was only very relevant for STE_84, and then only when the maximum size constraint was also ablated.

Synthetic data experiments. We generated random Erdös–Rényi 'ERk' and Barabási-–Albert 'SFk' DAGs and further generated data from these using a Gaussian equal–variance linear additive noise model. The number of edges possible in these synthetic graphs varies considerably, so the maximum size limit for STE_84 was adjusted in proportion to the expected size. We tested our methods against those highlighted in section 2 on sets of 24 random graphs, with results shown in fig. 2. We can see that the DAG-DB methods usually outperform the combinatoric approaches, and they are themselves outperformed by the other continuous methods. Of our methods, STE_84 performs better, with the maximum size adjustments noted above, while IMLE_None's performance is fairly close, except for ER4 and SF4 with $d = 100$ nodes, where its metrics worsen noticeably.

Real data experiments. We performed experiments on the Sachs cellular biochemistry dataset, representable as a DAG with 11 nodes and 17 edges [38]. Performance noted previously [4, 10] suggests that linear models perform reasonably on Sachs' 853-data point set of purely observational data. Table 1 compares our results against other methods. For STE_84, the table also indicates maximum size constraints which have to be reset, by a proportioning method (23) and with reference to typical, relatively successful, sizes predicted by other methods (8). We can see that IMLE_None is much more successful than either STE_84 setting, and, to account for a stochastic element in DAG-DB results, we therefore show, for I-MLE only, the results over 24 runs on the same Sachs data. We can see from the table that IMLE_None's performance is quite close to that of GOLEM and NOTEARS which are the best of the comparison methods. Standard deviations are discussed in appendix D.7.

We also ran a method IMLE_None_Tr, which, unlike our other DAG-DB methods, applies GFAS to make $\boldsymbol{Z}$ the adjacency matrix of a DAG, *in training*. In light of the associated compute time, we did not undertake a full hyperparameter search for IMLE_None, and instead only vary a Domke–related hyperparameter to which I-MLE can be sensitive. Generally digraph training performs marginally better on $\mathrm{SHD_c}$ and often markedly better on $\mathrm{prec_c}$ and $\mathrm{rec_c}$. This may be because it was not practical to optimise hyperparameters for DAG training. The results can be found in appendix D.7.

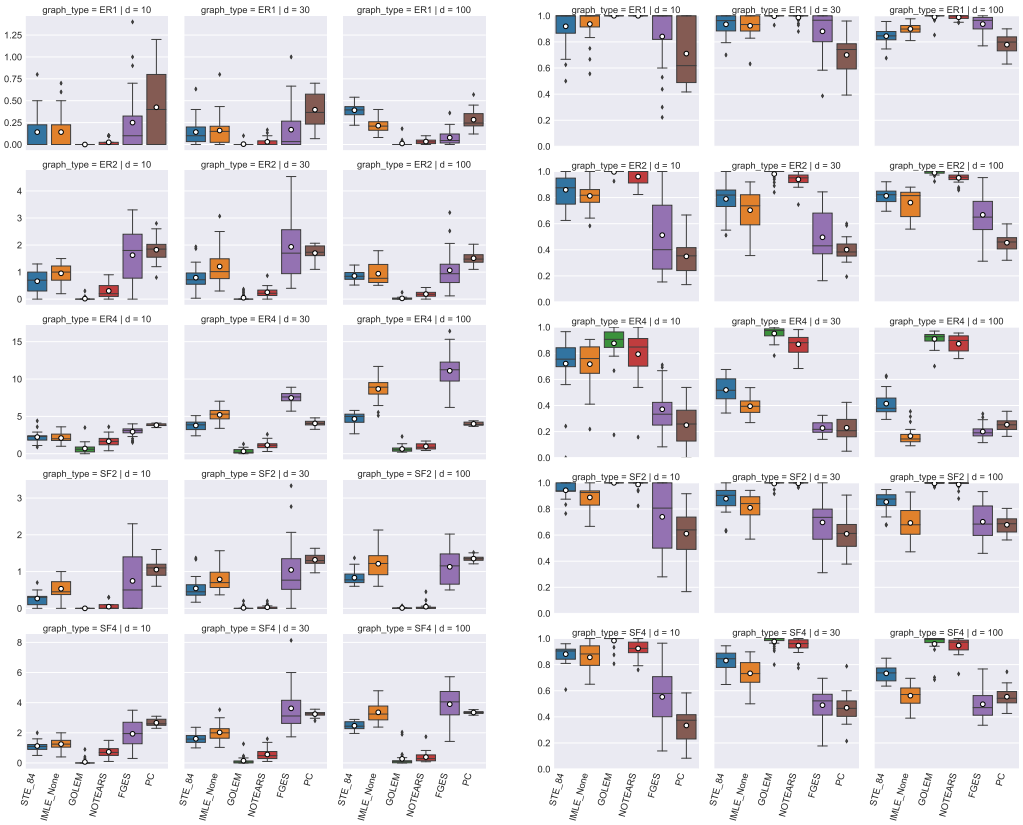

Figure 2: Tests of eight methods against selected graph type *(rows)* and numbers of nodes d *(columns)*. The two panels show $\mathrm{nSHD_c}$ *(left)* for which low values are good, and $\mathrm{prec_c}$ *(right)* for which high values are good. Vertical axis scales vary by panel, and, in the left–hand panel, also by row. The 'box–and–whiskers' indicate spread over 24 test DAGs (see appendix D.5 for details).

5 Conclusion and future work

For linear data, DAG-DB mostly outperforms combinatoric methods tested here, but is itself out–performed by the continuous methods. The DAG-DB framework should adapt to data generated by non–linear models and, potentially, to discrete data, or to interventional causal models [18]. It would also be interesting to see if the SFE-like backpropagation [31] used in SDI [27] would work for DAG-DB. Separately, one of the I-MLE paper's experiments [1] built a variational autoencoder (VAE), and a similar experiment could be performed with a latent space of DAGs. A VAE might learn a single DAG for a given dataset, and also learn values for each data point on that DAG's nodes. This might be interpretable as learning a discrete hierarchy representing characteristics of the dataset.

References

[1] Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: backpropagating through discrete exponential family distributions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14567–14579. Curran Associates, Inc., 2021. 2106.01798.

[2] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264(1):2146–2153, 2012.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. 1308.3432.

[4] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. DAGs with NO TEARS: Continuous optimization for structure learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi,

and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 1803.01422.

[5] Yue Yu, Jie Chen, Tian Gao, and Mo Yu. DAG-GNN: DAG structure learning with graph neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7154–7163, 2019. 1904.10098.

[6] Sébastien Lachapelle, Philippe Brouillard, Tristan Deleu, and Simon Lacoste-Julien. Gradient–based neural DAG learning, 2019. 1906.02226.

[7] Xun Zheng, Chen Dan, Bryon Aragam, Pradeep Ravikumar, and Eric Xing. Learning sparse nonparametric DAGs. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3414–3425, 2020.

[8] Ignavier Ng, Shengyu Zhu, Zhuangyan Fang, Haoyang Li, Zhitang Chen, and Jun Wang. Masked gradient–based causal structure learning. In Arindam Banerjee, Zhi-Hua Zhou, Evangelos E. Papalexakis, and Matteo Riondato, editors, *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 424–432, 2019. 1910.08527.

[9] Ignavier Ng, Shengyu Zhu, Zhitang Chen, and Zhuangyan Fang. A graph autoencoder approach to causal structure learning, 2019. 1911.07420.

[10] Ignavier Ng, AmirEmad Ghassami, and Kun Zhang. On the role of sparsity and DAG constraints for learning linear DAGs. In *Advances in Neural Information Processing Systems*, volume 33, pages 17943–17954. Curran Associates, Inc., 2020. 2006.10201.

[11] Ilyes Khemakhem, Ricardo Pio Monti, Robert Leech, and Aapo Hyvärinen. Causal autoregressive flows. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, 2021. 2011.02268.

[12] Yue Yu, Tian Gao, Naiyu Yin, and Qiang Ji. DAGs with No Curl: An efficient DAG structure learning approach. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12156–12166, 2021. 2106.07197.

[13] Chris Cundy, Aditya Grover, and Stefano Ermon. BCD Nets: Scalable variational approaches for Bayesian causal discovery. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 7095–7110. Curran Associates, Inc., 2021. 2112.02761.

[14] Bertrand Charpentier, Simon Kibler, and Stephan Günnemann. Differentiable DAG sampling. In *International Conference on Learning Representations*, 2022. 2203.08509.

[15] Valentina Zantedeschi, Jean Kaddour, Luca Franceschi, Matt Kusner, and Vlad Niculae. DAG learning on the permutahedron. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022.

[16] Yiwen Cao, Kui Yu, Xiaoling Huang, and Yujie Wang. A new skeleton–neural DAG learning approach. In João Gama, Tianrui Li, Yang Yu, Enhong Chen, Yu Zheng, and Fei Teng, editors, *Pacific–Asia Conference on Knowledge Discovery and Data Mining*, pages 626–638. Springer International Publishing, 2022.

[17] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[18] Judea Pearl. *Causality*. Cambridge University Press, second edition, 2013.

[19] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[20] Neville K. Kitson, Anthony C. Constantinou, Zhigao Guo, Yang Liu, and Kiattikun Chobtham. A survey of Bayesian network structure learning, 2021. 2109.11415.

[21] Matthew J. Vowels, Necati Cihan Camgoz, and Richard Bowden. D'ya like DAGs? A survey on structure learning and causal discovery. *ACM Comput. Surv.*, Mar 2022. 2103.02582. Just Accepted.

[22] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

[23] Diego Colombo, Marloes H Maathuis, et al. Order–independent constraint–based causal structure learning. *J. Mach. Learn. Res.*, 15(1):3741–3782, 2014. 1211.3295.

[24] Joseph Ramsey, Madelyn Glymour, Ruben Sanchez-Romero, and Clark Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high–dimensional graphical causal models, with an application to functional magnetic resonance images. *International journal of data science and analytics*, 3(2):121–129, 2017.

[25] Christopher Meek. *Graphical Models: Selecting causal and statistical models*. PhD thesis, PhD thesis, Carnegie Mellon University, 1997.

[26] David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

[27] Nan Rosemary Ke, Olexa Bilaniuk, Anirudh Goyal, Stefan Bauer, Hugo Larochelle, Bernhard Schölkopf, Michael C. Mozer, Chris Pal, and Yoshua Bengio. Learning neural causal models from unknown interventions, 2019. 1910.01075.

[28] Ronald J Williams. Simple statistical gradient–following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[29] Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020. 1912.02175.

[30] Justin Domke. Implicit differentiation by perturbation. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[31] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Nan Rosemary Ke, Sebastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*, 2020. 1901.10912.

[32] George Papandreou and Alan L Yuille. Perturb–and–MAP random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pages 193–200. IEEE, 2011.

[33] Peter Eades, Xuemin Lin, and William F Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993.

[34] Michael Simpson, Venkatesh Srinivasan, and Alex Thomo. Efficient computation of feedback arc set at web–scale. *Proceedings of the VLDB Endowment*, 10(3):133–144, 2016.

[35] Michael Simpson. Efficient computation of feedback arc set at web–scale GitHub repo. Accessed on 23 Sept. 2022.

[36] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[37] The MiniZinc Team. MiniZinc documentation. Accessed on 22 Sept. 2022.

[38] Karen Sachs, Omar Perez, Dana Pe'er, Douglas A Lauffenburger, and Garry P Nolan. Causal protein–signaling networks derived from multiparameter single–cell data. *Science*, 308(5721):523–529, 2005.

[39] Ignavier Ng. GOLEM GitHub repo, 2021. Accessed on 22 Aug. 2022.

[40] Xun Zheng. NOTEARS GitHub repo, 2020. Accessed on 22 Aug. 2022.

[41] Byron Ellis and Wing Hung Wong. Learning causal Bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103(482):778–789, 2008.

[42] Alexander G. Reisach, Christof Seiler, and Sebastian Weichwald. Beware of the simulated DAG! Causal discovery benchmarks may be easy to game, 2021. 2102.13647.

[43] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next–generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 2623–2631. Association for Computing Machinery, 2019.

[44] Optuna contributors. Optuna documentation, 2018. Accessed on 18 Aug. 2022.

[45] Center for Causal Discovery. Fast greedy equivalence search (FGES) algorithm for continuous variables, 2017. Accessed on 2 Aug. 2022.

[46] Joseph Ramsey, Jiji Zhang, and Peter L. Spirtes. Adjacency–faithfulness and conservative causal inference, 2006. 1206.6843.

[47] `bd2kccd`. `py-causal` GitHub repo, 2019. Accessed on 22 Aug. 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes]
(c) Did you discuss any potential negative societal impacts of your work? [N/A]
(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Notation and terminology

Notation. We write vectors in bold, like $\boldsymbol{x}$. Matrices are in bold capitals, like $\boldsymbol{Z}$ or $\boldsymbol{\Theta}$. Random variables, like U, are in non–bold capitals even if they yield vectors. Graph–related objects appear in sans serif, as in D or j. Non–standard sets are in calligraphic capitals, such as $\mathcal{R}$. Components of vectors and elements of matrices retain their bold typeface and their case, for example, $\boldsymbol{z}_i$ or $\boldsymbol{\Theta}_{\mathsf{ij}}$.

Graph terminology. To fix terminology, a directed graph, or digraph, $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ comprises a set of nodes V and a set of edges $\mathsf{E} \subset \mathsf{V} \times \mathsf{V}$. An edge $\mathsf{e} = (\mathsf{i}, \mathsf{j}) \in \mathsf{E}$ may be represented as either $\mathsf{i} \longrightarrow \mathsf{j}$ or $\mathsf{j} \longleftarrow \mathsf{i}$. We do not allow self–loops $\mathsf{i} \longrightarrow \mathsf{i}$. A directed cycle is an aligned sequence of edges $\mathsf{C} = \mathsf{i}_0 \longrightarrow \mathsf{i}_1 \longrightarrow \mathsf{i}_2 \longrightarrow \cdots \longrightarrow \mathsf{i}_{c-1} \longrightarrow \mathsf{i}_0$, for some positive integer, c.

A directed acyclic graph (DAG) is a digraph with no directed cycles. A partially–directed graph extends the concept of a digraph to allow undirected edges $\mathsf{i} \text{——} \mathsf{j}$. A partially–directed acyclic graph (PDAG) is a partially–directed graph with no *directed* cycles: cycles including undirected edges are allowed [see 19, p. 82].

A Bayesian network, $M = (\mathsf{D}, U)$ consists of a DAG $\mathsf{D} = (\mathsf{V}, \mathsf{E})$ and a vector of random variables indexed by the nodes $U = (U_{\mathsf{j}})_{\mathsf{j} \in \mathsf{V}}$, such that,

$$p(U) = \prod_{\mathsf{j} \in \mathsf{V}} p(U_{\mathsf{j}} \,|\, \{U_{\mathsf{i}} : \mathsf{i} \longrightarrow \mathsf{j} \in \mathsf{E}\}), \tag{6}$$

so the probability for U_{j} at a node j only directly depends on the values of U at the j's 'parent' nodes.

B Discrete backpropagation

Sampling. In the forward pass, via eq. (1), DAG-DB generates i.i.d. P&M samples $\boldsymbol{Z}^{(s)} = \mathrm{MAP}(\boldsymbol{\Theta} + \tau \boldsymbol{\Psi}^{(s)})$, $s = 1, ..., S$, of the binary adjacency matrix, where τ is the exponential family distribution temperature from eq. (2). For each matrix element ij, the noise samples $\boldsymbol{\Psi}^{(s)}_{\mathrm{ij}}$ are generated by the standard logistic distribution. DAG-DB then uses a method such as STE or I-MLE to backpropagate the gradient of resulting empirical loss function, $L = S^{-1} \sum_{s=1}^{S} \ell_{\boldsymbol{Z}^{(s)}}$, from the samples $\boldsymbol{Z}^{(s)}$ to $\boldsymbol{\Theta}$.

Blackbox estimation (BB) [29] can be considered as a special case of I-MLE: with no P&M sampling, it considers a purely deterministic function from $\boldsymbol{\Theta}$ to $\boldsymbol{Z} = \mathrm{MAP}(\boldsymbol{\Theta})$ [1]. BB's inability to deliver good results suggests that the sampling element of DAG-DB plays a useful role. Tracking $\boldsymbol{\Theta}$ components shows that $p(\boldsymbol{Z}; \boldsymbol{\Theta})$ becomes more concentrated around its MAP value as training develops: an initial, broad distribution helps exploration, and, as training progresses, the distribution concentrates around the predicted digraph $\boldsymbol{Z}$.

The maximum size constraint show the versatility of P&M. With no maximum size constraint, P&M logistic noise sampling is the same as independent Bernoulli sampling for each (non–diagonal) component $\boldsymbol{Z}_{\mathrm{ij}}$ [see, for example, 32]. With such a constraint, Bernoulli sampling is not possible because the $\boldsymbol{Z}_{\mathrm{ij}}$ components are no longer independent.

Straight–through estimation (STE). The STE approximation for backpropagation is given by

$$\nabla_{\boldsymbol{\Theta}} L \approx \tau^{-1} \sum_{s=1}^{S} \nabla_{\boldsymbol{Z}^{(s)}} \ell. \tag{7}$$

This is as in the original treatments of STE [2, 3], with the inclusion of a factor to account for the temperature.

Implicit maximum likelihood estimation (I-MLE). I-MLE [1] uses a procedure due to Domke [30] to set target distribution parameters, which, for DAG-DB, are, for $s = 1, .., S$,

$$\boldsymbol{\Theta}^{(s)} = \boldsymbol{\Theta} - \lambda \nabla_{\boldsymbol{Z}^{(s)}} \ell, \tag{8}$$

where λ arises from the Domke procedure. In I-MLE, λ is treated a hyperparameter, with the approximation of the loss gradient with respect to $\boldsymbol{\Theta}$ then being

$$\nabla_{\boldsymbol{\Theta}} L \approx \frac{1}{\lambda \tau S} \sum_{s=1}^{S} \left[\mathrm{MAP}(\boldsymbol{\Theta} + \tau \boldsymbol{\Psi}^{(s)}) - \mathrm{MAP}(\boldsymbol{\Theta}^{(s)} + \tau \boldsymbol{\Psi}^{(s)}) \right]. \tag{9}$$

C Details of graphification

Each data point is combined with every sample $\boldsymbol{Z} = \boldsymbol{Z}^{(s)}$ by the 'graphification'

$$(\boldsymbol{x} \diamond \boldsymbol{Z})_{\mathrm{ij}} \coloneqq \boldsymbol{x}_{\mathrm{i}} \boldsymbol{Z}_{\mathrm{ij}}. \tag{10}$$

The resulting matrix is then fed into a function $f_{\boldsymbol{\Phi}}$, where the $\boldsymbol{\Phi}$ are learnable parameters. In principle $f_{\boldsymbol{\Phi}}$ could be a multi–layer neural network, for predictions on non–linear Bayesian networks, but, for simplicity, we confine attention to a linear function with no bias, having $\boldsymbol{\Phi} \in \mathcal{R}$ as real–matrix parameters, and

$$[f_{\boldsymbol{\Phi}}(\boldsymbol{M})]_{\mathrm{j}} \coloneqq \sum_{\mathrm{i}} \boldsymbol{\Phi}_{\mathrm{ij}} \boldsymbol{M}_{\mathrm{ij}}, \quad \text{(with no sum over j)}, \tag{11}$$

for a $d \times d$ matrix $\boldsymbol{M}$. Given the digraph $\boldsymbol{Z}$, define the parents of node j as $\mathrm{pa}_{\boldsymbol{Z}}(\mathrm{j}) \coloneqq \{\mathrm{i} : \boldsymbol{Z}_{\mathrm{ij}} = 1\}$. Equations (10) and (11) then together imply a prediction for the value of $\boldsymbol{x}$ at node j from the values of $\boldsymbol{x}$ its parent nodes,

$$\tilde{\boldsymbol{x}}_{\mathrm{j}} \coloneqq [f_{\boldsymbol{\Phi}}(\boldsymbol{x} \diamond \boldsymbol{Z})]_{\mathrm{j}} = \sum_{\mathrm{i} \in \mathrm{pa}_{\boldsymbol{Z}}(\mathrm{j})} \boldsymbol{x}_{\mathrm{i}} \boldsymbol{\Phi}_{\mathrm{ij}}, \quad \text{(with no sum over j)}. \tag{12}$$

This ensures that, in prediction, a node j's value will only be a function of values associated with its parent nodes.

D Details of experiments

D.1 Synthetic DAGs and data

Summary. Synthetic data is often used for learning hyperparameters and testing as it can be easily and cheaply generated. We follow the approach in the NOTEARS paper [4], also adopted by GOLEM [10] and many other papers. The approach is to generate a random DAG, based on an underlying distribution for randomly–generating graphs or digraphs, and then to randomly–generate an appropriate linear additive noise model (LANM) on that DAG. The LANM is then used to create the synthetic dataset, the $n \times d$ matrix $\boldsymbol{X}$. To generate synthetic DAGs and data, we used open–source code from the GOLEM paper [39], itself derived from NOTEARS [40].

Generating DAGs. The NOTEARS and GOLEM papers consider two types of randomly–generated DAGs: Erdös–Rényi ERk DAGs, where the number of edges is binomially distributed, and in expectation, there are dk edges; and Barabási-–Albert SFk DAGs, where the number of edges is fixed, at $k[d-(k+1)/2]$, with some nodes being preferred in terms of having more edges. We use both these types in our experiments.

Generating data. We use a standard Gaussian equal–variance linear additive noise model. Such a model, with variance $\sigma^2 \in \mathbb{R}^+$, is a Bayesian network $M = (\mathsf{D}, U)$ with values $\boldsymbol{u} \sim U$ having each node–component $\boldsymbol{u}_{\mathsf{j}} \in \mathbb{R}$, and satisfying

$$\boldsymbol{u}_{\mathsf{j}} = \left(\sum_{\mathsf{i} \in \mathrm{pa}(\mathsf{j})} \boldsymbol{\Phi}_{\mathsf{ij}}\, \boldsymbol{u}_{\mathsf{i}} \right) + \boldsymbol{\nu}_{\mathsf{j}}, \qquad \boldsymbol{\nu}_{\mathsf{j}} \sim \mathcal{N}(0, \sigma^2) \text{ i.i.d.}, \tag{13}$$

for some weights matrix $\boldsymbol{\Phi}$. The NOTEARS/GOLEM implementation, which we use, creates such a model by sampling $\boldsymbol{\Phi}_{\mathsf{ij}}$ i.i.d. uniformly randomly from $[-2, -0.5] \cup [0.5, 2]$.

Potential limitations. There have been criticisms of this approach. In 2008, it was noted that a similar approach to DAG generation does not generate DAGs uniformly [41, s. 3.3], and this issue also affects the current DAG–generation method. It has also, more recently, been pointed out that linear additive noise models offer clues to prediction methods which may be undesirable. These clues are tendencies for variances and co–variances between nodes to increase as we follow the DAG's edges [42]. Nonetheless these approaches remain the standard synthetic data benchmark and we retain their use, supplemented by our real data experiment, which may mitigate these concerns.

D.2 Metrics

CPDAGs. Different DAGs may represent the same Bayesian network, being, in general, impossible to distinguish simply by observation — that is, by i.i.d. sampling of the Bayesian network's random variable. The Markov Equivalence Class (MEC) of a DAG D is formed by the DAGs D' for which there exists a Bayesian network which may be represented by both D and D'. A MEC may be represented by the DAG's class partially–directed acyclic graph (CPDAG) in which some of the DAG's edges become undirected. All DAGs in a MEC have the same skeleton: that is, the same edges, ignoring direction. Any MEC can be represented by a CPDAG: undirected edges indicate that the direction may differ between DAGs within the MEC.

Identifiability For some particular Bayesian networks, however, the exact DAG can be identified. Gaussian equal–variance linear noise models have this property. For real data, such as the Sachs data we experiment on, there is no such guarantee. For consistency, we will therefore use metrics based on CPDAGs. More discussion of identifiability can be found in Peters, Janzing & Schölkopf's textbook [19, sec. 7.1.2–4].

Structural Hamming distance (SHD). Suppose now that we have a true DAG $\mathsf{D}_{\text{true}} = (\mathsf{V}, \mathsf{E}_{\text{true}})$ and a predicted DAG $\mathsf{D}_{\text{pred}} = (\mathsf{V}, \mathsf{E}_{\text{pred}})$, on the same set of nodes V, but with potentially differing edge sets E_{true} and E_{pred}. A given unordered pair of nodes $\{\mathsf{i}, \mathsf{j}\}$'s join status will be one of the following: unjoined, joined $\mathsf{i} \longrightarrow \mathsf{j}$, or joined $\mathsf{i} \longleftarrow \mathsf{j}$. If we also allow undirected edges, as in CPDAGs, then this adds a fourth possibility $\mathsf{i} \text{——} \mathsf{j}$. SHD is the count of the ordered pairs i, j which differ in

terms of join status between D_{true} and D_{pred}. The class structural Hamming distance $\mathrm{SHD_c}$ is the corresponding metric for the CPDAGs associated with D_{true} and D_{pred}. Dividing by the number of nodes gives the normalized class structural Hamming distance $\mathrm{nSHD_c}$. (n)SHD is found in, for example, the NOTEARS [4] and GOLEM [10] papers, and $(\mathrm{n})\mathrm{SHD_c}$ in the GOLEM paper. Note that if we predicted an empty DAG D_{pred}, with no edges, we would have $\mathrm{nSHD_c} = |\mathsf{E}_{\text{true}}|/|\mathsf{V}|$, because a CPDAG has the same skeleton, and so the same number of edges, as any DAG it represents.

Precision and recall. In the current context, precision is defined as

$$\mathrm{prec}(\mathsf{D}_{\text{true}}, \mathsf{D}_{\text{pred}}) = \frac{|\mathsf{E}_{\text{pred}} \cap \mathsf{E}_{\text{true}}|}{\max(1, |\mathsf{E}_{\text{pred}}|)}, \tag{14}$$

treating the edges $\mathsf{i} \longrightarrow \mathsf{j}$ and $\mathsf{i} \longleftarrow \mathsf{j}$ as distinct. Similarly, recall is defined as

$$\mathrm{rec}(\mathsf{D}_{\text{true}}, \mathsf{D}_{\text{pred}}) = \frac{|\mathsf{E}_{\text{pred}} \cap \mathsf{E}_{\text{true}}|}{\max(1, |\mathsf{E}_{\text{true}}|)}. \tag{15}$$

In graph literature [e.g. 4, 10], precision may be replaced by the false discovery rate, which is one minus the precision, and recall may be termed the true positive rate. As for SHD, we may also use the CPDAGs associated with D_{true} and D_{pred} to define the class precision $\mathrm{prec_c}$ and class recall $\mathrm{rec_c}$. In this extension to CPDAGs, we regard $\mathsf{i} - \mathsf{j}$ as a further distinct type of edge between i and j.

D.3 Hyperparameters

Hyperparameter settings. To choose hyperparameters, we sought to optimise on a randomly–generated set of six synthetic Erdös–Rényi 'ER2' DAGs with $d = 30$ nodes and a mean of 60 edges. On each of these graphs, a synthetic dataset was created using a Gaussian equal–variance linear additive noise model. We assessed the viability of I-MLE, STE, SFE and BB as the discrete backpropagation method, finding I-MLE and STE to be the most promising. In exploring hyperparameter settings, we used a combination of Optuna [43, 44], a well–known package for Bayesian optimisation of hyperparameters, and manual adjustment, for example taking an Optuna–generated set of hyperparameters and varying the standard probability distribution used as noise for P&M. We also performed a grid search over hyperparameters, which was less successful in identifying optimal settings. We identified two settings as particularly promising, `STE_84`, using STE with a maximum size constraint of 84 edges, and `IMLE_None`, using I-MLE with no maximum size constraint.

We chose synthetic ER2 DAGs with 30 nodes, as lying roughly in the middle of the range of types and node–numbers considered here, and in the NOTEARS and GOLEM papers. Note that, following appendix D.2, a DAG with no edges would score an expected $\mathrm{nSHD_c} = 2$ when compared with a 'true' ER2 DAG, suggesting a cut–off maximum $\mathrm{nSHD_c}$. With BB, we did not find any settings which delivered $\mathrm{nSHD_c} < 2$; whilst for SFE, we only managed $\mathrm{nSHD_c}$ slightly less than 2. Table 2 shows the best hyperparameters for DAG-DB we found for `STE_84` and `IMLE_None`.

D.4 Ablation of regularizers

We can see from tables 3 and 4 that, for both `STE_84` and `IMLE_None` settings, all ablations worsen the metrics, although, for `IMLE_None` , the effect of ablating r_{sp} is very minor, as might be expected from table 2 which shows that its coefficient ρ_{sp} is rather small.

D.5 Variation by DAG

The standard box–and–whisker plots of fig. 2 indicate variation over 24 test DAGs. Each box covers from the first to the third quartile, with an interior line indicating the median. A white dot indicates the mean. The whiskers each extend to cover all points within a further 1.5 times the inter–quartile range, whilst any points outside that extended range are marked individually.

We also considered whether 24 DAGs was enough to give a reasonable sample, experimenting via a test set of 240 ER2 DAGs with 30 nodes. We found that taking a 100 000 random 24 DAG sub–samples of the 240 gave standard errors for $\mathrm{nSHD_c}$ of 0.1 for `STE_84` and 0.15 for `IMLE_None`, and of 0.03 for $\mathrm{prec_c}$ with either method. We then examined how much results could vary in repeated prediction on the same 24 DAGs. We made predictions ten times for a constant set of 24 DAGs,

Table 2: Hyperparameters for DAG-DB.

Symbol	Description	STE_84	IMLE_None
Basic set–up			
n	Total number of data points in data matrix $\boldsymbol{X}$	1000	
	Number of epochs training	1000	
	Whether to shuffle batches for each new epoch	True	
	Batch size	16	8
Discrete backprop			
	Method for backpropagation from discrete $\boldsymbol{Z}$ to continuous $\boldsymbol{\Theta}$	STE	I-MLE
S	Number of P&M samples	10	47
τ	Temperature in the exponential family distribution of eq. (2)	1.771×10^{-1}	8.786×10^{-1}
λ	I-MLE Domke hyperparameter	n/a	2.714×10^{1}
	Width of initial uniform distribution for each $\boldsymbol{\Theta}$ component. (Note that $\boldsymbol{\Phi}$ is initialised as for `torch.nn.Linear` layers)	2.169×10^{-1}	1.137×10^{-4}
Optimization			
	Optimizer	`torch.optim.Adam`, as in I-MLE [1]	
	Learning rate for $\boldsymbol{\Theta}$	1.134×10^{-4}	1.616×10^{-3}
	Learning rate for $\boldsymbol{\Phi}$ in $f_{\boldsymbol{\Phi}}$	1.232×10^{-2}	3.720×10^{-1}
Regularization			
ρ_{DAG}	Coefficient for the r_{DAG} regularizer	4.101×10^{-1}	1.575×10^{-1}
ρ_{sp}	Coefficient for the r_{sp} regularizer	1.023×10^{-2}	1.208×10^{-3}
M	Maximum size of digraph $\boldsymbol{Z}$	84	n/a

finding variation from minimum to maximum for $\mathrm{nSHD_c}$ of 0.03 for `STE_84` and 0.05 for `IMLE_None`, and of 0.01 for $\mathrm{prec_c}$ with either method. These results suggest that 24 DAGs may be a sufficient sample size for testing our methods.

D.6 Methods used for comparison

We follow the methods used for comparison in the GOLEM paper [10], with one exception. The combinatoric methods we use are FGES [24] (formerly known as FGS [45]) and PC [22]. The GOLEM paper used the Conservative PC algorithm [46] as its version of PC, however, this produces results in a form more general than a CPDAG, complicating comparisons. We therefore use the PC-Stable [23] variant, which, like the original PC algorithm, returns a CPDAG. The continuous methods we compare with are NOTEARS [4] and GOLEM [10]. For NOTEARS, as was done in the GOLEM paper, we employ the NOTEARS-L1 variant with the settings given as defaults for linear SEMs in the NOTEARS code repository. GOLEM provides two variants: GOLEM-EV, suitable for data generated with equal variance, and GOLEM-NV, suitable for non–equal variance data. Accordingly, we use GOLEM-EV for our synthetic experiments and GOLEM-NV for our real data experiment.

To run experiments for these methods, we used `py-causal` [47] for FGES and PC, and the NOTEARS [40] and GOLEM [39] repos. Licences for these packages, and for the code we adapted for GFAS, are shown in table 5.

Table 3: Effects of ablation for STE_84, with crosses indicating ablation of the relevant regularizer.

MAX_SIZE $= 84$	✗	✗	✗	✗	✓	✓	✓	✓
r_{DAG}	✗	✗	✓	✓	✗	✗	✓	✓
r_{sp}	✗	✓	✗	✓	✗	✓	✗	✓
nSHD_{c}	3.901	1.022	2.068	1.708	1.406	1.286	0.919	0.732
prec_{c}	0.282	0.724	0.481	0.552	0.565	0.643	0.719	0.814

Table 4: Effects of ablation for IMLE_None. Crosses are as in table 3, but note that, in this table, MAX_SIZE $=$ None is marked by ✓, while 'ablation' to MAX_SIZE $= 66$, which was found to be a useful maximum size for I-MLE, is marked by ✗.

MAX_SIZE $=$ None	✗	✗	✗	✗	✓	✓	✓	✓
r_{DAG}	✗	✗	✓	✓	✗	✗	✓	✓
r_{sp}	✗	✓	✗	✓	✗	✓	✗	✓
nSHD_{c}	1.425	1.411	1.268	1.293	1.401	1.311	1.064	1.058
prec_{c}	0.584	0.589	0.637	0.631	0.585	0.619	0.733	0.736

Table 5: Licences for comparison methods and GFAS.

Code	Ref.	Link to licence description
py-causal	[47]	GNU Lesser General Public License v2.1+
NOTEARS	[40]	Apache License 2.0
GOLEM	[39]	Apache License 2.0
GFAS	[35]	BSD License

D.7 Supplementary results on Sachs experiments

See table 6 for results from I-MLE, with training using digraphs, IMLE_None, and with training using DAGs, IMLE_None_Tr. Table 7 shows standard deviations, over 24 runs, for these methods. The varied hyperparameter, λ, is the Domke hyperparameter discussed in appendix B. We varied this as, in I-MLE, results can be particularly sensitive to λ [1]. Table 7's $\lambda = 27.14$ rows for IMLE_None are the standard deviations corresponding to table 1. GOLEM and NOTEARS results have very small standard deviations, whilst FGES and PC are deterministic.

As noted, DAG training is time consuming. Running time for each single prediction increases from about 0h15m with IMLE_None to around 1h15m with IMLE_None_Tr.

D.8 Computational resources

Computations were done on a combination of a laptop with a GeForce RTX 3080 Mobile 16GB GPU and a computer cluster. Prediction for a single ER2 graph on 30 nodes (roughly the mid–range of the graphs shown in fig. 2) took around 0h12 on the laptop, for the longest–running of all the methods in fig. 2, IMLE_None, giving around 72 hours to compile that element of the figure. The corresponding STE_84 computation took around a quarter of that time, and the other methods we compared with were quicker, giving a total time to compute fig. 2 of the order of 100 hours. Hyperparameter exploration, ablation experiments and Sachs experiments are estimated to add another 200 hours compute, while the experiments on variation described in appendix D.5 took around 120 hours.

Table 6: As for table 1, but comparing variants with and without DAGs in training for a range of values of λ. For each λ, the first row is the mean and the second row is the median. $\lambda = 27.14$ is the default setting for `IMLE_None`.

	`IMLE_None`				`IMLE_None_Tr`			
λ	SHD_c	prec_c	rec_c	pred. size	SHD_c	prec_c	rec_c	pred. size
0.01	12.0	1.000	0.294	5.0	13.3	0.778	0.314	7.0
	12	1.000	0.294	5	13.0	0.750	0.294	7
0.1	12.5	0.950	0.267	4.8	12.3	0.950	0.275	4.9
	12	1.000	0.294	5	12	1.000	0.294	5
1.0	12.3	0.823	0.275	5.7	13.6	0.584	0.208	6.1
	12.5	1.000	0.265	6	14	0.536	0.206	6
10.0	12.1	0.894	0.287	5.5	13.2	0.520	0.292	9.7
	12	1.000	0.294	6	13	0.500	0.294	9.5
27.14	12.7	0.869	0.255	5.0	13.8	0.475	0.287	10.5
	13	1.000	0.235	5	14	0.455	0.294	11
100.0	12.1	0.894	0.287	5.5	14.0	0.451	0.279	10.8
	12	1.000	0.294	5	14	0.455	0.294	10.5

Table 7: As for table 6, but showing standard deviations over the 24 runs.

	`IMLE_None`				`IMLE_None_Tr`			
λ	SHD_c	prec_c	rec_c	pred. size	SHD_c	prec_c	rec_c	pred. size
0.01 [a]	0.0	0.000	0.000	0.0	1.0	0.113	0.028	1.2
0.1	0.8	0.135	0.046	0.5	1.0	0.135	0.059	0.8
1.0	1.3	0.198	0.077	0.8	1.1	0.175	0.057	0.9
10.0	1.3	0.169	0.076	1.0	0.6	0.061	0.012	1.2
27.14	1.2	0.189	0.073	0.9	0.8	0.074	0.020	1.5
100.0	1.2	0.186	0.073	0.6	0.9	0.076	0.031	1.8

[a] Our data reports confirm that the zero standard deviation results for `IMLE_None` are indeed distinct runs.