Don't Be Afraid of the DAG

A 7.5 ECTS PUK

Sebastian Ø. Utecht (npd457)
Jakob H. Schauser (pwn274)

Supervisor:
Sebastian Weichwald

Contents

1 Introduction

Suppose you are handed a data set containing medical data on a drug and a number of diseases. How do you find their internal causal effects? This is essentially the field of structure learning: Finding causal relationships between different events, by recovering their graph representation (DAG) from data.

Benchmarking of the algorithms that perform structure learning is almost exclusively done on simulated data. However, simulated DAGs have a possible flaw: They can often be sorted by increasing marginal variance thereby recovering the causal order. This is a trait which has no theoretical backing in real life data.

In this project we will go through some of the intricacies of the apparent variance increase. We will do this by firstly describing the theory behind the problem and then proposing an array of methods for generating networks that do not exhibit the property. The networks will be laid out in increasing theoretical complexity as each was built upon knowledge gained from the creation of its predecessor.

2 Background & Theory

2.1 Causal Models & Bayesian networks

As this project deals greatly with causal models and Bayesian Networks (BN) we will in this section give a quick overview of the subject.

A causal model seeks to explain the causality between different events, one way this can be achieved is via the use of a causal BN. BN's are probabalistic graphical models, that allow for good intuitive overview of a system as well as means of applying methods of probability theory to a large number of variables. BN's are used in a wide array of fields from mapping gene-networks to use in cancer-care and engineering fault diagnosis [1].

A natural graphical representation of a BN can be found in a directed acyclical graph (DAG) as seen in figure 1. This graphical representations captures the causal relationships as it is evident that a node is only dependent on the subset of nodes in the graph that are ancestors to it. This type of graph can thus be used to model a causal structure.

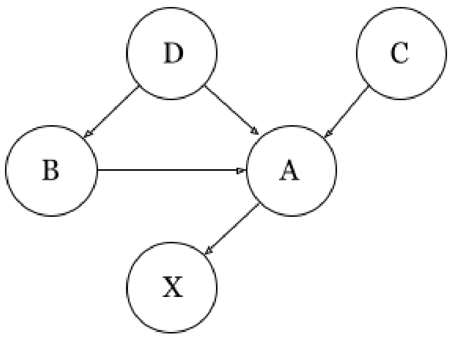


Figure 1: Example of a DAG

2.1.1 Additive Noise Models

In this project we have solely dealt with the subset of causal models/bayesian networks known as linear additive noise models (ANM).

In these types of models, each node is a linear combination of its parent nodes plus a term of added noise, which for example can be a zero-centered Gaussian. The value of A in figure 1 would for example be $A = w_{ba}B + w_{ca}C + w_{da}D + \mathcal{N}(0,1)$, where the weights w_{ij} denote the linear effect that the parent variables have on A and $\mathcal{N}(0,1)$ is a zero-centered Gaussian with variance 1.

Due the models linear and acyclical nature we can always represent these models with an upper triangular adjacency matrix with zero's along the diagonal $W = [w_{ij}]_{i,j=1,...,d} \in \mathbb{R}^{d\times d}$, where d is the number of nodes in the network. The rows $[w_i]_{j=1,...,d}$ gives us the causal order of the network as well as the children of each node, while the columns $[w_j]_{i=1,...,d}$ gives us the parents of each node. The weights are as such encoded in each entry in the matrix.

We note that the structural causal model is $X = W^T X + N$, where $X = [X_1, ..., X_d]^T \in \mathbb{R}^d$ is a vector of iid observations of the graph nodes, $N = [\mathcal{N}_1, ..., \mathcal{N}_d]^t \in \mathbb{R}^d$ is a vector of sampled noise and W^T is the adjacency matrix defined earlier. Using that $(\mathrm{Id} - W^T)$ is invertible (only non-zero entries along the diagonal) we can rearrange the structural casual model into the data-generating equation:

$$X = (\mathrm{Id} - W^T)N \tag{1}$$

This equation allows for quick data-generation, and thus an easy numerical method of finding the marginal variances of nodes in the network, as the analytical approach outlined above, while very informative, quickly grows computationally expensive, as the networks grow (and becomes virtually unviable for networks with $d > 15$).

For reasons that will become apparent later, we are very interested in the marginal variances of the nodes and will therefore quickly outline how these might be found analytically. For each node X_i the marginal variance is given by

$$\mathrm{Var}(X_i) = \sum_{j<i} \left(\sum_{k,l} \left[\prod_{k} w^{\mathrm{path}(k)} \prod_{l} w^{\mathrm{path}(l)} \mathrm{Var}(\mathcal{N}_j) \right] \right) + \mathrm{Var}(\mathcal{N}_i) \tag{2}$$

where the products over $w^{\mathrm{path}(\cdot)}$ denotes the products of weights assigned to the directed edges along a path between node X_j and its descendent X_i. $\mathrm{Var}(\mathcal{N}_j)$ is the variance of the added noise term of node X_j. The variance of node A in figure 1 is for example thus given by

$$\begin{aligned} \mathrm{Var}(A) = {} & w_{ba}^2 \mathrm{Var}(\mathcal{N}_b) + w_{ca}^2 \mathrm{Var}(\mathcal{N}_c) + w_{da}^2 \mathrm{Var}(\mathcal{N}_d) + w_{db}^2 w_{ba}^2 \mathrm{Var}(\mathcal{N}_d) \\ & + w_{db} w_{ba} w_{da} \mathrm{Var}(\mathcal{N}_d) + \mathrm{Var}(\mathcal{N}_a) \end{aligned} \tag{3}$$

2.2 Causal structure learning

Determining the graphical structure of a BN turns out not to be a trivial task. Usually it is either done using expert knowledge in the field of interest or automated discovery of the structure from data (or likely a combination of the two). As the motivation behind this project rests heavily on findings in the automated discovery sphere, we find it helpful quickly outline this field. The methods employed in automated structure discovery usually falls within one of the following four main categories:

1. Constraint based learning
2. Score based learning
3. Gradient-based learning
4. A combination of the methods above

We will here quickly outline the essence of each of these methods.

Constraint based learning methods seek to identify conditional independencies between nodes via statistical tests and then subsequently link nodes, that are not found to be independent. Different graphs can, however, exhibit the same conditional independencies - figure 2 displays such two graphs. We say, that graphs for which this is true, belong to the same Markov Equivalence Class (MEC). When evaluating automated structure discovery algorithms in general, both the ability to reconstruct the ground DAG and the MEC are important and reported. The statistical tests used, test whether two nodes A and B are conditionally independent given some seperationset of nodes $\boldsymbol{S} = \{S_1, ..., S_q\}$, where q is the size of the seperation-set, which can vary from the null set to $d-2$. The tests often start with the null-hypothesis that A and B are conditionally independent given $\boldsymbol{S}$ and then reject this hypothesis for a given p-value (often 0.05). A notable constraint-based algorithm is the *PC* algorithm found in [5].

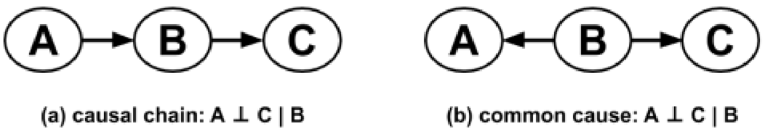


Figure 2: Illustration of two graphs belonging to the same MEC. Illustration is taken from [1]

Score based learning methods seek to find a DAG that optimizes some objective function, which is used to evaluate how well the proposed DAG describes the data. These consist of two elements: 1: The objective function used, and 2: Which search-space does the algorithm go through, and how does it traverse it. The two main categories of objective functions are bayesian scores, which generally focus on the goodness of fit and allows for inclusion of prior knowledge, and information-theoretic, which also consider model complexity to avoid over-fitting. An example of a search space could be all possible DAGs for the given number of nodes, and a traversal method could be iteratively greedily adding edges to it. This method doesn't guarantee finding the optimal graph and is thus an example of what is called an approximate algorithm. The opposite of course exist as well: Exact algorithms guarantees global convergence. A notable score based learning algorithm that also employs greedy choices is the *FGES* algorithm outlined in [2].

Gradient based learning methods are a relatively new addition to the automated structure discovery methods. They take inspiration in the score-based methods: They use an object-function, which they want to optimize. But where score-based methods are under the combinatorial constraint that the found solution must be a DAG (acyclicity constraint), gradient based algorithms introduce a smooth function $h : \mathbb{R}^{d\times d} \rightarrow \mathbb{R}$, which evaluates to 0 when the acyclicity constraint is met. This transforms the learning-problem into an entirely continuous one consisting of a smooth object-function to be optimized under a smooth constraint. In their paper pioneering this method, Zheng et al[6] do this by using use of the augmented Lagrangian method.

2.3 Varsortability

For a variety of reasons, real world datasets for which the DAGs are known, are very scarce. This has led to simulated data being the go-to for evaluation of causal discovery algorithms. This shouldn't pose a problem as long as the simulated data sufficiently mimics real-world data. However, Discoveries by Reisach et al. outlined in their paper [3] indicate that information about the data-generation process might be hidden in the data scale and marginal variances of of the data. I.e. the process by which the data is generated might not lead to datasets adequately mimicking real-world data.

In their paper, Reisach et al. find, that in many standard DAG/ANM generation procedures such as Erdös-Rényi (ER) and Scale-Free (SF) networks, the marginal variances of the nodes increase in causal order. As there is no theoretical backing for why this should be the case in real world data, this poses quite a problem, as these increasing variances can be exploited to find the causal order of a DAG and this performance would not map to real world data.

The paper introduces the concept of *Varsortability* as a measure for the agreement between increasing marginal variance and the causal order. It is defined as the fraction of directed paths that terminate in a node with strictly higher variance compared to the one they started in. I.e.

$$\begin{aligned} v &:= \frac{\sum_{k=1}^{d-1} \sum_{i \to j \in E^k} \text{ increasing } \left(\operatorname{Var}(X_i), \operatorname{Var}(X_j)\right)}{\sum_{k=1}^{d-1} \sum_{i \to j \in E^k}} \in [0,1] \\ &\text{where increasing } (a,b) = \begin{cases} 1 & a < b \\ 1/2 & a = b \\ 0 & a > b \end{cases} \end{aligned} \tag{4}$$

This is the definition we have chosen to work with in this project as well.

Naïvely it seems there is a simple fix for this whole problem: Standardize the data at each note, such that the marginal variance is fixed throughout the causal structure. And this approach is taken in the paper as well. The algorithm *sortnregress*, a simple causal structure learning algorithm exploiting varsortability, is introduced and tested against several state-of-the-art models, both combinatorial and continuous (gradient descent based.) The results of this experiment shows, that the performance of the gradient descent based algorithms mirror the performance of sortnregress. Meaning they all do well on non-standardized data, and fail when the data is standardized. This heavily indicates the gradient descent based algorithms rely on the data-scale and maybe also implicitly on the increasing marginal variance throughout the causal structure.

Reisach et al. suggest that the gradient descent algorithms do actually rely on the varsortability of the data. They find a strong correlation between the variance of the residual vectors $x_j - \boldsymbol{X} w_j$ [1] during the first couple of steps and the marginal variances of the nodes. The magnitude of the gradients of the object-functions (MSE and likelihood) in a given direction is proportional to these residual variances. This means that the addition of edges pointing to nodes with high marginal variances are favored compared to edges pointing away from them. In a graph with high varsortability, this corresponds

[1] Here x_j is dimension j of all n stacked observations of X in $\boldsymbol{X}$. w_j is weight vector for connections into X_j

to finding the causal structure. In this way, gradient descent algorithms may implicitly exploit high varsortability.

In their followup-paper [4], Reisach et al. also demonstrate, that even in completely standardized data you can use a neighbouring sorting criteria to varsortability, R^2-sortability, and achieve similar results to using varsortability. The coefficient of determination R^2 for regression of a node onto other nodes is used as a measure for cause-explained variance fraction

$$\frac{\operatorname{Var}\left(W_{i,t}^{\top} \operatorname{Pa}\left(X_t\right)\right)}{\left(\operatorname{Var}\left(W_{;}t^{\top} \operatorname{Pa}\left(X_t\right)\right)+\operatorname{Var}\left(N_t\right)\right)} \tag{5}$$

which is seen to increase along the causal structure as the marginal variance does. R^2-sortability further demonstrates that just standardizing the data might not be enough to rid the simulated benchmarking datasets of undesirable non-real-life traits.

Our goal in this project is to find methods to minimize the varsortability of linear ANM's without standardizing the data. More specifically, our goal is to lower the varsortability of generated ANMs to .5 (point of random sorting) by only tampering with the edge weights. This, in theory, would allow for testing whether the gradient descent based algorithms are dependent on the data-scale or the marginal variance increase. It will hopefully also shed some light upon how to generate data that more closely mimics real life data.

2.4 Reducing Varsortability - Preliminary Considerations

For most networks given an ER-sampling of weights, the variances are very likely to rise along the causal order (for a proof of why this is likely see B in the Appendix)

The natural question to pose now is simple: How do we avoid this mechanism? To answer that we turn to equations (2) and (3) and a few observations that follow from these:

1. If there is only a single path from X_j to X_i, the contribution to the marginal variance of X_i from X_j is purely positive regardless of the signs of the weights, as all these weights enter in quadrature. Thus, if we keep the mean and variance of the noise term fixed between nodes, the marginal variance can only increase along the causal structure. This is the case for the contribution to the marginal variance that node A receives from both node C and B in figure 1 and equation (3). If a graph only consists of these singular paths it is essentially a collection of chains (This also harmonizes quite well with the intuition we gain from equation (11) in the Appendix).

2. If there is more than one path from X_j to X_i, covariance terms between the different paths will enter the marginal variance calculation of X_i as well. Here the weights

do not enter in quadrature, and it thus possible to get negative terms, possibly lowering the marginal variance along the causal structure. This is the case for the node D's contribution to the marginal variance of A, which can be seen in equation (3)

From the observations above it seems clear, that if we want to lower the varsortability in an ANM changing only the weights, we have to exploit the covariance-terms of multiple paths travelling from one node to another, by ensuring that these give negative contributions to the marginal variance of the nodes they are terminating in. We give a short demonstration of how this might be done before we move on.

Consider the ANM below. For simplicity we let $\mathrm{Var}(\mathcal{N}_a) = \mathrm{Var}(\mathcal{N}_b) = \mathrm{Var}(\mathcal{N}_c) = 1$. The initial variances thus become $\mathrm{Var}(A) = 1$, $\mathrm{Var}(B) = 10$ and $\mathrm{Var}(C) = 15$ It is obvious that the varsortability is 1. After applying the simple augmentation suggested however, we get that $\mathrm{Var}(A) = 1$, $\mathrm{Var}(B) = 10$ and $\mathrm{Var}(C) = 9$, which brings our varsortability down to 3/4, as the path between node B and C would now be incorrectly sorted.

$$\begin{tikzcd}
 & A \arrow[dl, "3"'] \arrow[dr, "1"] & {} \arrow[r, "\text{Augmentation}"] & {} & A \arrow[dl, "-3"'] \arrow[dr, "1"] & \\
B \arrow[rr, "1"'] & & C & B \arrow[rr, "1"'] & & C
\end{tikzcd}$$

3 Methods & Results

During the course of this project we have implemented three main methods of minimizing the varsortability of ANMs. These include a *Genetic gRaph sElection alGorithm (GREG)*, a *Weight gradiEnt dEscent mEthod (WEEE!)* and a more transparent method that seeks to minimize local variance: *LOcal variance minimizer using negative COvariance* (LoCo). All these methods will be described and discussed in further detail below in their own sections.

3.1 General Simulation Details

All of our algorithms were tested on the same subset of ANMs. We implemented our own ANM class and sampled networks in an Erdös-Renyi(ER)-inspired manner: We sample a graph of d where each node has probability (connectivity) p of connecting to a node further down the causal structure with a directed edge with edge values sampled iid $w_{ij} \sim \mathrm{Unif}((-2, -.5) \cup (.5, 2))$. The noise terms are all sampled from Gaussians with standard deviations sampled as $\sigma_i \sim \mathrm{Unif}(0.5, 2)$. Unless otherwise noted, all calculations of variances above $d = 5$ are done using 1000 samples of data using eq. (1). For ease of comparability, all stated algorithm run times are on Intel Xeon 2.20 GHz CPU from the free cloud computing service Kaggle. Finally, we note that we call these networks 'ER-inspired' as opposed to just ER-networks, as the number of root nodes in our networks have all been set to 1 instead of being a function of the number of nodes and connectivity of the network. This is an unfortunate oversight on our behalves that we noticed too late in the process to be able to change.

3.2 Genetic gRaph sElection alGorithm (GREG)

3.2.1 The Algorithm

GREG was the first algorithm we implemented, as we examples of low-varsortability ANMs which we could use to identify important features of low varsortability ANMs.

The genetic algorithm seeks to generate solutions to optimization and search problems by emulating the Darwinian idea of natural selection: Survival of the fittest. Our implementation takes an ANM input and creates n variations of it by slightly altering the edge weights. These variations are called the population. The population is then sorted by a culling-metric (in our case, the varsortability as described in (4)) and the 50% best performing ANM's are allowed to live on to the next generation. At the next step, variations on the surviving population of step 1 are injected into the population and a culling commences once again. This process continues for a set number of generations or until a set value of the culling metric is reached.

For a more detailed overview of the implementation of the algorithm, including Python-esque pseudocode and important considerations, see section D.1 in the Appendix.

3.2.2 Results

Running GREG, we quickly found, that given enough generations and a big enough population-size, the varsortability of most ANMs can be reduced to very low values - Often well below the threshold of randomness of Varsort = .5. See figure 11 in section A.1 of the appendix for a plot detailing this further. Furthermore refer to figure 12 in the appendix for baseline varsortabilities of un-altered ANMs.

After we observed that GREG dramatically lowered the varsortability of the ANM's we fed it, the natural next question to pose was simple: What does it actually do? For small networks we found that we could follow a few basic tendencies: 1) It generally made sure that the first couple of nodes in the causal structure had very high marginal variances, presumably so it would be easier for it to sort the following nodes wrongly. And 2) It made sure to exploit the negative covariance terms (described in chapter 2.4) by placing strategic negative weights along paths with a large path-products (e.g. $w_{ab}w_{bc}...$ is large). Unfortunately, as the networks grow in size, it becomes increasingly difficult to map out these paths analytically. We therefore turn to a more holistic analysis of the resulting graphs. Figure 3 displays the marginal variances of 3 different ANMs with $d = 15$ and $p = .8$ after they have been fed through GREG. Observing this figure we see a familiar trend, namely that the algorithm seems to maximize the marginal variances of the first third of the nodes, whereafter it minimizes most nodes while still leaving some deliberate high-variance-nodes. Refer to figure 16 in section A.1 of the appendix for a figure displaying the baseline results for un-altered ANMs.

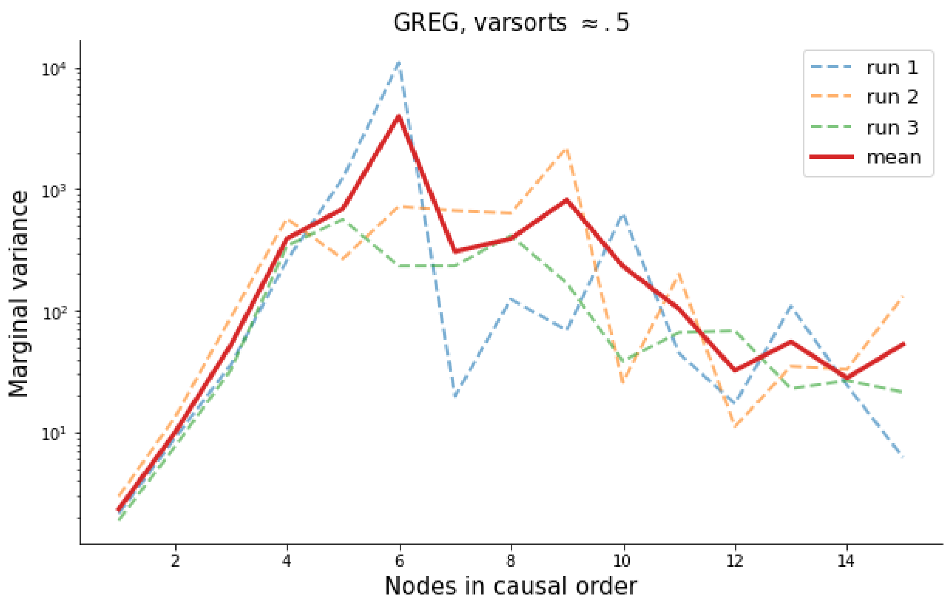


Figure 3: Plot displaying the marginal variances of the of 3 different ANMs with $d = 15$ and $p = .8$ after they have been fed through GREG. Notice: GREG maximizes upper third of nodes.

Besides figure 3 we also utilize a number of descriptors to analyze the change in the network. The results of these descriptors can be found in table 1, which displays average value and standard deviations of these descriptors attained by GREG on 10 sampled ANMs of $d = 10$ and $p = .8$. [2]

- **Relative L2 norm (L2)**: The L2 norm given in units of the expected value of the L2 norm of an ANM of that size and connectivity. Ie. for a value of 1 we do not see any change from randomly sampled ER-networks.
- **Upper relative L2 norm (U. L2)**: The L2 norm of the weights of the upper half of the adjacency matrix. Also given in units of the expected value.
- **Lower relative L2 norm (L. L2)**: As above, but the lower half of the adjacency matrix.
- **Fraction of weights being negative (Neg. w's)**: The fraction of all weights in the adjacency matrix being negative. The expected value is for a randomly generated ANM is of course .5.

[2]We note that this choice of d and p might seem very arbitrary, but we couldn't do an entire grid-search as that would be way too computationally costly. We chose these values, as in our prior experience we seem to be able to generalize well from these.

- **Upper fraction of weights being negative (U. neg. w's)**: Same as above, but only upper half of adjacency matrix is considered.
- **Lower fraction of weights being negative (L. neg. w's)**: Same as above, but only lower half of adjacency matrix is considered.
- **Variance of weight distribution (Var)**: Variance of the weight distribution given in units of the expected variance.

Observing table 1 we notice a couple of interesting features. For one, the L2 norm of the entire weight vector doesn't change, the value of L. L2 however do change. More specifically we see an indication of the magnitudes of the weights in the lower half of the adjacency matrix being lowered. This makes could make sense in the context of figure 3: The algorithm possibly uses the relatively larger weight magnitudes to maximize the marginal variance in the upper end of the causal structure.
The fraction of weights being negative are very much within the expected region (about 0.5). We included these measures mainly to check whether the maximization of the marginal variances in the upper causal chain was done by elimination of negative weights in this region, as this would generate networks in which the causal structure was identifiable by amount of negative weights which would be an undesirable side effect. Fortunately, this does not seem to be the case. Lastly, we don't see a significant change in variance. This measure was included as we had expected the algorithm to 'extremify' edgeweights in order to min/max high and low marginal variances. Curiously this does not seem to be the case.

GREG	Upper	Lower	Total
L2	1.02 ± 0.09	0.6 ± 0.2	1.0 ± 0.11
Frac. neg. weights	0.46 ± 0.09	0.44 ± 0.08	0.5 ± 0.2
Var			1.1 ± 0.2

Table 1: Network descriptors of results attained by GREG on 10 ANMs of $d = 10$ and $p = .8$

3.3 Weight gradiEnt dEscent mEthod (WEEE!)

The loss function: 'Continuous' Varsortability

We first bring attention to the fact, that the way we calculate marginal variances, and thus varsortability, is through simulation of data. This means that any loss function we implement is essentially a black-box algorithm and we thus do not have a closed form solution to its derivative. We therefore employ a finite difference method to approximate the gradients.

In order to implement any kind of a gradient descent algorithm to minimize the varsortability, we first had to redefine the varsortability 'loss function' (eq. 4), as it is essentially a series of step-functions, and this did now allow for good optimization. If the algorithm took a step in a direction where no new nodes were wrongly varsorted, the gradient would be 0 and the optimization would thus halt there. Our first approach to solve this problem was to use the marginal variance values of the nodes in order to make a continuous stand-in function for eq. 4:

$$\sum_{i=1}^{d}\sum_{j>i}^{d} \mathrm{Var}(X_i) - \mathrm{Var}(X_j) \tag{6}$$

This function initially seemed like the obvious candidate as it would decrease if the varsortability decreased. It can however also be minimized by simply lowering the edgeweights (and therefore the global variance), which was exactly what we observed to happen. We instead tried to implement

$$\sum_{i=1}^{d}\sum_{j>i}^{d} \frac{\mathrm{Var}(X_i) - \mathrm{Var}(X_j)}{\mathrm{Var}(X_i) + \mathrm{Var}(X_j)} \tag{7}$$

as the denominator here would naturally penalize a global lowering of edge weights. However, the use of this loss function often resulted in the algorithm maximizing the discrepancy between a single pair of nodes, which also did not decrease the varsortability much. After much trial and error we finally ended up with the following loss function, which deserves an explanation of its own.

$$\begin{align} v_{loss} &= \sum_{i=1}^{d}\sum_{j<i}^{d} \Theta(\mathrm{Var}(X_j) - \mathrm{Var}(X_i)) + \frac{\mathrm{Var}(X_i) - \mathrm{Var}(X_j)}{(\mathrm{Var}(X_i) + \mathrm{Var}(X_j))^{1.1}} \tag{8} \\ &= N_{\text{wrongly ordered}} + (9) \tag{9} \end{align}$$

Where $\Theta(\mathrm{Var}(X_j) - \mathrm{Var}(X_i))$ is a a Heaviside step function, which essentially counts how many pairs are wrongly sorted. The exponent 1.1 appearing in the denominater of 7 is there, as we heuristically found, that we had to penalize global edge weight lowering a bit more. We bring attention to the fact that the loss function is no longer continuous, however, this turns out not to be a big issue.

Figure 4 displays v_{loss} vs the marginal variance of the child node in 4 different child-parent node pairs. By observing this figure we see, that even if the algorithm after a step does not sort a note wrongly according to varsortability, v_{loss} still slopes towards the next tipping point where this will happen. This ensures that the optimization process will not halt as it would have done if one had used eq. (4) naïvely. Furthermore, the sharp drops in the function that happen when a pair is wrongly sorted according to the varsortability ensure, that the optimal next step to take, is the one that will do exactly this. Finally, by observing the child-parent pair where Var(parent) ≈ 0, we see that lowering both the parent and the child nodes marginal variances (global lowering) does not necessarily lead to a low v_{loss}. This function thus fixes the issues we have presented thus far, and it is therefore the one we chose to use.

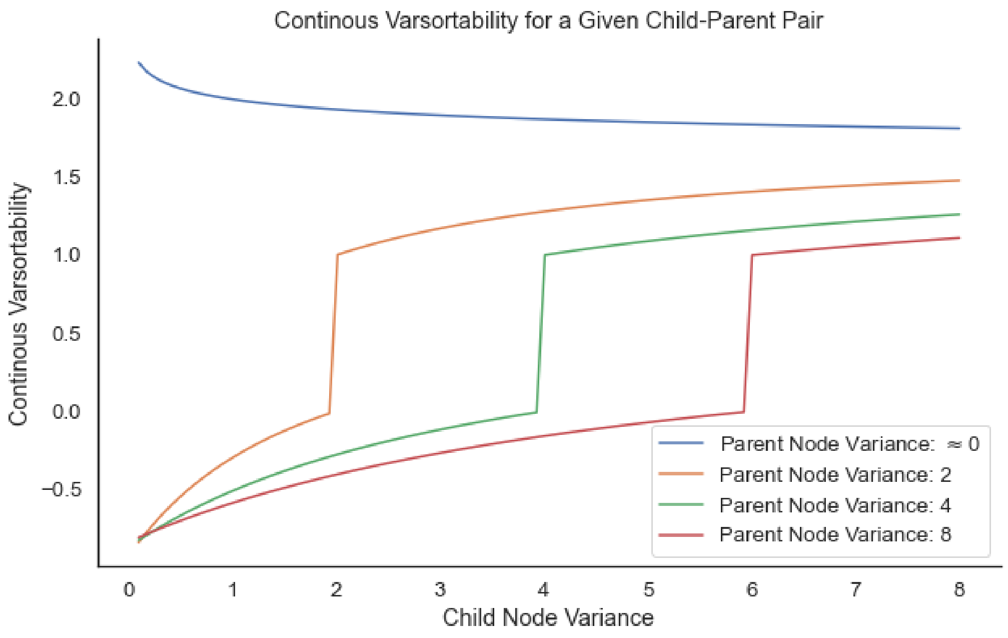


Figure 4: Plot of v_{loss} vs the marginal variance of the child node for 4 different child-parent pairs. There are three main features to look for (1) Greatest improvement in loss is to ruin correct sorting. (2) Even if no nodes are incorrectly sorted, the loss still slopes towards the next tipping point. (3) The algorithms are punished for simply pushing both variances towards 0

3.3.1 The Algorithm

The implementation of the gradient descent is relatively typical apart from the fact that the loss function required live simulation of data and was thus non-deterministic given an input. A Python-esque pseudo-code can be seen as algorithm 3.

For a more detailed overview of the implementation of the algorithm, including optimizations and important considerations, see section D.2 in the Appendix.

3.3.2 Results

The performance results of WEEE! much mirrors that GREG: Given enough epochs and a fitting learning rate, the varsortability of most networks can be reduced well below 0.5. Figure 13 in appendix displays some of these results.

As was also the case with GREG, once the network grows beyond about $d = 5$ or so it is very difficult to gauge what is going on, so we turn to same plot and descriptors described before in section 3.2.2. Figure 5 displays the the marginal variances of the of 3 different ANMs with $d = 15$ and $p = .8$ after they have been fed through WEEE! This figure shows results akin to those of 3, but with a few key differences: 1) The marginal variances are globally a lot smaller. 2) While the it is still true that the upper third of the causal chain have the largest variances, this feature is not as distinct as in figure 3. 3) The trend 'wobbles' a lot more.

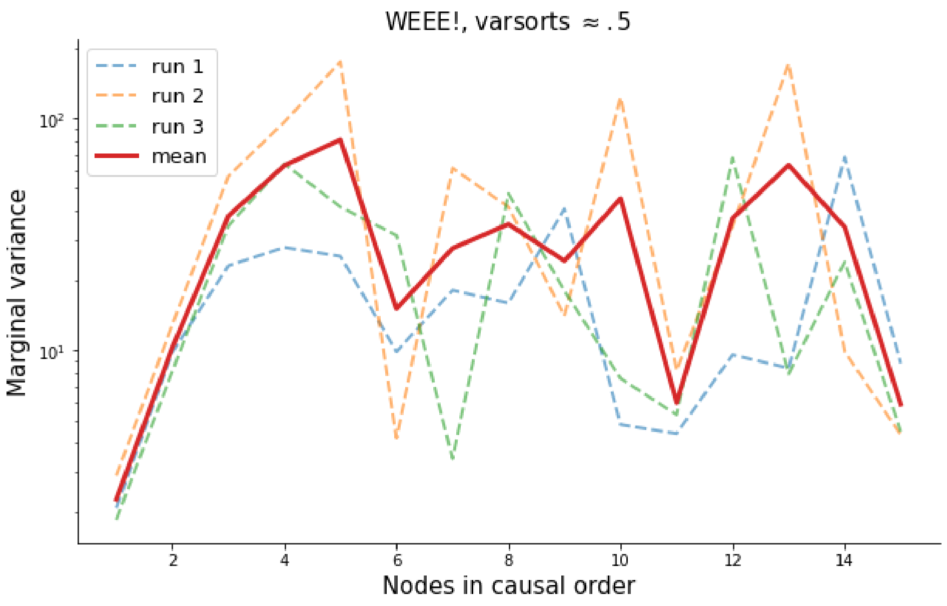


Figure 5: Plot displaying the marginal variances of the of 3 different ANMs with $d = 15$ and $p = .8$ after they have been fed through WEEE! Notice: WEEE! also maximizes upper third of nodes, although more weakly than GREG.

No matter our choice of hyperparameters, WEEE! seemed excruciatingly slow. Figure 6 displays the time it takes for both GREG and WEEE! to get a varsortability of .5 as a function of the number of nodes in the networks as well as connectivity. The figure makes it very clear that even though the performance of GREG and WEEE! are comparable (as can be seen by observing figure 13 in the appendix), GREG clearly wins out in terms of speed.

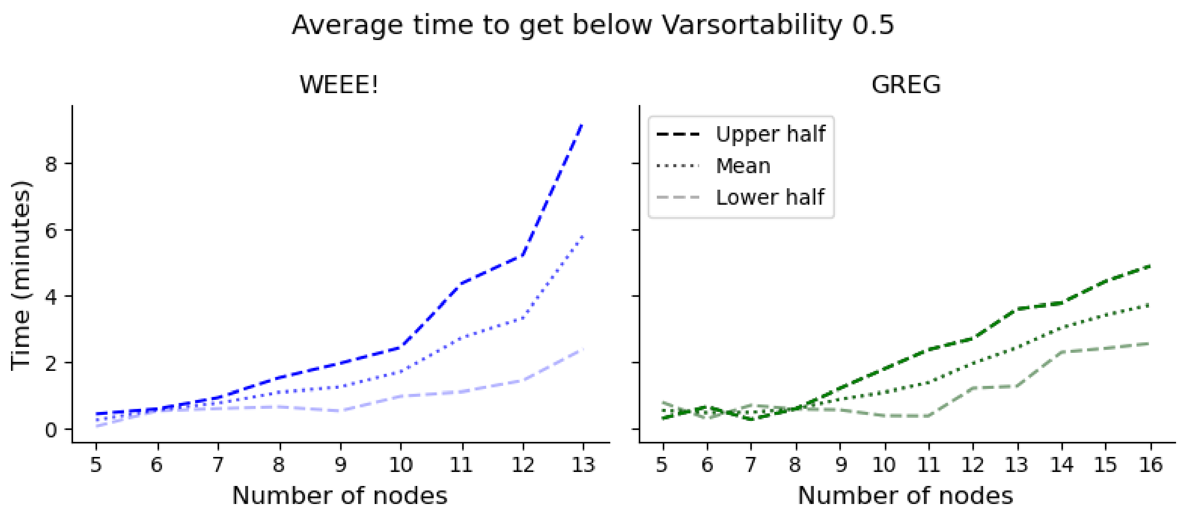


Figure 6: Plots display convergence times for (time to arrive at varsortability 0.5) as a function of connectivity and network size. The results and error bands displayed are means across the upper, lower, and all connectivities respectively.

Table 2 displays the means of the descriptors for an ANM of $d = 10$ and $p = .8$ fed through WEEE! ten times.

WEEE!	Upper	Lower	Total
L2	0.88 ± 0.05	0.6 ± 0.2	0.81 ± 0.06
Frac. neg. weights	0.55 ± 0.10	0.7 ± 0.2	0.6 ± 0.10
Var			0.7 ± 0.11

Table 2: Network descriptors of results attained by WEEE! on 10 ANMs of $d = 10$ and $p = .8$

Observing the table we see that L2 decreases for WEEE! This is not a surprise, as although we did a lot to hinder this behaviour via the tweaking of the loss function, there is apparently still a small incentive for the algorithm to globally lower its weights. The trend to have smaller magnitude weights in the lower part of the causal structure persists. The fraction of weights being negative are 'normal' within a single standard deviation, and finally we believe the change in variance can be mostly explained by the overall change in the L2 norm.

3.4 Local variance minimizer using negative Covariance (LoCo)

While GREG and WEEE! both do very well in minimizing the overall varsortability for ANMs they have some issues. First and foremost they are both essentially blackbox algorithms, and the pattern in the way they change the ANMs is therefore not immediately obvious. Secondly, they are both slow algorithms, especially for larger networks.

The combination of these two issues make it unfeasible that these algorithms will be used in the field. The algorithms did however grant us knowledge about the possible configurations of ANMs with low varsortability, which we used to design our final algorithm.

We wanted to create a transparent and computationally cheap algorithm. Inspired by the trend of figures 3 and 5, namely that GREG and WEEE! both seem to maximize the marginal variances of a good portion of the upper causal structure.

3.4.1 The Algorithm(s)

We implemented a variety of versions of the LoCo algorithm. We here describe the two versions that performed the best: The *Max-top-third* variant, and the *Genghis* variant.

Max-top-third LoCo works by iterating down the causal structure, it changes the individual nodes marginal variance by simply flipping the sign on the incoming edges. For the upper third (rounded up) of the causal structure, signs are chosen such that the local marginal variance of the node is maximized. For the lower 2 thirds of the causal structure, signs are chosen such that the variance is minimized. Essentially the algorithm maximizes or minimizes the covariance terms of (2). The fraction of nodes in the upper causal structure to be maximized was numerically found to be around a third as can be seen in figure 17.

Genghis LoCo Works in much the same way, only it doesn't rely on causal structure position in order to gauge whether to var-minimize or var-maximize a node. Instead it maximizes *Genghis nodes*: Nodes that have more children than parents. This naturally works as kind of a 'stand-in' for var-maximizing the upper causal structure, as nodes higher in the structure naturally have fewer parents and more children. By maximizing these nodes we amplify the chances for many paths to start in a nodes whose variance has been maximized and terminate in a node whose variance has been minimized.

The algorithm(s) doesn't rely on any global metric, but only on the local variances and the fact that LoCo goes down the causal structure allows it to run without any repeated loops over the network, meaning that the algorithm is consistent in time of convergence, in opposition to GREG and WEEE!

We also implemented a `flip_up_to` parameter, which allows us to control how many signs the LoCo algorithsm are allowed to flip per node. Reducing this from `All` makes the combinatorial task of trying all flips lighter.

For a pseudo-code implementation of the max-top-third LoCo variant can be found in section D.3 in the Appendix.

Other variants We succinctly introduce a few other LoCo-variants here for later comparison purposes.

- **Update-by-Varsort LoCo** Updates the network by minimizing the global Varsortability.
- **Minimize-all LoCo** Simply minimizes the marginal variance of every node.

3.4.2 Results

Figure 7 displays the varsortability achieved by the max-top-third variant of LoCo on ANMs of different node-count and connectivities. These results are for `flip_up_to = All`. The results for the Genghis variant closely resembles this and can be found in figure 14 in the appendix.

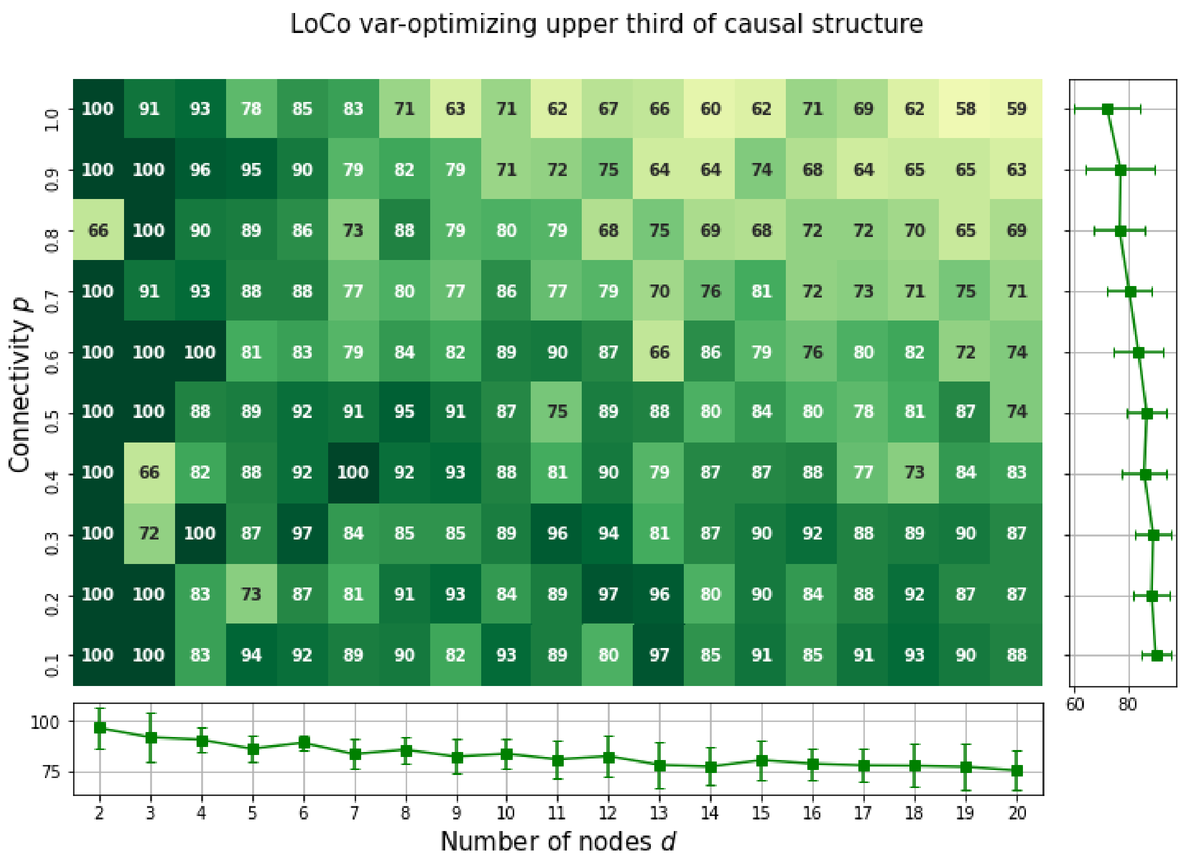


Figure 7: Figures of the varsortabilities achieved by max-top-third-LoCo on ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively.

The figure above clearly shows, that as d and p increases, so does the algorithms capability of reducing varsortability. We believe that is behaviour will generalize to higher d's and p's, as we see no reason as to why it wouldn't, but unfortunately we did not have the time or computing power to test this.

Figure 8 displays the performance for the different LoCo variants for different graph sizes and averaged over connectivities $[0.5, ..., 1]$. Note that the Genghis and Maximize-upper-third variants of LoCo do quite a bit better than the Minimize Varsortability version, thus showing that greedily optimizing after the varsortability actually does worse than locally optimizing after marginal variance. For more details on the results achieved by the Minimize-Varsortability and Minimize-All variants of LoCo refer to figures 19 and 18 in the appendix, respectively.

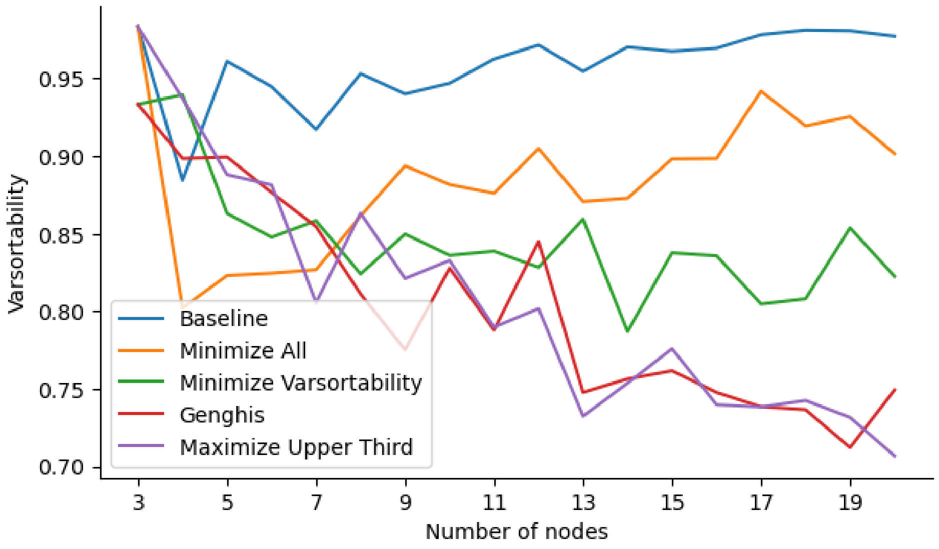


Figure 8: A simple varsortability-plot for different implementations of the LoCo Algorithm. The is a mean three runs each of connectivities [.5,.6,.7,.8,.9, 1]. While the base algorithm stays the same (iterating through the network in the causal order), the different variations performs drastically different!

Figure 9 displays time and performance of Genghis-Loco for different values of `flip_up_to`. Notice that while the perfomances seem to be comparable, the execution time varies by up to almost 3 orders of magnitude! The time difference between flipping 3 signs pr. node and flipping all signs for an ANM of $d = 20$ and $p = .8$ is 34 minutes! For more analysis on the time-complexity refer to section A.2 in the appendix.

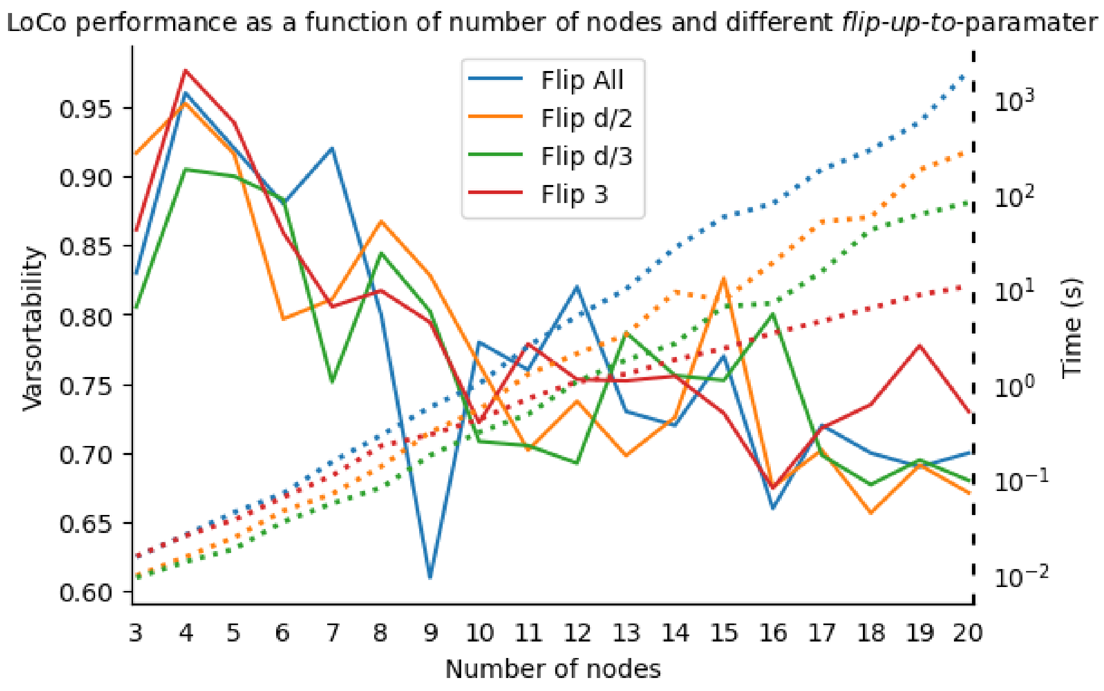


Figure 9: In dotted lines we have the average time for the LoCo algorithm to terminate. In filled lines the achieved varsortability. It should be noted that while performance is comparable, the completion time spans almost three orders of magnitude. Tested on ANMs with connectivity 0.8.

Figure 10 displays the marginal variances for nodes in causal order for three ANMs with $d = 20$ and $p = .8$ that have been fed through max-top-third-LoCo. Observing the figure we see that it shares the sought after characteristics of figures 3 and 5. We also note that the marginal variances of nodes 1 through 8 have been maximized in these ANM's which shines through, as it is around this point that the marginal variances start to decline.

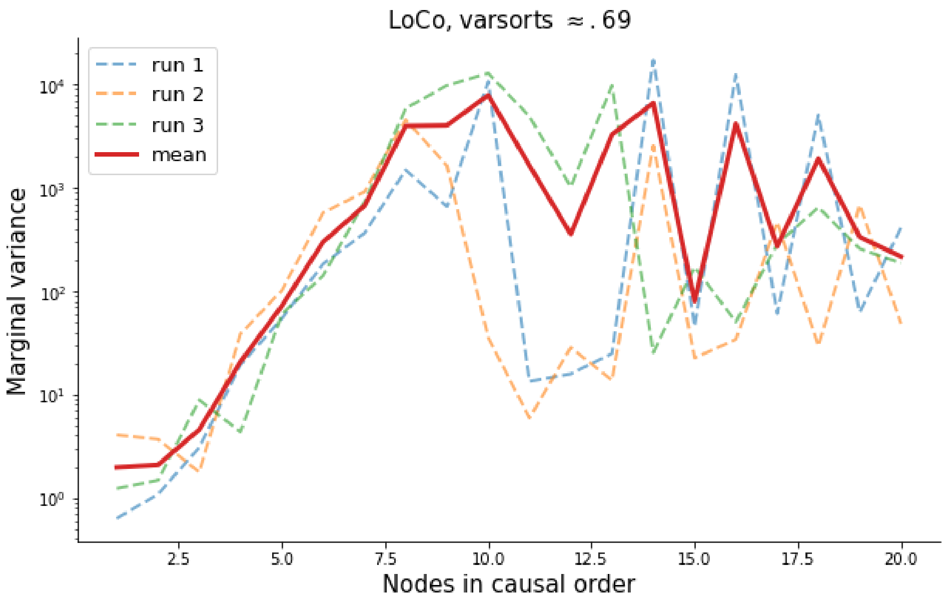


Figure 10: Displays the marginal variances for nodes in causal order for three ANMs with $d = 20$ and $p = .8$.

Finally, table 3 shows the network descriptors described first in section 3.2.2. The L2 norms have been omitted, as these are all trivially unchanged. By observing the table we see that although fraction of negative weights doesn't change much, it seems more consistent in its output. Given more data, one might be able to find a statistically significant change in the results, but as it stands we are unfortunately unable to conclude whether LoCo-modified networks are detectably different from randomly sampled ER-networks.

	Neg. w's	U. neg. w's	L. neg. w's
LoCo	0.51 ± 0.06	0.51 ± 0.06	0.5 ± 0.13

Table 3: Network descriptors of results attained by LoCo on 50 ANMs of $d = 15$ and $p = .8$

4 Discussion

In general we have found, that given enough compute-hours and a high enough connectivity, it is possible to generate an ANM of arbitrary varsortability for most graph structures. This is given by the results of GREG and WEEE! However when utilizing these two algorithms we typically see a lowering of the magnitude of the weights in the lower part of the causal order. This in itself probably does make these networks distinguishable from proceduraly generated ANMs and would probably also make them unfit to mimic real-life data. In addition to this, we can't guarantee that the networks aren't distinguishable by an entirely other metric, which we have not thought about. Finally, these algorithms are blackboxes and are therefore not very appropriate to use, if one wishes to exert control over how exactly the networks are changed.

One of the more interesting findings we had, was the fact that simply utilizing sign-flips to change the local variance can result in a global reduction in varsortability! However, were we to continue the project the very first thing to focus on would be to apply the LoCo algorithm to networks of greater size to see we can find the convergence on varsortabilities below 0.5 we have seen evidence of. It was also unfortunately pretty late in the process that we discovered that we could achieve comparable results by allowing only for the sign-flipping of a fraction of the incoming edge weights and thereby lowering the execution time immensely. If the project went on we would have very much liked to study this aspect. I.e. how few flips can we go down to and still maintain good performance? Continuing down this train of thought, many optimizations of the code could be made. Some smart caching would for example lower the burden of calculating variances for the nodes a lot. We also believe that if we where to sacrifice some of the algorithm's simplicity and non-invasivity and allow for more varied meddling, we could achieve greater performance. This idea can be supported by the fact that we see an additional rise in variances in the upper half of the causal order for the genetic and gradient algorithm (see figures 3 & 5). We believe that we could utilize this in LoCo by, in some way, allowing it to change the magnitude of the edge weights. Finally, we note that as LoCo doesn't rely on any global metric of the networks, it is theoretically possible to utilize this algorithm under the sampling of the networks themselves.

Another interesting finding that we have, is that over the course of project we found some persistent anatomy in the ANMs exhibiting low varsortability. This includes raising the variances on the first third of the causal order while lowering the rest except for leaving strategically placed variance-"spikes" in the graph.

Finally, we acknowledge that all of our algorithms only have been tested on a small subset of ER networks and not on Scale Free (SF) networks at all. Also, due to time restraints, most of our experiments haven't been repeated to the point of statistical significance. The results of our project are therefore to be questioned and tested further before conclusions are drawn from them.

5 Conclusion

In this project we have implemented three algorithms that minimize varsortability for a given sampled ANM: GREG, WEEE! and LoCo. THe former two gave us important information about the nature of low varsortability ANMs, and demonstrated that given enough time and a high enough connectivity, the varsortability of most ANMs can become almost arbitrarily low. The algorithms are however, very slow and opaque. The latter (LoCo) seems to be the most interesting of the bunch by far. It is computationally cheap, transparent, not very invasive, and it is very surprising that only flipping the signs of the edge weights can achieve results on this scale. We believe that simple algorithms like LoCo could be interesting to the field and that the study of these could lead to further understanding of the internal structures of simulated ANMs, especially with regards to avoiding non-real-life like traits like high varsortability.

References

[1] Neville Kenneth Kitson et al. “A survey of Bayesian Network structure learning”. In: *CoRR* abs/2109.11415 (2021). arXiv: `2109.11415`. URL: `https://arxiv.org/abs/2109.11415`.

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

[3] Alexander G. Reisach, Christof Seiler, and Sebastian Weichwald. *Beware of the Simulated DAG! Causal Discovery Benchmarks May Be Easy To Game*. 2021. arXiv: `2102.13647 [stat.ML]`.

[4] Alexander G. Reisach et al. *Simple Sorting Criteria Help Find the Causal Order in Additive Noise Models*. 2023. arXiv: `2303.18211 [stat.ML]`.

[5] Peter Spirtes and Clark Glymour. “An Algorithm for Fast Recovery of Sparse Causal Graphs”. In: *Social Science Computer Review - SOC SCI COMPUT REV* 9 (Apr. 1991), pp. 62–72. DOI: `10.1177/089443939100900106`.

[6] Xun Zheng et al. *DAGs with NO TEARS: Continuous Optimization for Structure Learning*. 2018. arXiv: `1803.01422 [stat.ML]`.

A Appendix

A.1 I Like Big Plots and I Cannot Lie

(This is a reference to Sir Mix-a-Plot's 1992 hit single *Baby Got DAG*)

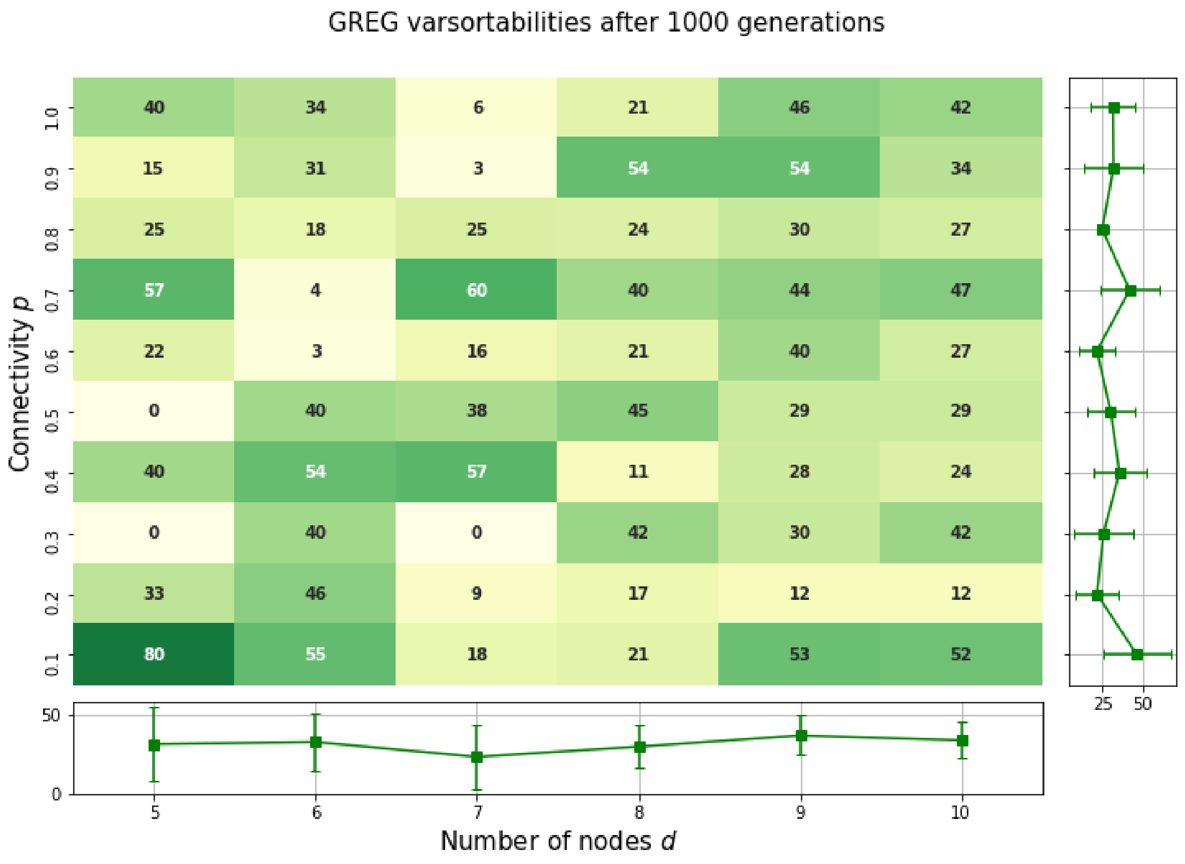


Figure 11: Figures of the varsortabilities achieved by GREG on ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively. We note that we restricted this plot to ANMs of node count $d \in [5, 10]$. This range wasn't expanded primarily due to lack of time, as running the algorithm can become very computationally expensive for larger networks.For this particular run we did not implement an early-stopping feature at .5 varsortability (The point of randomly sorted nodes in regards to marginal variance) as we wanted to see just how low we could get it.

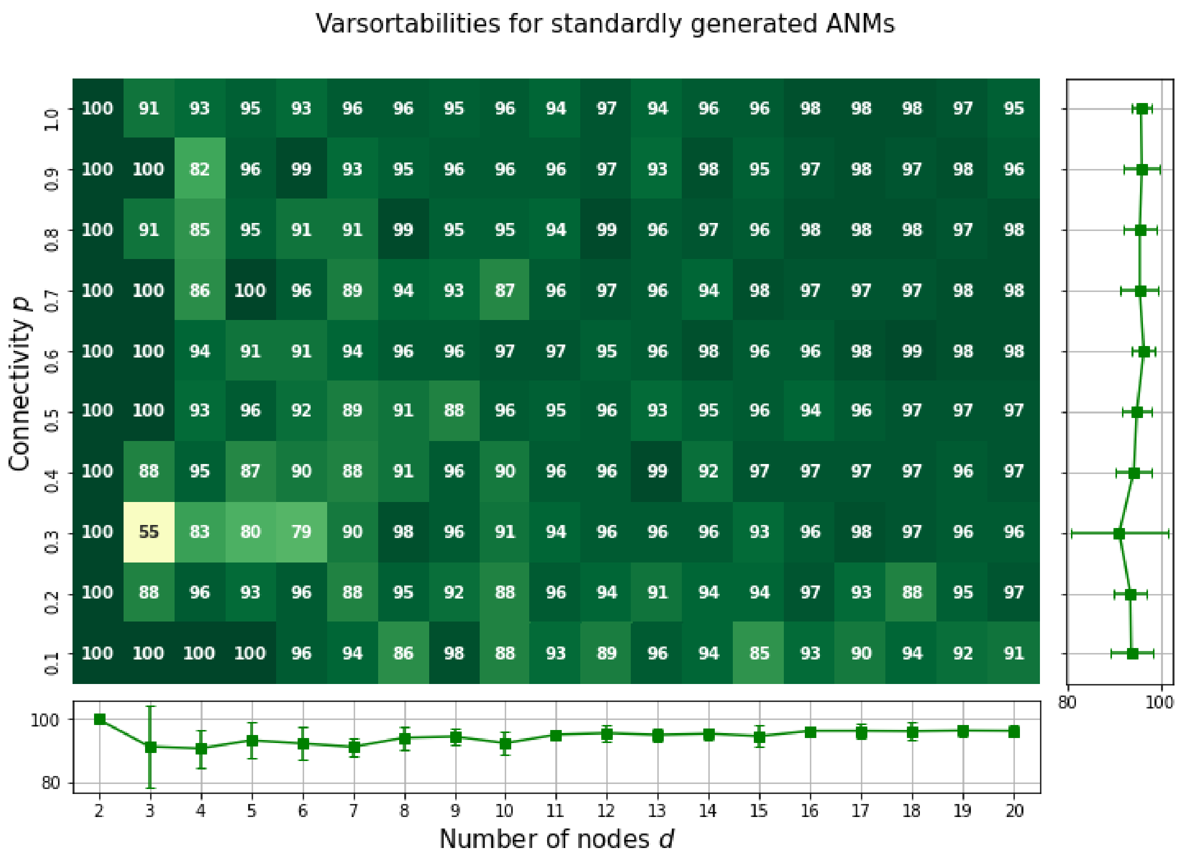


Figure 12: Figures of the varsortabilities of unaltered ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively

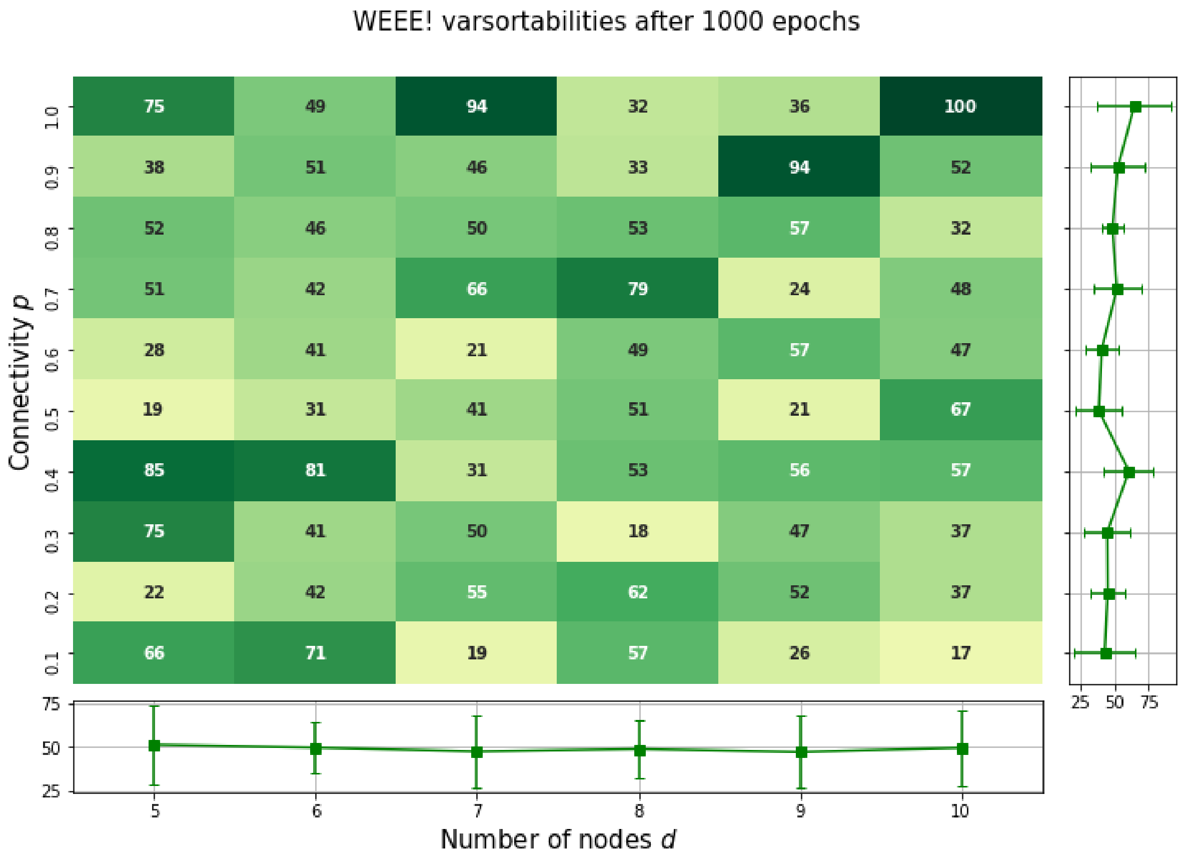


Figure 13: Figures of the varsortabilities achieved by WEEE! on ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively. As it was with GREG, the range of nodes is low due to the slow converging time of the algorithm. Notice: The varsortability reported for $d = 10$ and $p = 1$ is 1, this is an example of the algorithm not converging - most likely due to a lack of epochs.

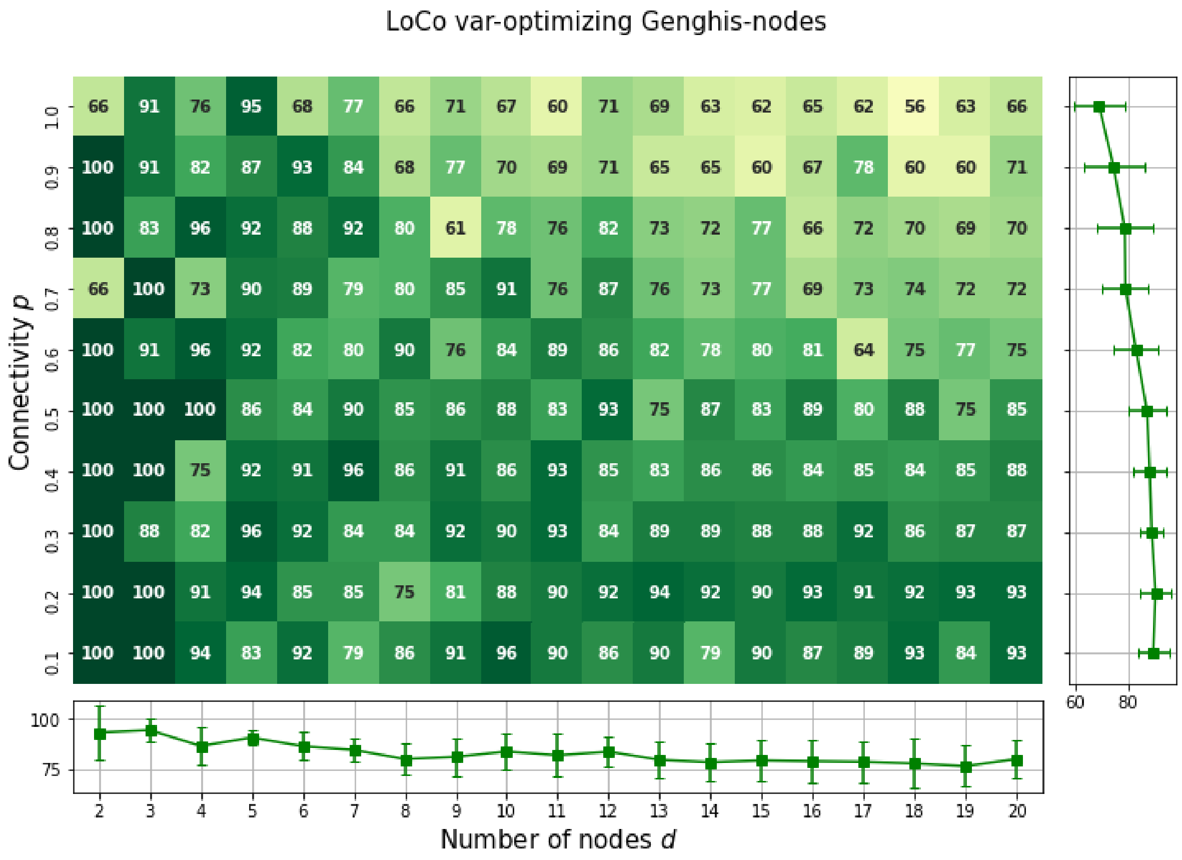


Figure 14: Figures of the varsortabilities achieved by Genghis-LoCo on ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively. Notice: The results displayed here closely resembles that of the max-top-third-LoCo.

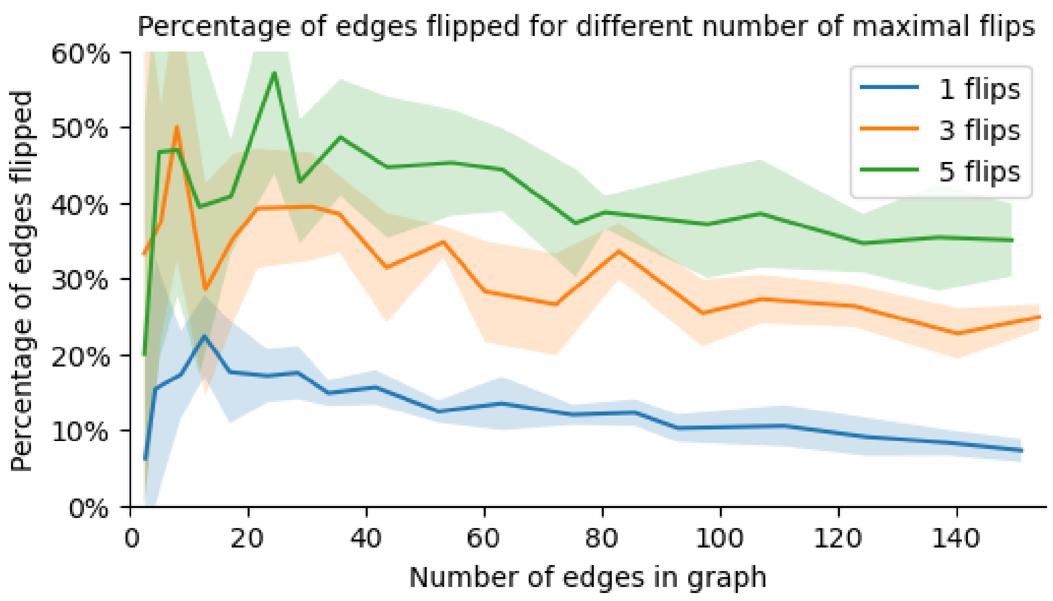


Figure 15: The actual percentage of edges experiencing a sign flip after running LoCo with textitflip-up-to-parameter set to different values.

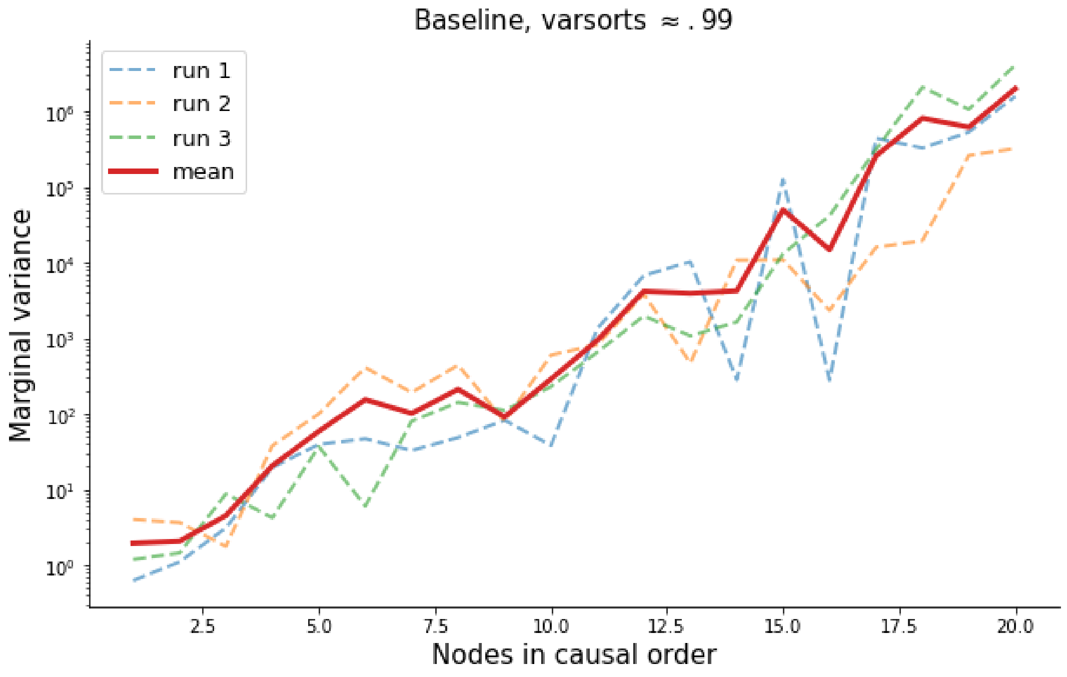


Figure 16: Plot displaying the marginal variances of 3 different unaltered ANMs with $d = 20$ and $p = .8$. Notice: The marginal variance trends upwards $\rightarrow$ High varsortability!

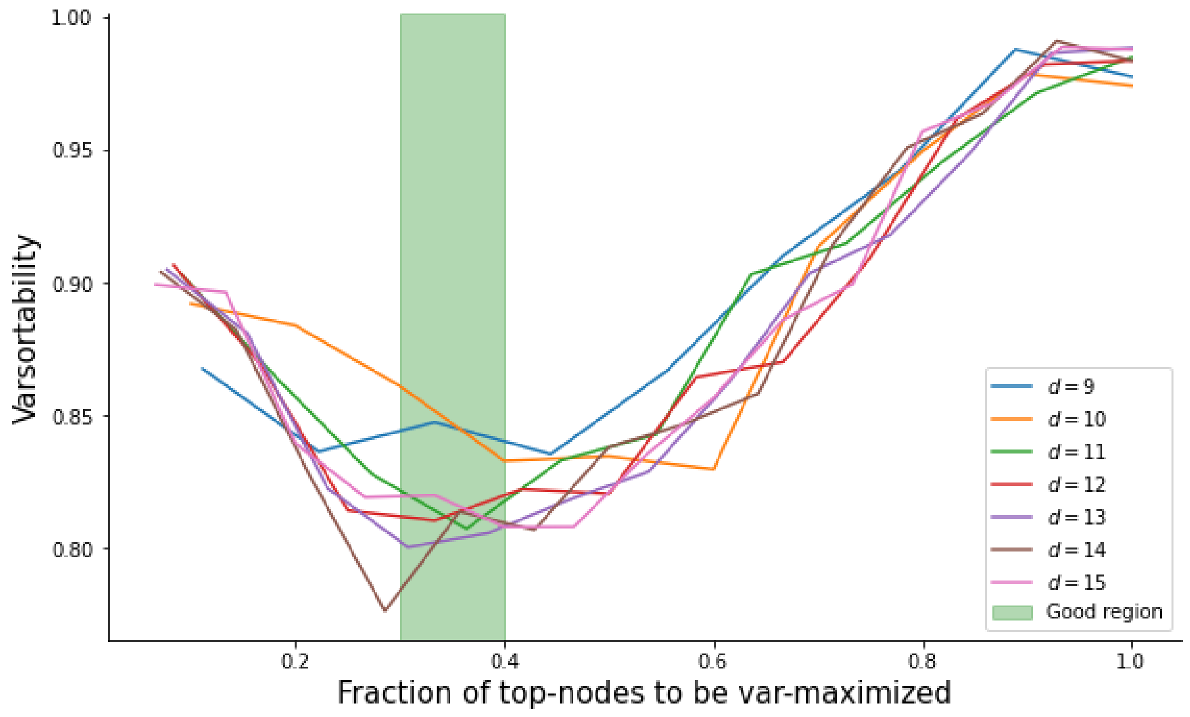


Figure 17: The results of a numerical search over multiple dag sizes for the best percentages of nodes to be maximized. Each size was run three times all with $p = .8$. Notice: The fraction seems to converge towards 1/3 for larger networks

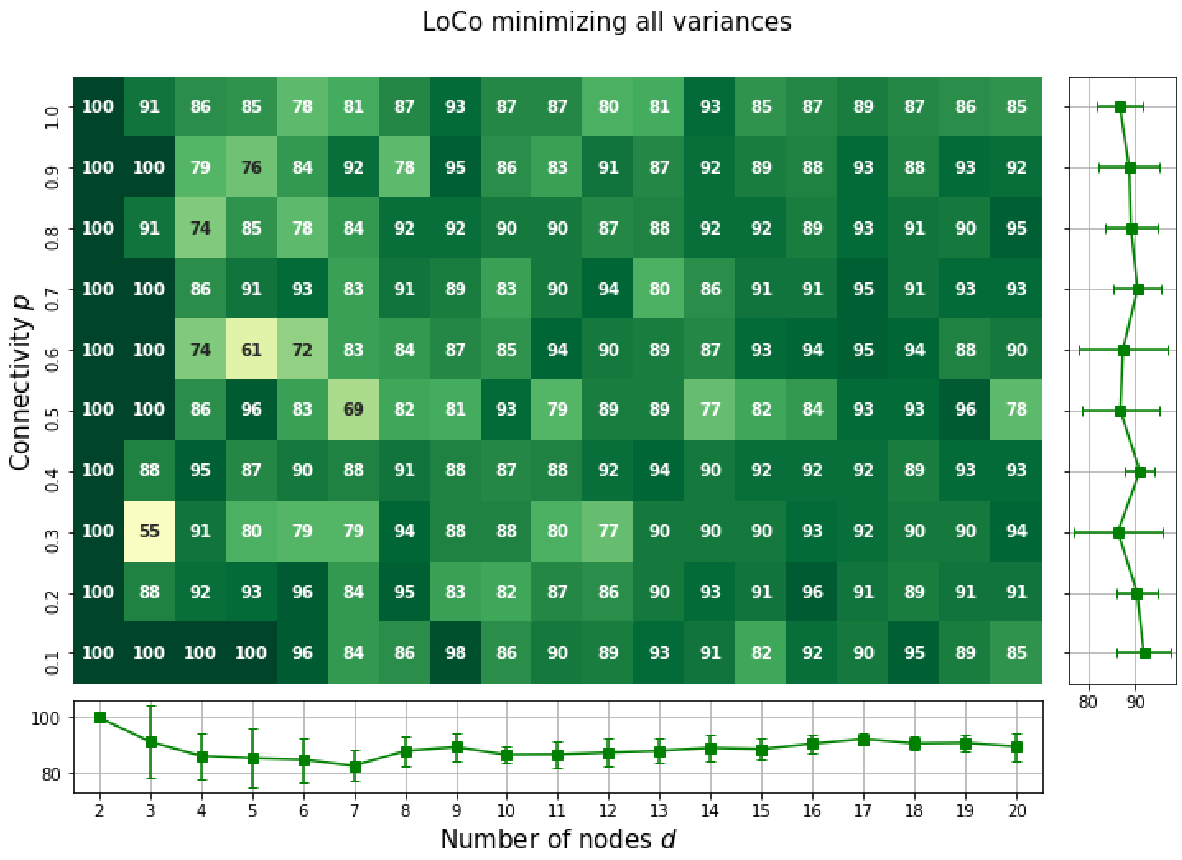
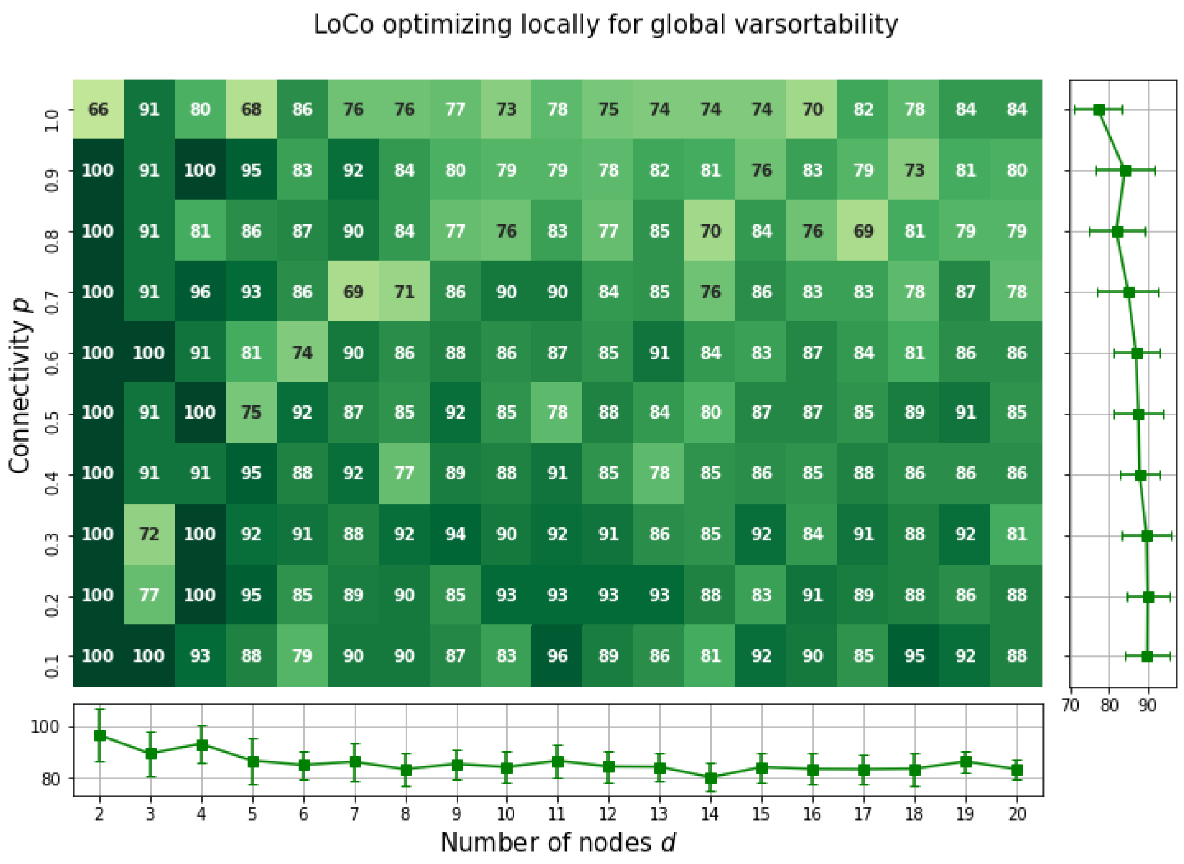


Figure 18: Figures of the varsortabilities achieved by Minimize-All LoCo variant on ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively.



Connectivity p	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1.0	66	91	80	68	86	76	76	77	73	78	75	74	74	74	70	82	78	84	84
0.9	100	91	100	95	83	92	84	80	79	79	78	82	81	76	83	79	73	81	80
0.8	100	91	81	86	87	90	84	77	76	83	77	85	70	84	76	69	81	79	79
0.7	100	91	96	93	86	69	71	86	90	90	84	85	76	86	83	83	78	87	78
0.6	100	100	91	81	74	90	86	88	86	87	85	91	84	83	87	84	81	86	86
0.5	100	91	100	75	92	87	85	92	85	78	88	84	80	87	87	85	89	91	85
0.4	100	91	91	95	88	92	77	89	88	91	85	78	85	86	85	88	86	86	86
0.3	100	72	100	92	91	88	92	94	90	92	91	86	85	92	84	91	88	92	81
0.2	100	77	100	95	85	89	90	85	93	93	93	93	88	83	91	89	88	86	88
0.1	100	100	93	88	79	90	90	87	83	96	89	86	81	92	90	85	95	92	88

Figure 19: Figures of the varsortabilities achieved by Minimize Varsort LoCo variant on ANMs of differing node-count and connectivities. The results displayed are means over 3 samples. The marginal plots display means and standard deviations over connectivity and nodes respectively.

A.2 Time Analysis of LoCo

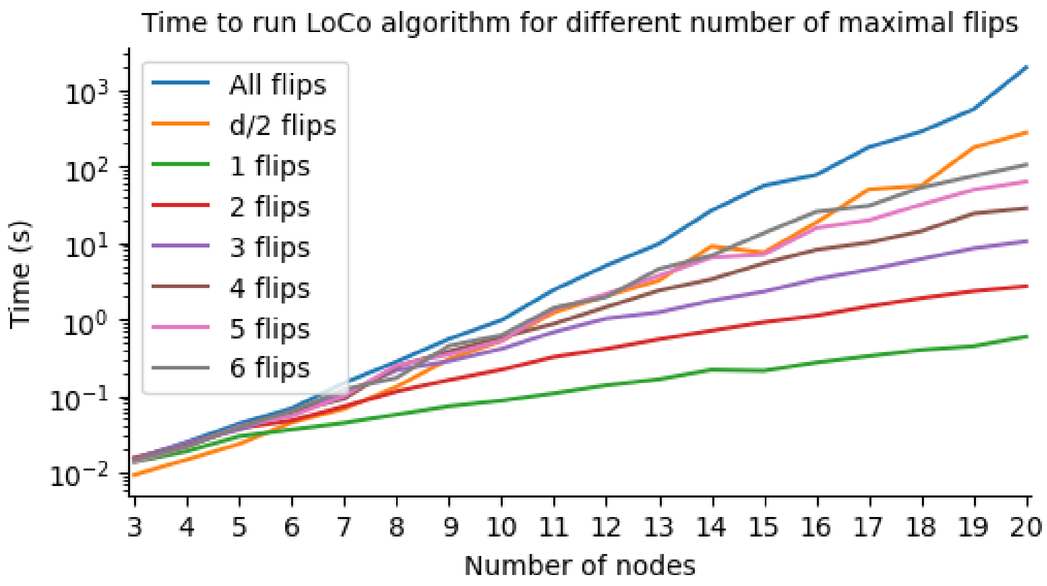


Figure 20: The computation time for a dag of size 20 goes from 0.5 seconds to 34 minutes!

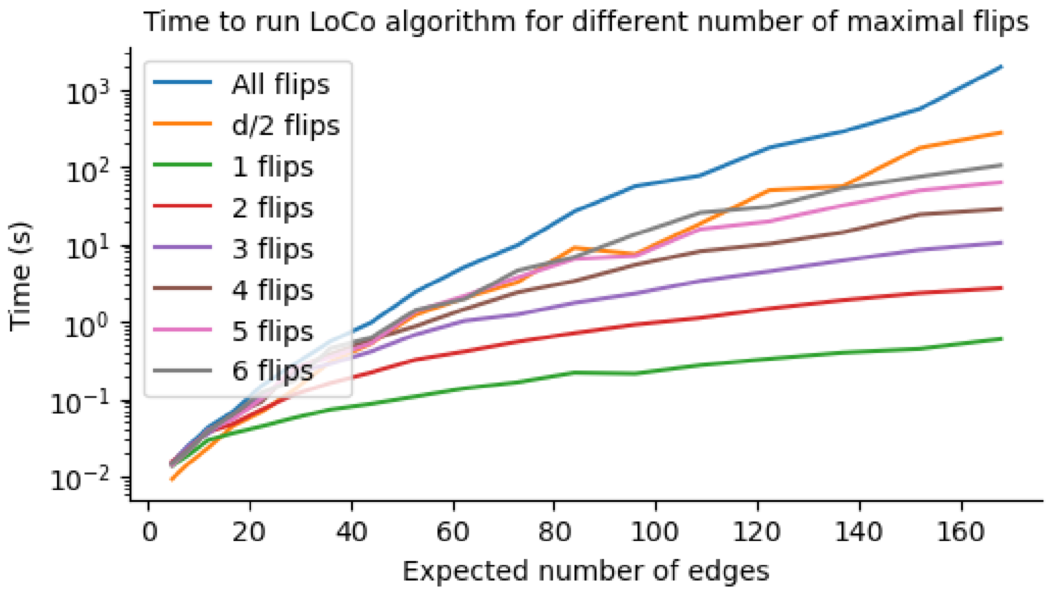


Figure 21: Again, LoCo is sub-exponential in number of edges

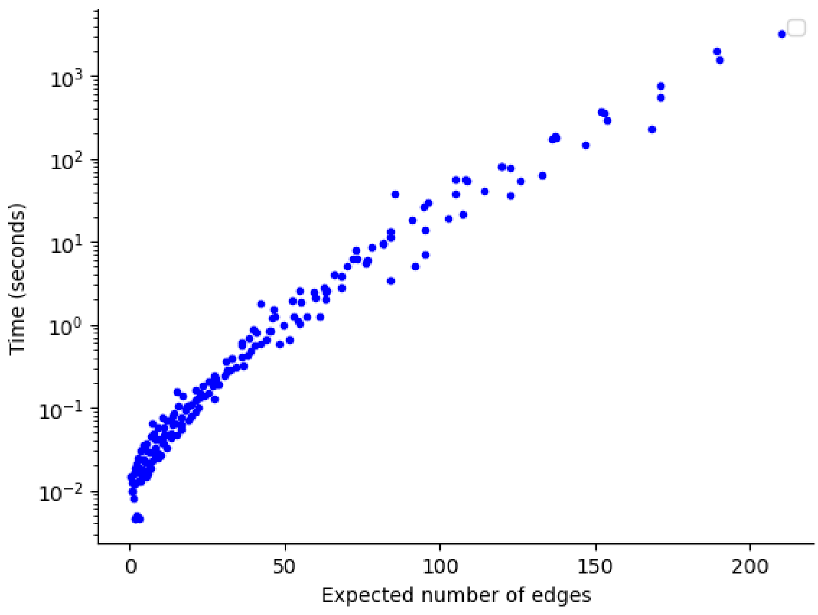


Figure 22: The time plot for the Genghis variant of LoCo when it is allowed to flip all of its children. As can be see, that while the time-complexity is bad, it is sub-exponential in number of edges. A better look at the growth rate can be seen in figure 23

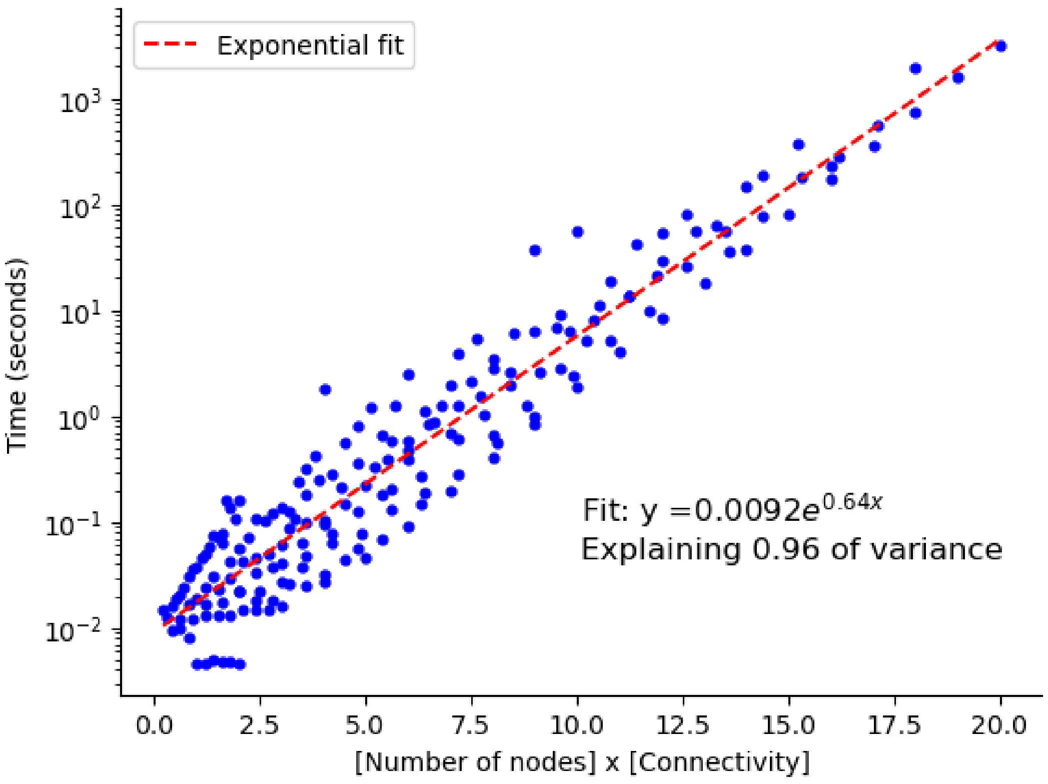


Figure 23: The time plot for the Genghis variant of LoCo when it is allowed to flip all edges. Curiously the computational time seems to grow remarkably $\propto e^d$. We believe this might have something to do with its combinatorical nature.

B Bound on Varsortability for Causal Graphs

As this is an important part of why our algorithms might be needed, we will present a bound on the var-sortabilitity in causal chains. Note that the reasoning presented here closely follows that of section 4 in [4], we choose to present it here as it makes for a good foundation for other discoveries.

Let $P_N(\phi)$ be a distribution with parameter ϕ controlling the standard deviation. Now, given an array of standard deviations $\Sigma = [\sigma_1, ..., \sigma_d]$ sampled as $\sigma_t \sim P_\sigma$, let $N = [\mathcal{N}_1, ..., \mathcal{N}_d]^T$ be the vector of noise variables in the graph sampled as $\mathcal{N}_t \sim P_N(\Sigma_t)$. Furthermore, let the weights of the edges be sampled as $w_{i,j} \sim P_W$. Now, given a graph $\mathcal{G}$ and noise distribution $P_N(\phi)$, the variances of the nodes are only determined by P_σ and P_W. Given this setup we can learn a few things about the varsortability of nodes in relation to their conditional variances, given everything but the noise vector N. For the simple graph $\mathcal{G}_2$ of a root note X_1 with a directed edge to a single other node X_2 we can bound the probability of $\text{Var}(X_1) < \text{Var}(X_2)$, i.e. the varsortability being 1, in the following way:

$$P_{W,\sigma}(\text{Varsort} = 1) \geq P(1 \leq |w_{1,2}|) \tag{10}$$

The probability of the varsortability being 1 is simply lower bounded by the probability of the edge weight between node X_1 and X_2 being lower than 1. For a one long causal chain graph $\mathcal{G}_p$ of length p, where we wish to compare the variances of X_1 and X_p this bound generalizes to

$$P_{W,\sigma}(\text{Var}(X_1) < \text{Var}(X_p)) \geq P\left(0 < \sum_{s=0}^{p-1} \ln\left(|w_{s,s1}|\right)\right) \tag{11}$$

We see, that for a P_W with even a slight probability of returning 1 or above, this bound goes to 0 as p tends to infinity. We thus see a possible mechanism for high varsortability in simulated ANM's, as the variance tends to increase along the causal chain and thereby likely also along a more general causal structure.

C Mein Further! I Can Work!

We had so many ideas for further work on this project. Each section below (summed up by the terms in **bold face** are the ones deemed most feasible/benifitting by us):

An idea was to train a yet-to-be-defined Machine Learning algorithm on **classifying** networks that has been modified by our algorithms from randomly sampled Erdös-Renyi networks.This would also allow us to run SHAP values (or some other explainability-measure) and maybe thereby get a deeper understanding of the makings of a non-varsortable network. It would also have been interesting seeing the results of a simple TSNE or other clustering algorithm to find common features between augmented data.

As the L2-norm seemed to change somewhat after running our algorithms, we thought about **rescaling** it back so as to obscurify our meddling even further. This would most likely change the varsortability, but we did not have the time to test the amount.

An idea we had, was to allow for the algorithm to **add new edges**. While we had a lot of different ideas for the implementation[3], it ended up being highly non-trivial to choose a heuristic approach as to balance being non-invasive while still improving performance.

As invasivity was the crux of our search for algorithms, we wanted to implement a genetic algorithm where the original batch of children were only allowed to **change a**

[3]maximally one per node, a random amount per DAG, only to the root or only to nodes with less than two parents, etc.

subselection of their parents weights. We believed this would give a competition where only the (say 3) most important weights were changed. As this obviously required larger populations we unfortunately had to cut the research for time.

Greedy Gradient Descent was an idea we (in concert with Weichwald) had, where instead of changing all the weights in the ANM concurrently, we would instead change only the weights around a single node at a time. This node could be chosen at random or sequentially. The hope was, that this would allow for use of a much simpler loss function such as eq. (6) or (7) and still avoid a global lowering of edge weights. We did implement a simple version of this, but the results were very lacking compared to WEEE! and the other implemented algorithms. When adjusting the algorithm we found that either the quality of the algorithm fell or we saw no change in the output from a normal WEEE!-run. We do however believe, that given some further tweaking, this approach could possibly produce some good results.

We thought about implementing **Riemannian Gradient Descent**, as this method would essentially allow us to 'lock in' the norm of of the weights W, and global edge weight lowering would therefore become impossible. This is achieved by only allowing for optimization along the surface of the sphere with radius $r = ||W_{init}||_2$. We unfortunately did not have the time to implement this ourselves, but we thought the idea very interesting and thus wanted to jot it down as a suggestion for further work. As a pseudo-implementation of the above we spent some time trying to add a 'destruction score' to the loss function to limit the total change in the weight matrix. This left us with an inversely proportional relationship between the achieved varsortability-decrease and the amount of destruction - a realistic trade-off does not seem unfeasable. Having the loss function be dependent on the iteration number (ie. only care about the amount of destruction *after* finding a minimum) improved this somewhat.

For LoCo two ideas where implemented halfways: Finding and **only looking at certian edges** with paths coming from the same node[4] could also improve the algorithm as the extra computation time would be mitigated by the lower number of possible combinations for each node. This is fully implemented and can be used by changing a bool. But as it to our surprise did not change the running time we refrained from using it.

Caching would be very beneficial! A lot of time could be saved by caching the data once a part of the causal chain was optimized. This would undoubtedly be an exponential speed-up, as the variance calculations are part of the time-bottleneck for LoCo (together with the $n!$ scaling of combinations of edges). We wrote a custom data-simulation algorithm that would allow for this change, but we unfortunately never got to the point of having a working implementation.

[4]as this is what gives rise to possible covariance

D Details on the Algorithms

D.1 GREG

D.1.1 Implementation

Algorithm 1 GREG

```
def GREG(anm : ANM):

    # start out by initializing a population
    new_anms = []
    for _ in population_size:
        new_anms.append(new_anm.mutate())

    # run the algorithm
    for _ in N_generations:
        #sort the population by culling metric
        sorted_anms = sort(new_anms)

        # copy first half
        new_anms = sorted_anms[:(population_size // 2)]

        # fill second half with variations on the first
        for i in range(population_size // 2):
            new_anms.append(start_anms[i].mutate())

    # return a sorted list of the best variations
    return sort(new_anms)
```

Each ANM has a `mutate()`-method that returns a mutated version of itself. This can be seen in algorithm 2

Algorithm 2 GREG mutate()

```
def mutate(self, p : float, strength : float):
    for edgeweight in self.adjacency_matrix:
        if edgeweight == 0:
            continue

    if random() < p:
        connection += normal(strength)
```

Where `normal(strength)` pulls a number from a normal distributions centered in 0 with a standard deviation of `strength` while making sure the edges to not go out of bound.

D.1.2 Additional Implementation Notes

Algorithm 1 and 2 outline the main functionality of GREG, but we heuristically found the few following additions to be beneficial.

- In order to avoid the algorithm getting stuck in local minima, a random ANM is also added to the population for the first 50% of generations.
- The amount of change per mutation was a vital hyperparameter. Much like an adaptive learning rate, we let the mutation-probability fall over the course of generations, as we found that this netted us better results.
- We restrict the search domain of the weights to be $[-2, .1] \cup [-.1, 2]$ instead of the more intuitive $[-2, -.5] \cup [.5, 2]$ (the domain that we initially sample weights from) as to better be able to compare it with our other algorithm as described in section 3.3.

D.2 WEEE!

D.2.1 Implementation

Algorithm 3 WEEE!

As in the case of GREG we restricted the weight search domain to $[-2, -.1] \cup [.1, 2]$.
A short pseudo-code is provided here:

```
#initialize ANM
ANM = ANM.init()

# Looping for a set number of epochs
# or until satisfactory varsortability is reached
Loop:
    #  Finding the gradients for the weights
    #  by finite difference method
    initial_position = loss_fn(ANM)
    for i,edge_weights in enumerate(ANM):

        # define what a small step in each direction entails
        step_forward  = edge_weight + delta

        # finite difference gradients are calculated
        gradient[i] = (loss_fn(step_forward) - initial_position)/delta

    # move in the opposite direction of the gradient
    ANM -= learning_rate * gradient

    check if desired varsort is reached or final epoch

# (maybe) done
return ANM
```

D.2.2 Additional Implementation Notes

We found that most of our troubles in the implementation came from the fact that if the network is too big, the loss function cannot calculated analytically and will have jitter on a scale comparable to the change over a small step size! We seem to have found a point in the trade-off between speed of every iteration and speed of convergence, but it required quite a lot of simple trial and error. For notes on the implementation check our GitHub: `https://github.com/JakobSchauser/PUK_softwoods`

D.3 LoCo

D.3.1 Implementation

In very simple pseudo-code, the *Max-top-third*-variant of LoCo does the following:

Algorithm 4 LoCo

```
for i,node in enumerate(ANM) do:

    try all sign flips for edges into node

    if i < (number of nodes in  ANM) // 3 do:
        pick the sign flips that yield the largest variance for node
    else:
        pick the sign flips that yield the smallest variance for node

return ANM
```

D.3.2 Additional Implementation Notes

An unexpected result was, that changing the optimizing metric to global Varsortability without changing the algorithm only lowered the performance of the algorithm!

Choosing the edges to flip (as we will mention in section D.3.2), the algorithm simply tries every possibility. This scales as 2^N and is the source of the main bottleneck for LoCo. Therefore we have implemented a ”flip-up-to” parameter that limits the maximal number of flipped bits and therefore shrinks the parameter space drastically for large networks. As can be seen in figure 9 (in appendix) this can improve computation time by multiple orders of magnitude without loosing all its effect.