Lab Rotation Report
NEM inference method: NEMorderMCMC

Sarah Morillo Leonardo

July 12, 2021

Contents

1 Introduction

Nested Effects Models (NEMs) are a statistical model to uncover hierarchical relationships between entities, such as for example certain genes of interest, from perturbation data. They infer that a certain gene A operates upstream of a gene B in a pathway if the downstream effects resulting from knocking down gene B are a noisy subset of those resulting from knocking down gene A. The perturbed genes in the signaling pathway are called *S-genes* and we are interested in finding the S-gene network. The genes that show expression changes in response to perturbation are called *E-genes*. These form the leaves of the signaling network [1, 2].

The caveat of this method is that the naive and exact approach to discover these relationships and infer a network is computationally not feasible for large networks. This is why we are in need of an approximate method that can discover such relations in a reasonable time frame. The proposed method NEMORDERMCMC infers networks from perturbation data with a good trade-off between accuracy and time. The idea behind this is to search in the order (permutations of S-genes) space instead of the much larger DAG space. In every iteration we propose a new order and find the highest scoring DAG for this order and the given data. To find this best fitting DAG we came up with three different approaches. The first one is via numerical optimization, the second via the gradient of the likelihood function and the last using greedy optimization. Then we compared these variants to three already existing NEM inference methods.

2 Method

In this section we will first see the theoretical and mathematical background of how to find the highest scoring DAG for a given order π and after that how we actually implemented it and some pseudo-code.

The objective is to find a S-gene network that best fits the data for a given order π. Note that the following ideas are taken from Dr. Kuipers handout.

2.1 Experimental Data

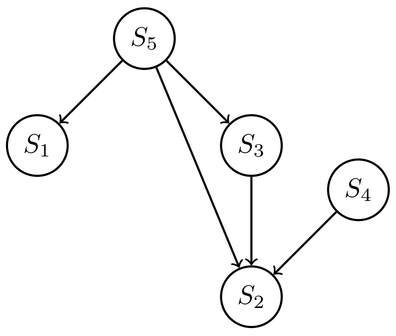


Figure 1: Signaling network: a transitively closed DAG

We are going to go through the method using the following example: In Figure 1 we see the true underlying signaling network of five S-genes $\mathcal{S} = \{S_1, ..., S_5\}$. The corresponding attachment vector σ of eight E-genes is

$$\sigma = (3, 3, 2, 2, 3, 5, 4, 1)$$

i.e S-gene 3 is parent of E-genes 1,2 and 5. Note that in reality we do not know these attachment vectors, which is why we will make use of the concept of marginalization. I.e we will marginalize over the attachments of the E-genes.

Knockdown experiments were conducted where each S-gene got knocked down once. The result is a binary knockdown table K. For our example we observed an effect in E_5 and E_8 when we knocked down S_1, see Table 1.

$$K = \begin{array}{c|cccccccc} & E_1 & E_2 & E_3 & E_4 & E_5 & E_6 & E_7 & E_8 \\ \hline S_1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ S_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ S_3 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ S_4 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ S_5 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

Table 1: Observation of knockdown experiments, knockdown table K

Notation

$$m = \text{number of E-genes} \qquad n = \text{number of S-genes}$$
$$\alpha = \text{false positive rate} \qquad \beta = \text{false negative rate}$$
$$A = \log(\frac{\alpha}{1-\beta}) \qquad B = \log(\frac{\beta}{1-\alpha})$$

Note that the experimental data are these knockdown table which are usually the input data for the NEM inference methods. But for our approach we need to further process them before we can input them into our method. This process is described in the next subsection.

2.2 Log Score Tables

Given a knockdown table K we build n score tables. The i-th score table T^i of S_i is computed as followed: First, we compute the *base row*. By *base row* we mean row S_i of score table T^i. For this we assume that S-gene S_i is the only parent of every E-gene and hence we expect to see an effect in each E-gene when S_i is knocked down, i.e. we set

$$F_j^i := \begin{cases} 0 & \text{if } K_{S_i,j} = 1 \\ B & \text{if } K_{S_i,j} = 0 \end{cases} \qquad , \qquad j = 1, ..., m$$

and no effect when another S-gene is knocked down, i.e.

$$G_j^i := \sum_{S \in \mathcal{S} \setminus S_i} \mathbb{1}\{K_{S,j} = 1\} A \qquad , \qquad j = 1, ..., m$$

Finally we combine the results from all the knockdown experiments into one single score in the *base row*:

$$T_{S_i,j}^i := F_j^i + G_j^i = \mathbb{1}\{K_{S_i,j} = 0\} B + \sum_{S \in \mathcal{S} \setminus S_i} \mathbb{1}\{K_{S,j} = 1\} A \qquad , \qquad j = 1, ..., m$$

The rest of the table is computed similarly, e.g. to compute row $T_{S_k,\cdot}^i$ for $S_k \in \mathcal{S} \setminus S_i$ we assume that S_k is parent of S_i. Therefore we would expect to see an effect on all E-genes when S_i or/and S_k is/are knocked down (and no effect if any of the other S-genes is knocked down), then we combine again the results into one score but we only store the increase in score (not the total scores), i.e

$$\begin{aligned} T_{S_k,j}^i &:= \left(\sum_{S \in \{S_i, S_k\}} \mathbb{1}\{K_{S,j} = 0\} B + \sum_{S \in \mathcal{S} \setminus \{S_i, S_k\}} \mathbb{1}\{K_{S,j} = 1\} A \right) - T_{S_i,j}^i \\ &= \mathbb{1}\{K_{S_k,j} = 0\} B - \mathbb{1}\{K_{S_k,j} = 1\} A \\ &= \begin{cases} B & , \ K_{S_k,j} = 0 \\ -A & , \ K_{S_k,j} = 1 \end{cases} \qquad , \qquad j = 1, ..., m \end{aligned}$$

$T^1 =$

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	B+A	B+A	B+4A	B+2A	2A	B+A	B+A	B
S_2	+B	+B	-A	+B	+B	+B	+B	+B
S_3	+B	+B	-A	-A	-A	+B	+B	+B
S_4	+B	+B	-A	-A	+B	+B	-A	+B
S_5	-A	-A	-A	+B	-A	-A	+B	-A

Table 2: Score table for node S_1, highlighted is the *base row*

The score table T^1 for S_1 in our example can be seen in Table 2.

Lastly, for convenience, we define a matrix U of dimension $(n+1) \times m$ which contains all the base rows and an additional row for dummy variable S_0 (which represents the case when all E-genes are disconnected, i.e. we would expect to see no effect for all S-genes).

$$U_{i,.} := T^i_{i,.} \qquad i = 1, ..., n$$
$$U_{n+1,.} := \sum_{S \in \mathcal{S}} \mathbb{1}\{K_{S,j} = 1\}A$$

Note that all the scores are in log-space. After this step we have a list of n score tables $\mathcal{T} = (T^1, ..., T^n)$.

2.3 Consider Order

A permutation of n S-genes gives us an order of them. With the aim of avoiding cycles in the DAG we need to respect that order. This means that a S-gene is only allowed to have parents form further up the chain, i.e. following it in the ordering.

For a given order, we simply remove rows of the table with parents outside the permissible set. We do this for all score tables

$$T^{i,\prec} = T^i_{Pa(S_i),.} \qquad i = 1, ..., n$$
$$Pa(S_i) = \{S | S \in \pi(\pi^{-1}(S_i) + 1, ..., n)\}$$

Assume the permutation order of the signaling genes in our example is the following

$$\pi = (4,\ 5,\ 2,\ 3,\ 1)$$

S_2 can have S_3 and S_1 as parents, therefore we only keep the S_1 and S_3 row of score table T^2 as can be seen in Table 3.

$T^{2,\prec} =$

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	+B	+B	+B	+B	-A	+B	+B	-A
S_3	+B	+B	-A	-A	-A	+B	+B	+B

Table 3: Score table with only permissible parents for node S_2

We are left with a list of reduced tables: $\mathcal{T}^{\prec} = (T^{1,\prec}, ..., T^{n,\prec})$.

Interpretation The row $U_{i,.}$ is the log score of having just S-gene i as a parent for each E-gene. Therefore, we can score different networks with the score tables: $U_{i,j}$ contains the log score of the connection between S_i and E_j assuming that S_i is the only parent of E_j. But if we want to score a network, where S_k is parent of S_i (and S_i still parent of E_j and S_k is a permissible parent of S_i) we just add $T^{i,\prec}_{S_k,j}$ to $U_{i,j}$. This is the rough idea, but how we actually score a network we will see in the next section.

2.4 Compute Log Likelihood

2.4.1 Parent Weights

For a given order, we want to know which is the "best" DAG (i.e. the highest scoring). But since we don't know yet which S-gene is parent of whose, we introduce auxiliary variables/ parent weights $w_{i,j}$, $i = 1, ...n, j \in Pa(S_i)$. Note that we have in total $W = \frac{n(n-1)}{2}$ weights. We initialize $w_{i,j} = 0.5 \, \forall i, j$. The goal is to find the optimal weights, i.e $\forall i, j, w^*_{i,j} \in \{0, 1\}$. The interpretation of these weights is the following: Assume in the real network $S_k \in Pa(S_i)$ is parent of S_i and $S_l \in Pa(S_i)$ is not, then we would wish to find $w^*_{i,k}$ to be 1 and $w^*_{i,l}$ to be 0. So at the end, $w^*_{i,j}$ indicates whether S_j is parent of S_i or not. Therefore our objective is to find

$$w^* = \underset{w \in \{0,1\}^W}{\arg\max} \, \mathcal{L}(w, \pi)$$

for a given order π, where $\mathcal{L}$ is our likelihood function.

2.4.2 Total scores

For n S-genes there are at most 2^{n-1} possible combinations of parents for one S-gene. But because not all parents are permissible it is actually less: $2^{|Pa(S_i)|}$ possible combinations for S_i. We want to compute the total scores of all possibilities in real space (this is why we take the exponential of the log scores).

If we look at our example order, S_4 can have S_1, S_2, S_3 and S_5 as parents or any combination of these :{}, S_1,S_2, S_3, S_5, $\{S_1, S_2\}$, ..., $\{S_1, S_2, S_3\}$,..., $\{S_1, S_2, S_3, S_5\}$. Assuming E_1 is attached to S_4, the total score would be computed in as followed

$$\begin{aligned}
&e^{U_{S_4,1}} + e^{U_{S_4,1}+T^{4,\prec}_{S_1,1}} + ... + e^{U_{S_4,1}+T^{4,\prec}_{S_5,1}} + e^{U_{S_4,1}+T^{4,\prec}_{S_1,1}+T^{4,\prec}_{S_2,1}} + ... + e^{U_{S_4,1}+T^{4,\prec}_{S_3,1}+T^{4,\prec}_{S_5,1}} \\
&\quad + e^{U_{S_4,1}+T^{4,\prec}_{S_1,1}+T^{4,\prec}_{S_2,1}+T^{4,\prec}_{S_3,1}} + ... + e^{U_{S_4,1}+T^{4,\prec}_{S_2,1}+T^{4,\prec}_{S_3,1}+T^{4,\prec}_{S_5,1}} + e^{U_{S_4,1}+T^{4,\prec}_{S_1,1}+T^{4,\prec}_{S_2,1}+T^{4,\prec}_{S_3,1}+T^{4,\prec}_{S_5,1}} \\
&= (e^{U_{S_4,1}} + e^{U_{S_4,1}+T^{4,\prec}_{S_1,1}} + e^{U_{S_4,1}+T^{4,\prec}_{S_2,1}} + e^{U_{S_4,1}+T^{4,\prec}_{S_1,1}+T^{4,\prec}_{S_2,1}})(1 + e^{T^{4,\prec}_{S_3,1}})(1 + e^{T^{4,\prec}_{S_5,1}}) \\
&= (e^{U_{S_4,1}} + e^{U_{S_4,1}} e^{T^{4,\prec}_{S_1,1}})(1 + e^{T^{4,\prec}_{S_2,1}})(1 + e^{T^{4,\prec}_{S_3,1}})(1 + e^{T^{4,\prec}_{S_5,1}}) \\
&= e^{U_{S_4,1}}(1 + e^{T^{4,\prec}_{S_1,1}})(1 + e^{T^{4,\prec}_{S_2,1}})(1 + e^{T^{4,\prec}_{S_3,1}})(1 + e^{T^{4,\prec}_{S_5,1}})
\end{aligned}$$

We see that the sum of 16 terms (all the different combinations of parents) at the beginning can be expressed more compactly as a product. In general we exponentiate U and the score tables, add 1 to all rows of the tables and take the product. Assuming E_j is attached to S_i we would compute the total score in the following way

$$e^{U_{i,j}} \prod_{S \in Pa(S_i)} (1 + e^{T^{i,\prec}_{S,j}})$$

We compute this in real space but at the end we want to have log scores again, therefore we take the log and see that

$$\log \left(e^{U_{i,j}} \prod_{S \in Pa(S_i)} (1 + e^{T^{i,\prec}_{S,j}}) \right) = \log(e^{U_{i,j}}) + \sum_{S \in Pa(S_i)} \log(1 + e^{T^{i,\prec}_{S,j}})$$

2.4.3 Combine Parent Weights and Total Scores

Generally we exponentiate, add 1 and multiply the scores together but we cannot do this so straightforwardly since we don't know which S-gene is parent of which S-gene. So we include the parent weights. A illustration is given in the Table 4.

For table $T^{i,\prec}$ we have the following expression

$$e^{U_{i,j}} \prod_{S \in Pa(S_i)} (1 - w_{i,S} + w_{i,S} e^{T^{i,\prec}_{S,j}})$$

$$\tilde{T}^2 = \begin{array}{c|cccc} & E_1 & E_2 & E_3 & \dots \\ \hline S_1 & (1 - w_{2,1} + w_{2,1}e^{B}) & (1 - w_{2,1} + w_{2,1}e^{B}) & (1 - w_{2,1} + w_{2,1}e^{B}) & \dots \\ S_3 & (1 - w_{2,3} + w_{2,3}e^{B}) & (1 - w_{2,3} + w_{2,3}e^{B}) & (1 - w_{2,3} + w_{2,3}e^{-A}) & \dots \end{array}$$

Table 4: Weighted score table with permissible parents for node S_2

As above we take the log to store the scores in log space again.

$$\begin{aligned} x_{i,j} : &= \log\left(e^{U_{i,j}} \prod_{S \in Pa(S_i)} (1 - w_{i,S} + w_{i,S}e^{T^{i,\prec}_{S,j}})\right) \\ &= U_{i,j} + \sum_{S \in Pa(S_i)} \log(1 - w_{i,S} + w_{i,S}e^{T^{i,\prec}_{S,j}}) \end{aligned} \tag{1}$$

The rational behind this is that the optimal weights should be either 0 or 1. For example say S is not parent of S_i, then we have $w_{i,S} = 0$ and therefore we add $\log(1-0+0*e^{T^{i,\prec}_{S,j}}) = \log(1) = 0$, i.e. it has no contribution to the score. On the other hand if S would be parent of S_i then $w_{i,S} = 1$ and we add $\log(1 - 1 + 1 * e^{T^{i,\prec}_{S,j}}) = \log(e^{T^{i,\prec}_{S,j}}) = T^{i,\prec}_{S,j}$ to the score.

At the end of this step, we have a matrix with the weighted log scores

$$X := (x_{i,j})_{i=1,\dots,n,j=1\dots,m}$$

Interpretation We can think that $x_{i,j}$ contains the score that E_j is connected to S_i taking into consideration all the possible parents for S_i.

2.4.4 Likelihood

Finally we can compute the total log likelihood using *log-add*, i.e. we add in real space and convert it to log space at the end. Remember that each E-gene can only be connected to at most one S-gene. Since we don't know to which S-gene a specific E-gene is connected, we marginalize over the attachment of the E-genes

$$\hat{X}_j := \log(\sum_{i=1}^{n+1} \exp(x_{i,j})) \quad \forall j = 1, ..., m$$

Finally we obtain the total log-likelihood for a given order

$$\begin{aligned} \mathcal{L}(w, \pi) &= \sum_{j=1}^{m} \hat{X}_j \\ &= \sum_{j=1}^{m} \log(\sum_{i=1}^{n+1} \exp(U_{i,j} + \sum_{S \in Pa(S_i)} \log(1 - w_{i,S} + w_{i,S}e^{T^{i,\prec}_{S,j}}))) \end{aligned}$$

2.4.5 Cell Weights/Probabilities

$\hat{X}_j$ can be understood as the cumulative/total probability of an E-gene and can be seen as a normalizing factor. For each E-gene we want to have the probability that it is attached to a specific S-gene. We can obtain these by dividing the columns of X by the normalizing factors and taking the exponential to obtain values in real space. Note that we are working with log scores, so division becomes subtraction, i.e

$$\begin{aligned} p_{i,j} &:= \exp(x_{i,j} - \hat{X}_j) \\ &= \frac{\exp(x_{i,j})}{\exp(\hat{X}_j)} \\ &= \frac{\exp(x_{i,j})}{\sum_{k=1}^{n+1} \exp(x_{k,j})} \end{aligned} \tag{2}$$

2.5 Objective

The objective is to find a fast optimization method to solve the following problem

$$w^* = \underset{w \in \{0,1\}^W}{\arg\max}\ \mathcal{L}(w, \pi)$$

2.6 Optimization

Instead of maximizing all weights simultaneously we do this iteratively. We set all weights $w_{i,j} = 0.5$. Then we maximize the weight $w_{i,j}$ keeping all the other weights fixed. Once we have found the maximum $w^*_{i,j}$ we maximize $w_{k,l}$, given $w^*_{i,j}$ and keeping all the other weights fixed, etc. So we maximize one weight, keep the others fixed, then in the next iteration we maximize another weight, using the optimal weights from previous iteration steps (it's similar to the idea of Gibb's sampling). Therefore, we want to know how the log-likelihood changes when we vary one single parent weight.

Assume $w = (w_{i,j})_{i=1,\dots n, j \in Pa(S_i)}$ are the weights from the previous optimization step. Let $\hat{w}_{l,t}$ be the weight we want to optimize in the current step, set $w' = \hat{w}_{l,t} \cup (w_{i,j})_{i=1,\dots n, i \neq l, j \in Pa(S_i), j \neq t}$. Look at the difference in log-likelihood

$$\begin{aligned}
\mathcal{L}(w', \pi) - \mathcal{L}(w, \pi) &= \sum_{j=1}^{m} \log(\sum_{i=1}^{n+1} \exp(U_{i,j} + \sum_{S \in Pa(S_i)} \log(1 - w'_{i,S} + w'_{i,S} e^{T^{i,\prec}_{S,j}}))) \\
&\quad - \sum_{j=1}^{m} \log(\sum_{i=1}^{n+1} \exp(U_{i,j} + \sum_{S \in Pa(S_i)} \log(1 - w_{i,S} + w_{i,S} e^{T^{i,\prec}_{S,j}}))) \\
&= \sum_{j=1}^{m} \log \left(\frac{\sum_{i=1}^{n+1} \exp(U_{i,j} + \sum_{S \in Pa(S_i)} \log(1 - w'_{i,S} + w'_{i,S} e^{T^{i,\prec}_{S,j}})}{\sum_{i=1}^{n+1} \exp(U_{i,j} + \sum_{S \in Pa(S_i)} \log(1 - w_{i,S} + w_{i,S} e^{T^{i,\prec}_{S,j}}))} \right) \\
&\overset{(1)}{=} \sum_{j=1}^{m} \log \left(\frac{\sum_{i=1}^{n+1} \exp(x'_{i,j})}{\sum_{i=1}^{n+1} \exp(x_{i,j})} \right) \\
&= \sum_{j=1}^{m} \log \left(\frac{\sum_{i=1}^{n+1} \exp(x_{i,j}) - \exp(x_{l,j}) + \exp(\hat{x}_{l,j})}{\sum_{i=1}^{n+1} \exp(x_{i,j})} \right) \\
&= \sum_{j=1}^{m} \log \left(1 - \frac{\exp(x_{l,j})}{\sum_{i=1}^{n+1} \exp(x_{i,j})} + \frac{\exp(\hat{x}_{l,j})}{\sum_{i=1}^{n+1} \exp(x_{i,j})} \right) \\
&\overset{(2)}{=} \sum_{j=1}^{m} \log \left(1 - p_{l,j} + \frac{\exp(\hat{x}_{l,j})}{\sum_{i=1}^{n+1} \exp(x_{i,j})} \right)
\end{aligned}$$

Let's look at the third component in more detail

$$\begin{aligned}
\frac{\exp(\hat{x}_{l,j})}{\sum_{i=1}^{n+1} \exp(x_{i,j})} &\overset{(1)}{=} \frac{\exp\left(U_{l,j} + \log(1 - \hat{w}_{l,t} + \hat{w}_{l,t} e^{T^{l,\prec}_{t,j}}) + \sum_{S \in Pa(S_i), S \neq t} \log(1 - w_{l,S} + w_{l,S} e^{T^{l,\prec}_{S,j}})\right)}{\sum_{i=1}^{n+1} \exp(x_{i,j})} \\
&\overset{(2)}{=} \frac{\exp\left(U_{l,j} + \log(1 - \hat{w}_{l,t} + \hat{w}_{l,t} e^{T^{l,\prec}_{t,j}}) + \sum_{S \in Pa(S_i), S \neq t} \log(1 - w_{l,S} + w_{l,S} e^{T^{l,\prec}_{S,j}})\right)}{\frac{\exp(x_{l,j})}{p_{l,j}}} \\
&\overset{(1)}{=} p_{l,j} \frac{e^{U_{l,j}} (1 - \hat{w}_{l,t} + \hat{w}_{l,t} e^{T^{l,\prec}_{t,j}}) \prod_{S \in Pa(S_i), S \neq t} (1 - w_{l,S} + w_{l,S} e^{T^{l,\prec}_{S,j}})}{e^{U_{l,j}} \prod_{S \in Pa(S_i)} (1 - w_{l,S} + w_{l,S} e^{T^{l,\prec}_{S,j}})} \\
&= p_{l,j} \frac{(1 - \hat{w}_{l,t} + \hat{w}_{l,t} e^{T^{l,\prec}_{t,j}})}{(1 - w_{l,t} + w_{l,t} e^{T^{l,\prec}_{t,j}})}
\end{aligned}$$

All together

$$\mathcal{L}(w',\pi) - \mathcal{L}(w,\pi) = \sum_{j=1}^{m} \log\left(1 - p_{l,j} + p_{l,j}\frac{1-\hat{w}_{l,t}+\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}}}{1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}}\right) \tag{3}$$

and we see that the likelihood of the new weights are dependent on the likelihood of the old weights

$$\mathcal{L}(w',\pi) = \mathcal{L}(w,\pi) + \sum_{j=1}^{m} \log\left(1 - p_{l,j} + p_{l,j}\frac{1-\hat{w}_{l,t}+\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}}}{1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}}\right) \tag{4}$$

In each iteration we optimize one weight component

$$\begin{aligned} w^*_{l,t} &= \underset{\hat{w}_{l,t}}{\arg\max}\, \mathcal{L}(w',\pi) \qquad\qquad w' = \hat{w}_{l,t} \cup (w_{i,j})_{i=1,\ldots n, i\neq l, j\in Pa(S_i), j\neq t} \\ &= \underset{\hat{w}_{l,t}}{\arg\max}\, \mathcal{L}(w,\pi) + \sum_{j=1}^{m} \log\left(1 - p_{l,j} + p_{l,j}\frac{1-\hat{w}_{l,t}+\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}}}{1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}}\right) \\ &= \underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\left(1 - p_{l,j} + p_{l,j}\frac{1-\hat{w}_{l,t}+\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}}}{1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}}\right) \end{aligned}$$

Observe that since the $w_{l,t}$ is known from the previous step, the denominator is constant. Hence

$$\begin{aligned} &\underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\left(1 - p_{l,j} + p_{l,j}\frac{1-\hat{w}_{l,t}+\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}}}{1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}}\right) \\ &\quad= \underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\Big((1-p_{l,j})(1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}) + p_{l,j}(1-\hat{w}_{l,t}+\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}})\Big) \\ &\quad= \underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\Big((1-p_{l,j})(1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}) + p_{l,j} - p_{l,j}\hat{w}_{l,t} + p_{l,j}\hat{w}_{l,t}e^{T^{l,\prec}_{t,j}}\Big) \\ &\quad= \underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\Big((1-p_{l,j})(1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}) + p_{l,j} + \hat{w}_{l,t}p_{l,j}(e^{T^{l,\prec}_{t,j}}-1)\Big) \end{aligned}$$

We set (for $\hat{w}_{l,t}$ and E-gene j)

$$\begin{aligned} a^{l,t}_j &= p_{l,j}(e^{T^{l,\prec}_{t,j}}-1) \\ b^{l,t}_j &= 1 + w_{l,t}(e^{T^{l,\prec}_{t,j}}-1) - w_{l,t}a^{l,t}_j \end{aligned}$$

and observe that $b^{l,t}_j$ is constant and therefore

$$\begin{aligned} &\underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\Big((1-p_{l,j})(1-w_{l,t}+w_{l,t}e^{T^{l,\prec}_{t,j}}) + p_{l,j} + \hat{w}_{l,t}p_{l,j}(e^{T^{l,\prec}_{t,j}}-1)\Big) \\ &\quad= \underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\Big(b^{l,t}_j + \hat{w}_{l,t}a^{l,t}_j\Big) \\ &\quad= \underset{\hat{w}_{l,t}}{\arg\max} \sum_{j=1}^{m} \log\left(\hat{w}_{l,t}\frac{a^{l,t}_j}{b^{l,t}_j} + 1\right) \end{aligned}$$

With $c_j^{l,t} = \frac{a_j^{l,t}}{b_j^{l,t}}$ we obtain the likelihood function for one single weight

$$\mathcal{L}'(\hat{w}_{l,t}, \pi) = \sum_{j=1}^{m} \log\left(\hat{w}_{l,t} c_j^{l,t} + 1\right) \tag{5}$$

Our new objective is

$$w_{l,t}^* = \arg\max_{\hat{w}_{l,t}} \mathcal{L}'(\hat{w}_{l,t}, \pi) \tag{6}$$

2.7 Gradient

The gradient of the new objective function is

$$\begin{aligned}
\frac{\partial}{\partial \hat{w}_{l,t}} \mathcal{L}'(\hat{w}_{l,t}, \pi) &= \frac{\partial}{\partial \hat{w}_{l,t}} \sum_{j=1}^{m} \log\left(\hat{w}_{l,t} c_j^{l,t} + 1\right) \\
&= \sum_{j=1}^{m} \frac{\partial}{\partial \hat{w}_{l,t}} \log\left(\hat{w}_{l,t} c_j^{l,t} + 1\right) \\
&= \sum_{j} \frac{c_j^{l,t}}{\left(\hat{w}_{l,t} c_j^{l,t} + 1\right)}
\end{aligned}$$

2.8 Optimization Methods

To find the optimal weights we have three approaches.

2.8.1 Approach 1: Numerical

We iterate over all weights, calculate each time $c_j^{l,t}$ for all E-genes j (which gives us `Cvec`) and minimize `localLL` $= -\mathcal{L}'(\hat{w}_{l,t}, \pi)$ for a given order π using a numerical optimization function called `optimize` with the following parameters:

```
optimize(localLL, c(0, 1), tol = 0.01, c = Cvec)$minimum
```

And then we repeat this until the log likelihood of the network, given the current parent weights, converges (see Algorithm 1).

Algorithm 1 Optimization of Parent Weights

```
1: $oldLL$ = - inf ;  $LLdiff$ = -inf
2: $parentWeights_{i,j} = 0.5 \, \forall i, j$
3: while LLdiff > 0.1  do
4:    calculate log likelihood $LL$, given $parentWeights$
5:    $new\_parentWeights = parentWeights$
6:    for $\hat{w}_{i,j}$ in $parentweights$ do
7:       optimize log likelihood w.r.t. $\hat{w}_{i,j}$
8:       $new\_parentWeights_{i,j} = \hat{w}^*_{i,j}$
9:    end for
10:   $parentWeights = new\_parentWeights$
11:   $LLdiff = LL - oldLL$;  $oldLL = LL$
12: end while
13: return $new\_parentWeights$
```

2.8.2 Approach 2: Gradient

The idea is to compute the gradient at 0.5 and then project to either 0 or 1 depending on the sign of the gradient. So we do not really optimize the likelihood as in Approach 1 and are therefore faster but also less precise. Hence for a given order won't necessarily find the optimal weights.

As in Approach 1 we iterate again over all weights, calculate each time the corresponding $c_j^{l,t}\,\forall j$ values to compute the gradient at 0.5. Since we are searching for the maximum, when the gradient is negative the optimal weight is $\hat{w}^*_{l,t} = 0$ and otherwise 1 (see Algorithm 2).

Algorithm 2 Optimization of Parent Weights via Gradient

```
    function OPTIMIZATION_VIA_GRADIENT()
 1: $parentWeights_{i,j} = 0.5\,\forall i,j$
 2: $new\_parentWeights = parentWeights$
 3: for $\hat{w}_{l,t}$ in $parentweights$ do
 4:     compute $c_j^{l,t}\ \forall j$ w.r.t. $parentWeights$
 5:     compute gradient for $\hat{w}_{l,t}$ at 0.5
 6:     $new\_parentWeights_{l,t} = 0$ if gradient is negative, else $new\_parentWeights_{l,t} = 1$
 7: end for
 8: return $new\_parentWeights$
```

2.8.3 Approach 3: Greedy

The aim is to make Approach 2 more precise. The first step is to calculate the gradient at 0.5 and then project the weights to either 0 or 1. Then we perform a greedy search: We flip each weight and calculate the new log likelihood and see if we obtain a higher likelihood. If so, we take these new weights as a starting point and flip again all weights one by one and update the likelihood until the new likelihood is not better than the old likelihood (see Algorithm 3).

If the gradient is not accurate enough, we can make it more precise with this greedy step. But if the gradient is already sufficient then we skip the greedy step. Hence in the best case this approach is as fast as Approach 2.

Algorithm 3 Greedy Optimization of Parent Weights via Gradient

```
 1: $parentWeights$ = optimization_via_gradient()
 2: $new\_parentWeights = parentWeights$
 3: while $newLL > oldLL$ do
 4:     $oldLL$ = compute $LL$ w.r.t. $parentWeights$
 5:     for $\hat{w}_{l,t}$ in $parentweights$ do
 6:         flip weight: $new\_parentWeights_{l,t} = 1$ if $\hat{w}_{l,t} = 0$, else $new\_parentWeights_{l,t} = 0$
 7:         $newLL$ = update $LL$ w.r.t. $new\_parentWeights$
 8:     end for
 9:     $newLL$ is the largest value found in the for loop and $new\_parentWeights$ are the corre-
        sponding weights that led to that likelihood
10:     $parentWeights = new\_parentWeights$
11: end while
12: return $new\_parentWeights$
```

2.9 Finding the highest scoring DAG for a given order

Finally we put all these things together and obtain a method to find a high scoring DAG for a given order (see Algorithm 4).

Algorithm 4 Find highest scoring DAG for a given order π

```
    function NEMOrderScore
    input: startorder = $\pi$, score tables $\mathcal{T}$, opt_function $\in \{numerical, gradient, greedy\}$
1:  subset score tables w.r.t. to $\pi$, obtain $\mathcal{T}^{\prec}$
2:  compute optimal parent weights $w^*$ via opt_function($\mathcal{T}^{\prec}$)
3:  transitively close $w^*$
4:  compute $LL$ w.r.t. $w^*$
5:  compute adjacency matrix based on $w^*$
6:  return adjacency matrix, $LL$
```

2.10 Comparison of optimization methods

An exhaustive search for the parent weights was also implemented to compare these approaches. But since the exhaustive search takes long, the comparison could only be done in a small network with 5 S-genes (see file *compare_parentweights.R*). The results are summarized in Figure 2. For 9 different order of the S-genes, which are consistent with the order underlying the true network, we calculated the hamming distance between the parent weights vector from each method to the parent weight vector obtained by the exhaustive search. We can see that for all methods and error rates the average hamming distance is not larger than 2, i.e. the methods seem to perform as well as the exhaustive search but are for sure much faster. The version with the gradient (GDC) seem to perform worse than the numerical optimization (custom) and the greedy approach but is the fastest method.

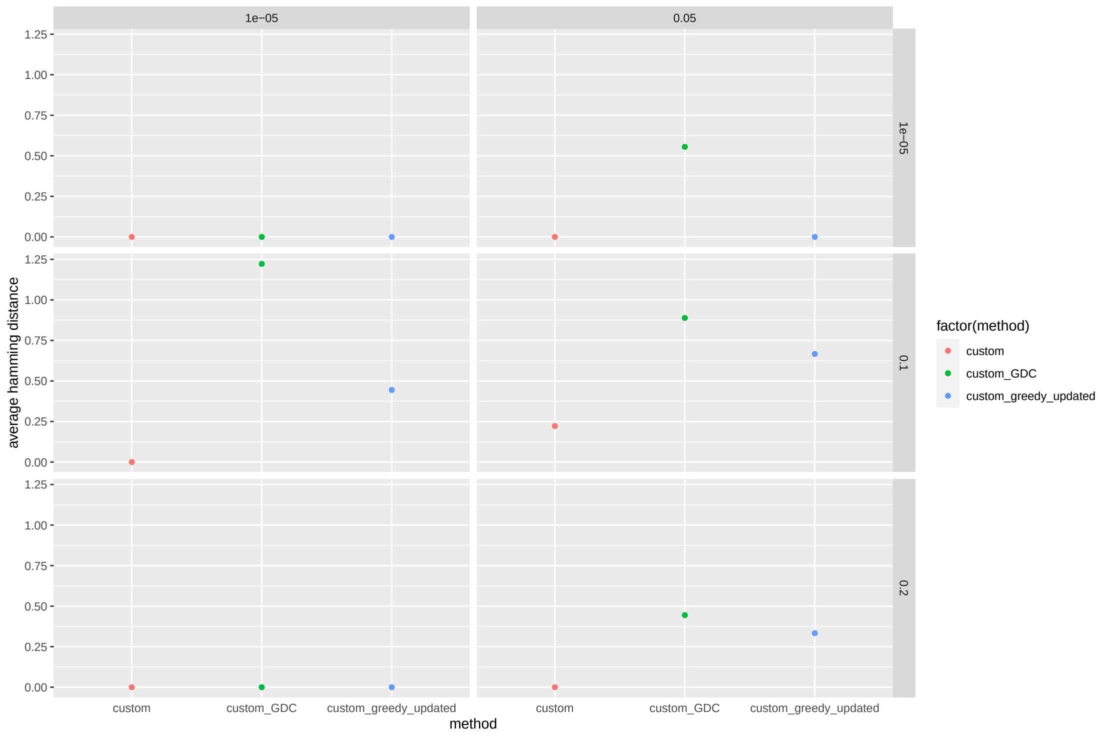


Figure 2: Comparison between variants of the optimization step for a network of 5 S-genes and 8 E-genes with error rates $\alpha = 10^{-5}, 0.05$ and $\beta = 10^{-5}, 0.1, 0.2$. The hamming distance between the result from the exhaustive search and the other approaches was computed and averaged across 9 different orders.

3 NEM Order-MCMC

The goal is the inference of NEMs from perturbation data. In the previous section we have seen different approaches to find the best network given a specific order of nodes. But if we have an order that is not consistent with the order underlying the network, the best fitting network can't be the true network. This is why we have to consider different orders and make use of the concept of *order MCMC*. The rough idea is to start with a order and calculate the log likelihood of the best fitting network via our approach from the previous section. Then we permute the order: either we swap any two elements at random (global move) or swap two adjacent elements (transposition move). With this new order we again search for the best fitting network and compute it's log likelihood. Afterwards we compute the acceptance probability ρ by dividing the old likelihood score by the new likelihood and multiplying it by an acceptance parameter γ. But because we have log likelihoods and we want ρ to be in real space we actually calculate

$$\rho = \exp\Big(\gamma * (LL_{old} - LL_{new})\Big) \tag{7}$$

and we accept this move only if $\rho > u$, where $u \sim Unif([0, 1])$.

The whole NEMOrderMCMC method is described in Algorithm 5.

Algorithm 5 NEMorderMCMC

```
    input: startorder = $\pi$, score tables $\mathcal{T}$, niterations, $\gamma$, opt_function, moveprobs
 1: $LL_{old}$ = NEMOrderScore($\pi$, $\mathcal{T}$, opt_function)
 2: current order $\pi' = \pi$
 3: for 1,..., niterations do
 4:   sample move $\in$ {transposition move, global move} $\sim$ moveprobs
 5:   propose order $\hat{\pi}$ = move($\pi'$)
 6:   $LL_{new}$ = NEMorderScore($\hat{\pi}$, $\mathcal{T}$, opt_function)
 7:   if $\rho > u$ then
 8:     accept $\hat{\pi}$ and update $\pi'$, $LL_{old}$ accordingly
 9:   end if
10:   keep track of best $LL$
11: end for
```

3.1 Hyper-parameter Search

For the NEMOrderMCMC we have to choose the number of iterations *niterations* and γ.

3.1.1 Number of iterations

In a first step different number of iterations ($\frac{1}{2}nS^2, nS^2, nS^2\log(nS)$, $\frac{1}{3}nS^2\log(nS), 2nS^2$) were compared in their performance (i.e. how the likelihood of the found network compares to the likelihood of the true network) keeping γ fixed at 1 (see files *find_optimal_parameters.R*, *find_optimal_parameters_plots.R* and *find_optimal_parameters_evaluation.R*). Another attempt to find a sensible choice for the number of iterations was to run the three different approaches with $2nS^2\log(nS)$ iterations and $\gamma = 1$ and then see after how many iterations the likelihood converges, i.e. does not improve anymore (see files *find_optimal_parameters_nitr.R*, *find_optimal_parameters_nitr_eval.Rmd* and *find_optimal_parameters_nitr_eval.html*). This comparison was done for different sized networks ($nS = 5, 10, 20$ and $nE = 8, 50, 100, 200$) with different error rates ($\alpha = 0.05$ and $\beta = 0.1, 0.2$) and is summarized in Figure 4 and Figure 3.

In Figure 4 we see at which iteration a certain method converges depending on the size of the network and the error rates. For example the *greedy* (greedy_updated) method converged after 92 iterations for the network of 10 S-genes, 100 E-genes and error rates $\alpha = 0.05$ and $\beta = 0.1$. All in all we see that choosing $niterations = nS^2\log(nS)$ is a sensible choice. Note that these are based on one run of the methods per network and error rates, so the results might not be robust. It would be better to run the methods several times per network and error rate and average over the outcomes to obtain more consistent results but due to time constraints this has not been done.

In Figure 3 we see how long each method took to compute the best scoring network. Note that there is a trade off between the accuracy and time of a method for a given number of iterations. The more accurate the method, the longer it takes, the less accurate the faster.

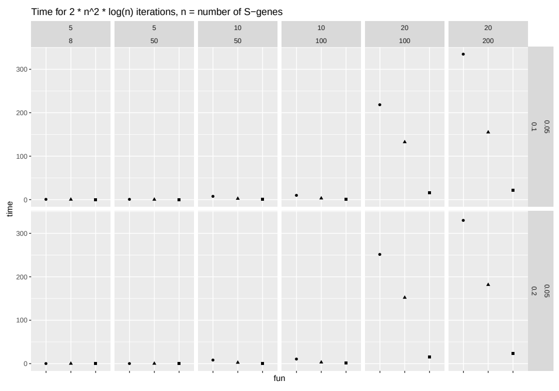


Figure 3: Time of inference methods.

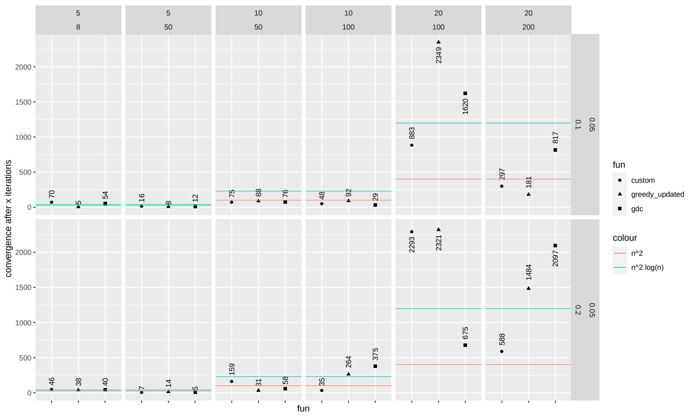
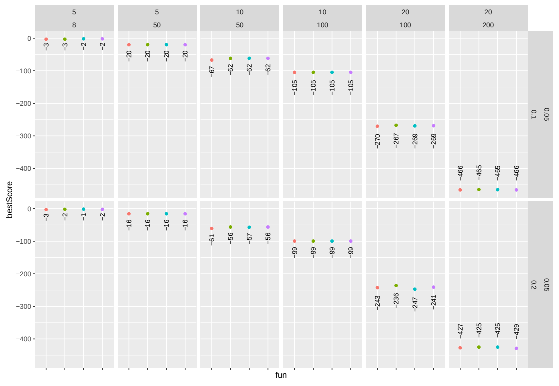
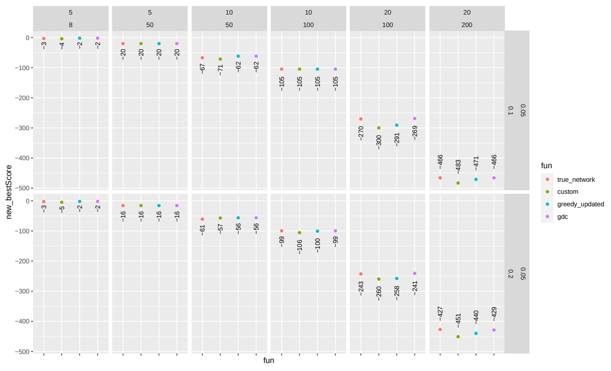


Figure 4: Convergence after x iterations.

Assume there is a fixed time, i.e. each method is only allowed to spend a certain time to do the inference, how would they compare now? For this I used the time that the *gradient* (gdc) approach took for each network and error rates and calculated how many iterations the other methods would have done in this time. The results are in Figure 5. We see that in general the log likelihood scores of the inferred networks drop for the *numerical* (custom) and *greedy* (greedy_updated) method and are lower than the likelihood scores of the true network and the *gradient* approach, especially for larger networks. Therefore, we can hypothesize that the number of iterations for each method is more important than which method is used, i.e. if we use enough iterations we explore more of our search space and the probability to find the right network is higher.



(a) likelihood score after $2nS^2\log(nS)$ iterations.



(b) likelihood score after a fixed amount of time.

Figure 5: Likelihood score on the y-axis of inferred networks (of different sizes $nS = 5, 10, 20$, $nE = 8, 50, 100, 200$ and error rates $\alpha = 0.05$ and $\beta = 0.1, 0.2$) after a fixed number of iterations or fixed amount of time. The red value of *true_network* is the likelihood score of the true network underlying the given data and serves as a reference point.

3.1.2 Gamma

Seeing that nS^2 and $nS^2\log(nS)$ are good choices we combined them with different choices of γ, i.e. $\gamma = 1, \frac{nS}{nE}, 2\frac{nS}{nE}$, to see which combination obtains the best results (see files *find_optimal_params_gamma.R*, *find_optimal_params_gamma_eval.Rmd* and *find_optimal_params_gamma_eval.html*). Again this is only based on one run of the methods on the different networks. For more robust results we should run this multiple times and on different networks of the same size or different perturbation data from a network but due to time constraints this was not possible. After comparing their performances and considering the trade-of between accuracy and time the optimal parameters are summarized in Table 5.

Method	niteration	γ
custom	nS^2	$2\frac{nS}{nE}$
gradient	$nS^2\log(nS)$	$2\frac{nS}{nE}$
greedy	$nS^2\log(nS)$	$2\frac{nS}{nE}$

Table 5: Best Hyper Parameters

3.2 Compare different Runs

We want to show that the chosen parameters lead to robust results, i.e that the number of iterations is high enough such that in every run we have approximately the same outcome (see files *compare_mcmcRuns.R*, *compare_mcmcRuns_eval.Rmd* and *compare_mcmcRuns_eval.html*). Due to time constraints this comparison was only done with a small network of size 5 with 8 and 50 E-genes and different error rates. We ran each method with the optimal hyper parameters found in the previous section ten times. In Figure 6 we can see that most of the times the methods infer the same networks with the same log likelihoods. The black horizontal lines indicated the log likelihood of the true underlying network. We see that for almost all runs we obtain similar scores, so the chosen parameters seem to give robust results.

4 Benchmark

A *snakemake* pipeline was created for this benchmark. The user has to set some parameters such as nS, the number of S-genes in a network, nE the number of E-genes in a network, *alpha*, type I error rate, β, type II error rate, and define different seeds and which inference methods to use. First, we create networks for specific nS and nE combinations. Then we create artificial perturbation data from these true networks and error rates, run the various inference methods on this perturbation data and finally calculate some evaluation metrics such as accuracy and f1 score and also plot the results. The workflow is described in more detail in the following subsections.

4.1 Data

In the first step we generate a network structure using the `sampleRndNetwork` function from the *nem* package for each nS, the number of S-genes, and *seed*.

```r
S_genes <- paste("S",1:nS,sep="")
set.seed(seed)
S_network <- sampleRndNetwork(S_genes, trans.close = TRUE, DAG = TRUE)
```

Then we generate perturbation data for each network, *seed* and nE, number of E-genes, using the `sampleData` function and given error rates *alpha* and *beta*.

```r
set.seed(seed)
data <- sampleData(Phi = S_network, m = nE, uninformative = 0, type = 'binary',
                   replicates = 1, typeI.err = alpha, typeII.err = beta)
```

For our approach we compute then the score tables.

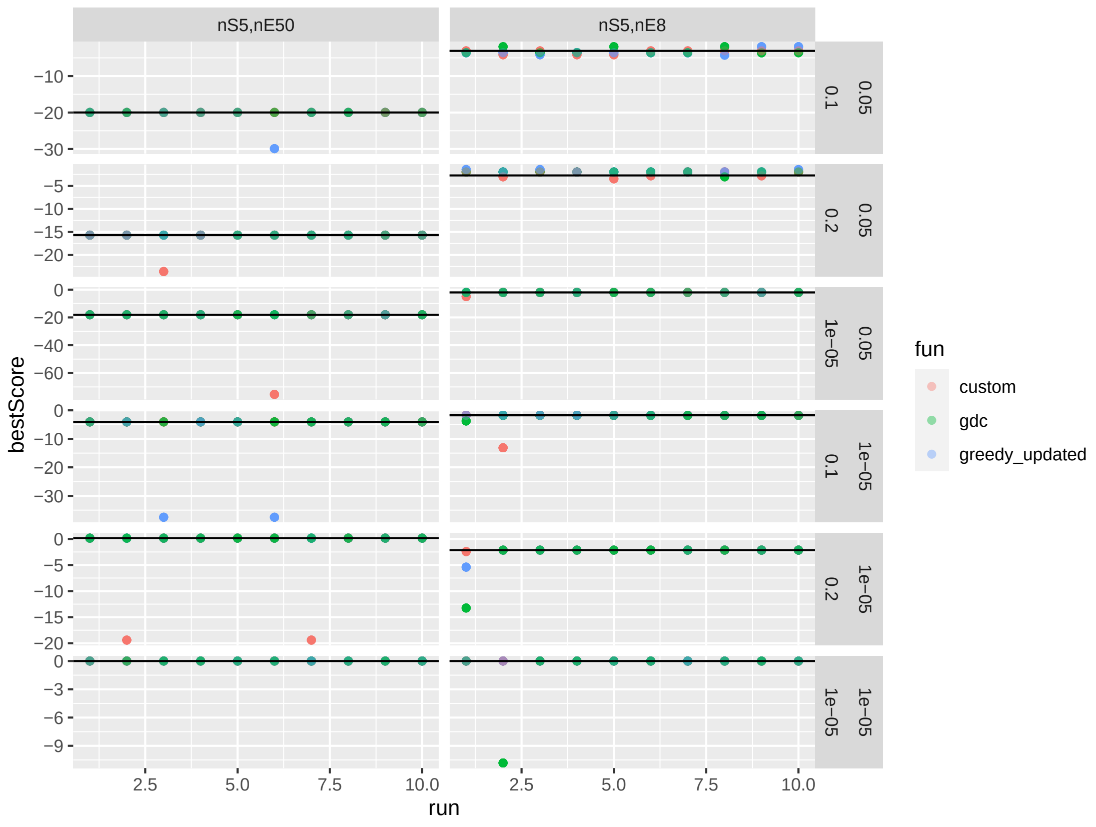


Figure 6: Summary of outcomes of ten runs of the three different approaches on networks with five S-genes and eight or fifty E-genes and different error rates. The black lines mark the log likelihood score of the true network.

4.2 Inference Methods

We compare our three approaches to three different methods from the *nem* package which can infer S-gene networks from perturbation data. As input for the variants of our methods are the score tables.

4.2.1 Custom

Here we run approach 1 described in section 2.8.1 with number of iterations being nS^2 and setting $\gamma = 2\frac{nS}{nE}$ and the score tables as input.

```
niterations <- nS^2
gamma <- 2 * nS / nE
NEMorderMCMC(startorder = permutation, allparentRs = score_tables,
             iterations = niterations, gamma = gamma, opt_fun="custom")
```

4.2.2 Gradient

Here we run approach 2 described in section 2.8.2 with number of iterations being $nS^2\log(nS)$ and setting $\gamma = 2\frac{nS}{nE}$ and the score tables as input.

```
niterations <- round(nS^2*log(nS), 0)
gamma <- 2 * nS / nE
NEMorderMCMC(startorder = permutation, allparentRs = score_tables,
             iterations = niterations, gamma = gamma, opt_fun="gdc")
```

4.2.3 Greedy

Here we run approach 3 described in section 2.8.3 with number of iterations being $nS^2 \log(nS)$ and setting $\gamma = 2\frac{nS}{nE}$ and the score tables as input.

```
niterations <- round(nS^2*log(nS), 0)
gamma <- 2 * nS / nE
NEMorderMCMC(startorder = permutation, allparentRs = score_tables,
             iterations = niterations, gamma = gamma, opt_fun="greedy_updated")
```

4.2.4 Edge-wise learning

For each pair of S-genes it scores four different two node networks. Then it combine all highest scoring pairwise relations into one network by graph merging. Consider that this approach makes the assumption that edges are independent, which is not true.

```
hyper  <- set.default.parameters(para=c(alpha, beta))
res <- nem(D = t(knockdown_data), inference = "pairwise", control = hyper)
```

4.2.5 Triplets learning

For each triplet of S-genes it scores 29 unique networks. For each edge between two S-genes we can compute a frequency, if this is higher than a given threshold then this edge is added to the network. Note that this approach is slower than pairwise learning. Furthermore, this approach does not make the assumption of independent edges and should hence be more accurate.

```
hyper  <- set.default.parameters( para=c(alpha, beta))
res<- nem(D = t(knockdown_data), inference = "triples", control = hyper)
```

4.2.6 Learning NEMs via EM and MCMC

In the description they say that this approach couples the EM Algorithm with a MCMC with the benefit of avoiding local optima. It was a bit difficult to set the right hyper-parameters (e.g. choose the appropriate *type*) because this function is not well documented and it's not well explained which *type* values you can choose and which make sense in which situation, etc. mcmc.nsamples was set to $2nS^2 \log(nS)$ similar to the number of iterations for our approach in hope for a fair comparison.

```
hyper  <- set.default.parameters( para=c(alpha, beta))
hyper$type = "CONTmLLMAP"
hyper$Pm = NULL
hyper$mcmc.nburnin = 10
hyper$mcmc.nsamples = round(2 * nS^2* log(nS),0 )
res <- nem(D = t(knockdown_data), inference = "mc.eminem", control = hyper)
```

4.3 Metrics

Once all the different inference methods outputted a network we compared their adjacency matrix to the adjacency matrix of the true underlying network and constructed a contingency table and the following evaluation metrics:

		inferred network	
		1	0
true network	1	TP	FP
	0	FN	TN

$$\text{recall} = \frac{TP}{TP + FN} \qquad \text{precision} = \frac{TP}{TP + FP}$$
$$\text{accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \qquad \text{f1 score} = \frac{2 * precision * recall}{precision + recall}$$
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{TNR} = \frac{TN}{TN + FP}$$
$$\text{balanced accuracy} = \frac{TPR + TNR}{2}$$

5 Results

The pipeline was run with the following parameters and 27 different seeds:

$$\begin{aligned} nS &= 20 \\ nE &= 50, 100, 200 \\ \alpha &= 0.05, 0.12 \\ \beta &= 0.08, 0.1 \end{aligned}$$

This means that for each combination of these parameters we run the inference methods on 27 different networks structures and perturbation data. The results are summarized in Figure 7. We observe that the MCMC method from the *nem* package performs across all combinations and in all metrics the worst. We also notice that the more E-genes, i.e. data we have, the better all methods perform in all metrics. This is also true for lower error rates. For the networks with 200 E-genes and various error rates the three variants of our approach seem to perform better than the other methods in almost all metrics. But for the networks with 50 E-genes and different error rates, the other methods from the *nem* package seem to perform better overall. For the networks with 100 E-genes the methods perform more or less equal, i.e. in some metrics our approach is stronger and in other metrics the other methods are better. But there seems to be a distinction between the different error rates. For this network with 100 E-genes and lower error rates our variants seem to work better but for higher error rates the other methods look as if they are better.

If we take a look at Figure 8 we see that pairwise learning and the MCMC method are the fastest methods and that the greedy approach seems to be the slowest among all. Between the three variants of our orderMCMC, the one with the gradient optimization is the fastest. It is even faster than triplet search but slower than pairwise learning.

6 Discussion

Out of the three variants the gradient approach seems to be a good choice for this algorithm. Even though it does not necessarily find the best fitting DAG for every order, it is very fast and can therefore be run for more iterations. And if it runs for enough iterations and explores the search space intensively, then it might be enough to find the highest scoring DAG sometimes but not every time.

This NEM order-MCMC method could be extended and improved in a couple of ways. For example instead of fixing γ to a constant we could implement it to be self-regulating using a variant of simulated annealing. Then it can adapt to where we are in the likelihood space and can help to escape local optima by enforcing more random moves. Another thing to try out would be the combination of the *gradient* and *greedy* approach. The idea is that with the *gradient* we want to search through the search space roughly and once we are in a high likelihood area we want to refine the search using the *greedy* approach. At the moment we are optimizing the parent weights in an iterative way and maybe the order in which we optimize does influence the optimization. This could be verified and one might also want to see if we can optimize the weights simultaneously. So far we have run our method on score tables from perturbation data,

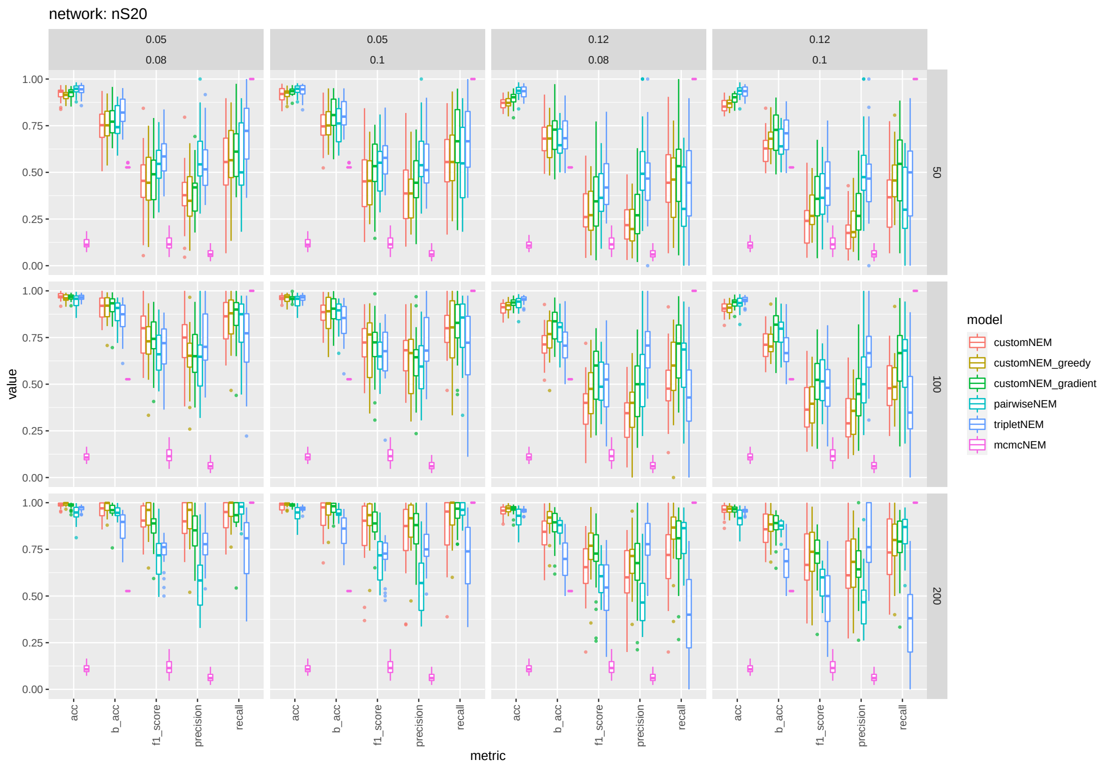


Figure 7: Evaluation of different inference methods for networks with 20 S-genes, 50,100 or 200 E-genes and different error rates.

where we have performed for each S-gene one knockout. But in theory we could also have data where S-genes have been knocked down multiple times, i.e. where we have repeats. Therefore, a way to extend this method is to extend the function to build score tables such that this can also be done for knockdown data with repeats.

Concerning the benchmark there are also a couple of things that could be done. For example instead of generating just once knockdown data from one network we could also generate different perturbation data from one network and then average over the results. Specially interesting would be to see how these methods perform with even larger networks. Potentially our method has a better trade-off between time and accuracy than the other methods, but due to time constraints this could not be explored. So far we have computed permutation data with zero uninformative E-genes. But in reality there might be E-genes that don't offer any insights, i.e. that are not attached to any S-gene in the network. These might introduce even more and different kind of errors and complexity to the data. So it would be interesting to see how these inference methods behave on different number of uninformative E-genes.

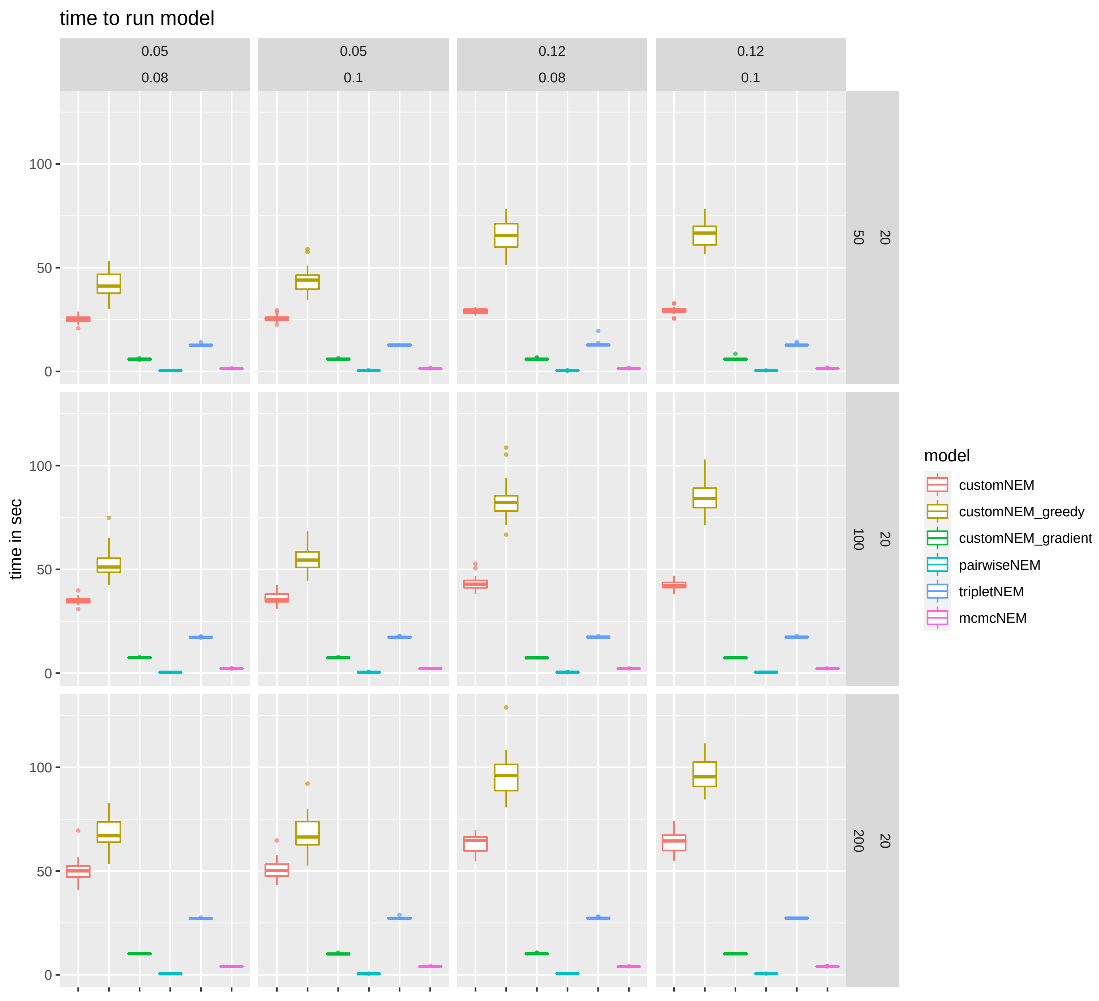


Figure 8: Time for different runs of inference methods on different network sizes and error rates.

References

[1] Florian Markowetz, Jacques Bloch, and Rainer Spang. Non-transcriptional pathway features reconstructed from secondary effects of RNA interference. *Bioinformatics*, 21(21):4026–4032, 09 2005.

[2] Florian Markowetz, Dennis Kostka, Olga G. Troyanskaya, and Rainer Spang. Nested effects models for high-dimensional phenotyping screens. *Bioinformatics*, 23(13):i305–i312, 07 2007.