— Nested Effects Models —
An example in *Drosophila* immune response

Holger Fröhlich* Florian Markowetz†

October 30, 2017

Abstract

Cellular signaling pathways, which are not modulated on a transcriptional level, cannot be directly deduced from expression profiling experiments. The situation changes, when external interventions like RNA interference or gene knock-outs come into play.

In [11, 12, 4, 5, 15, 7] algorithms were introduced to infer non-transcriptional pathway features based on differential gene expression in silencing assays. These methods are implemented in the Bioconductor package `nem`. Here we demonstrate its practical use in the context of an RNAi data set investigating the response to microbial challenge in Drosophila melanogaster.

We show in detail how the data is pre-processed and discretized, how the pathway can be reconstructed by different approaches, and how the final result can be post-processed to increase interpretability.

1 Drosophila RNAi data

We applied our method to data from a study on innate immune response in *Drosophila* [2]. Selectively removing signaling components blocked induction of all, or only parts, of the transcriptional response to LPS.

Dataset summary The dataset consists of 16 Affymetrix-microarrays: 4 replicates of control experiments without LPS and without RNAi (negative controls), 4 replicates of expression profiling after stimulation with LPS but without RNAi (positive controls), and 2 replicates each of expression

*University of Bonn, Bonn-Aachen International Center for Information Technology, frohlich@bit.uni-bonn.de

†Cancer Research UK, Florian.Markowetz@cancer.org.uk

profiling after applying LPS and silencing one of the four candidate genes *tak*, *key*, *rel*, and *mkk4/hep*.

Preprocessing and E-gene selection For preprocessing, we perform normalization on probe level using a variance stabilizing transformation (Huber *et al.*, 2002), and probe set summarization using a median polish fit of an additive model (Irizarry *et al.*, 2003). The result is included as a dataset in the package `nem`.

```
> library(nem)
> data("BoutrosRNAi2002")
```

The function `nem.discretize` implements the following two preprocessing steps: First, we select the genes as effect reporters (E-genes), which are more then two-fold upregulated by LPS treatment. Next, we transform the continuous expression data to binary values. We set an E-genes state in an RNAi experiment to `1` if its expression value is sufficiently far from the mean of the positive controls, *i.e.* if the intervention interrupted the information flow. If the E-genes expression is close to the mean of positive controls, we set its state to `0`.

Let C_{ik} be the continuous expression level of E_i in experiment k. Let μ_i^+ be the mean of positive controls for E_i, and μ_i^- the mean of negative controls. To derive binary data E_{ik}, we defined individual cutoffs for every gene E_i by:

$$E_{ik} = \begin{cases} 1 & \text{if } C_{ik} < \kappa \cdot \mu_i^+ + (1-\kappa) \cdot \mu_i^- , \\ 0 & \text{else.} \end{cases} \tag{1}$$

```
> res.disc <- nem.discretize(BoutrosRNAiExpression,neg.control=1:4,pos.control=5:8,c

discretizing with respect to POS and NEG controls
```

Estimating error probabilities From the positive and negative controls, we can estimate the error probabilities α and β. The type I error α is the number of positive controls discretized to state `1`, and the type II error β is the number of negative controls in state `0`. To guard against unrealistically low estimates we add pseudo counts. The error estimates are included into the discretization results:

```
> res.disc$para
```

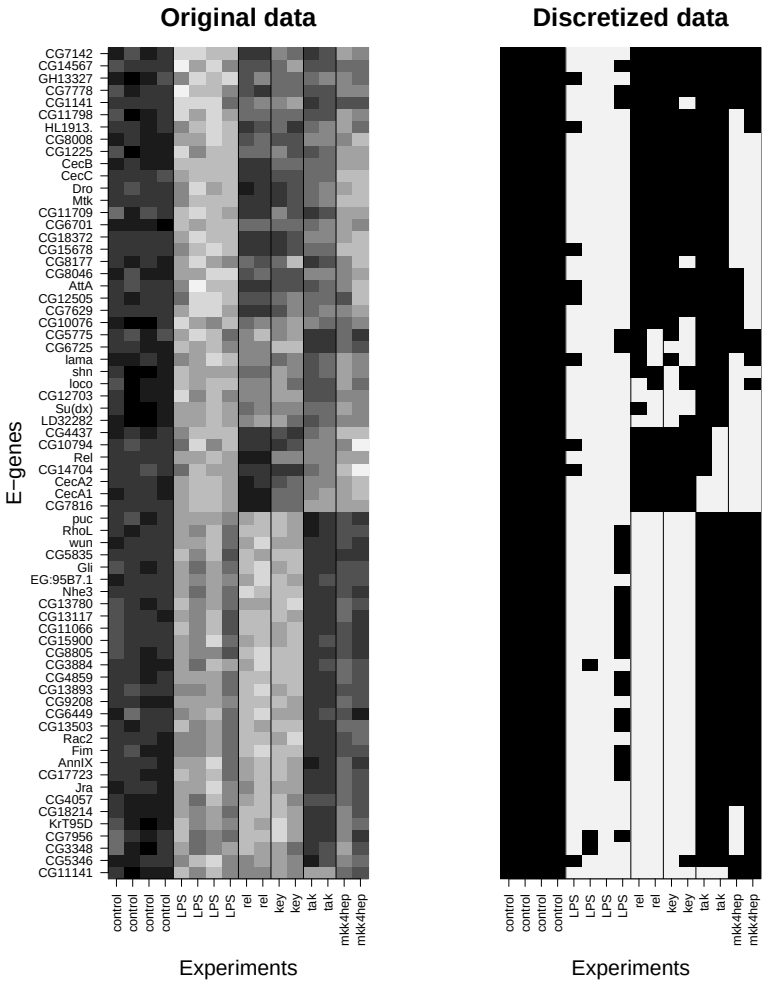

Figure 1: Continuous and discretized data

```
   a    b
0.19 0.07
```

2 Applying Nested Effects Models

Which model explains the data best? With only four S-genes, we can exhaustively enumerate all pathway models and search the whole space for the best-fitting model. To score these models use either the marginal likelihood depending on α and β (details found in Markowetz *et al.* (2005)) or the full marginal likelihood depending on hyperparameters (details in [10]).

In cases, where exhaustive search over model space is infeasible (i.e. when we have more than 4 or 5 perturbed genes) several heuristics have been developed and integrated into the *nem* package:

- edge-wise and triplets learning [12]
- greedy hillclimbing
- module networks [4, 5]
- alternating MAP optimization [15]

An interface to all inference techniques is provided by the function `nem`.

2.1 Exhaustive search by marginal likelihood

Scoring models by marginal log-likelihood is implemented in function `score`. Input consists of models, data and the type of the score (`"mLL"`, `"FULLmLL"`, `"CONTmLL"`, `"CONTmLLBayes"`, `"CONTmLLMAP"`). Furthermore, there are several additional options, which allow you to specify parameters, priors for the network structure and for the position of E-genes, and so on. The score types `"mLL"` and `"FULLmLL"` refer to discretized data, which we use in this example. The score types `"CONTmLL"`, `"CONTmLLBayes"`, `"CONTmLLMAP"` are discussed in Section 2.9. With type `"mLL"` we need to pass error probabilities *alpha* and *beta* in the argument `para`. Since version 1.5.4 all parameters are summarized in one hyperparameter, which is passed to the main function `nem`. For convenience a function `set.default.parameters` has been introduced, which allows to change specific parameters, while all others are set to default values.

```
> hyper = set.default.parameters(unique(colnames(res.disc$dat)), para=res.disc$para)
> result <- nem(res.disc$dat,inference="search",control=hyper, verbose=FALSE)
> result

scores for  355  models
--> best model is number  334
```

```
Information on this model:
Object of class  score

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  mLL
log posterior (marginal) likelihood $mLL: -240.8066
Error probabilities alpha and beta: 0.19 0.07
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
67  selected E-genes:
--> rel : 30  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

The output contains the highest scoring model (`result$graph`), a vector of scores (`result$mLL`) and a list of E-gene position posteriors (`result$pos`), and a MAP estimate of E-gene positions (`result$mappos`). We can plot the results using the commands:

```
> plot.nem(result,what="graph")
> plot.nem(result,what="mLL")
> plot.nem(result,what="pos")
```

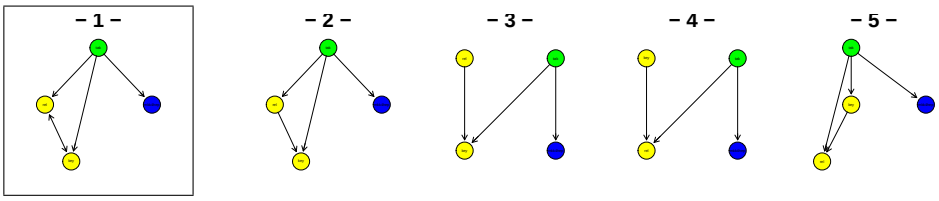

Figure 2: The five silencing schemes getting high scores in Fig. 3. It takes a second to see it, but Nr.2 to 5 are not that different from Nr.1. The main feature, ie. the branching downstream of *tak* is conserved in all of them.

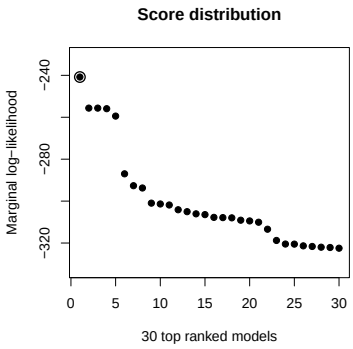

Figure 3: The best 30 scores

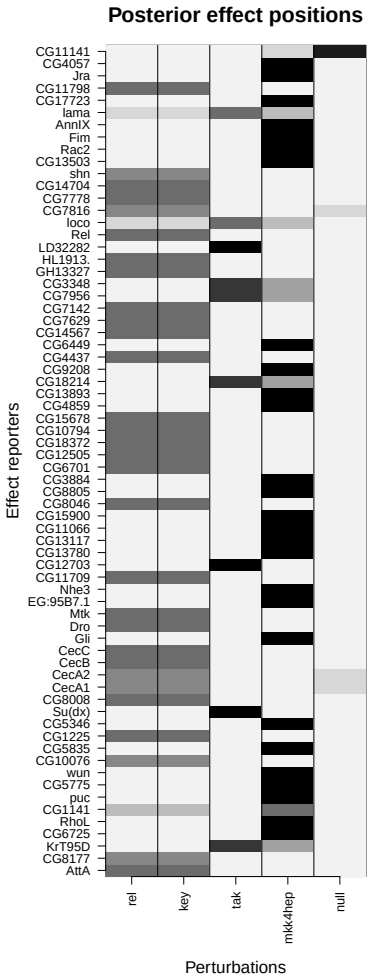

Figure 4: Posterior distributions of E-gene positions given the highest scoring silencing scheme (Nr. 1 in Fig. 2). The MAP estimate corresponds to the row-wise maximum.

2.2 Exhaustive search Full marginal likelihood

Additionally to what we did in the paper [11] the PhD thesis [10] contains equations for a "full marginal likelihood" in which error probabilities α and β are integrated out. This section shows that using this score we learn the same pathways as before.

```
> hyper$type="FULLmLL"
> hyper$hyperpara=c(1,9,9,1)
> result2 <- nem(res.disc$dat,inference="search",control=hyper,verbose=FALSE)
> result2


scores for  355  models
--> best model is number  334
Information on this model:
Object of class  score

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  FULLmLL
log posterior (marginal) likelihood $mLL: -242.925
Hyperparameters for error probability distributions: 1 9 9 1
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
67  selected E-genes:
--> rel : 30  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

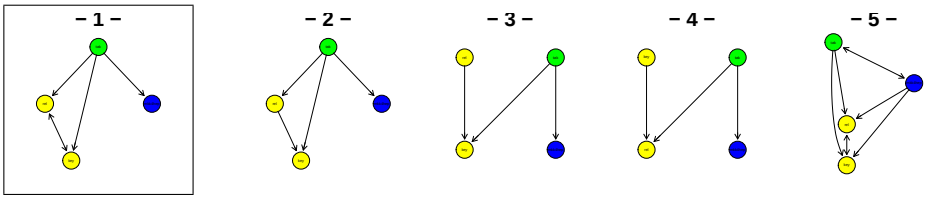

Figure 5: Same topologies as before.

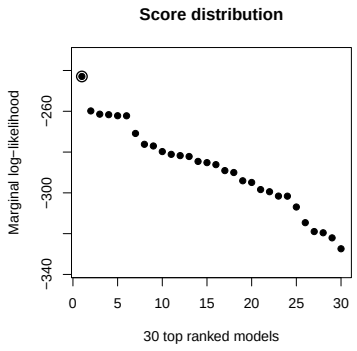

Figure 6: The best 30 scores by full marginal likelihood

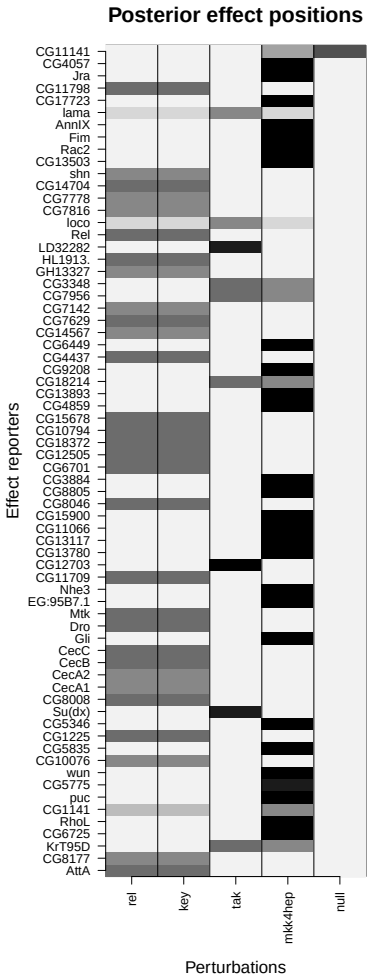

Figure 7: Posterior distributions of E-gene positions given the highest scoring silencing scheme (Nr. 1 in Fig. 5). The MAP estimate corresponds to the row-wise maximum.

2.3 Edge-wise learning

Instead of scoring whole pathways, we can learn the model edge by edge [12]. For each pair of genes A and B we infer the best of four possible models: $A \cdot \cdot B$ (unconnected), $A \rightarrow B$ (effects of A are superset of effects of B), $A \leftarrow B$ (subset), and $A \leftrightarrow B$ (undistinguishable effects).

```
> resultPairs <- nem(res.disc$dat,inference="pairwise",control=hyper, verbose=FALSE)
> resultPairs

Object of class  pairwise

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  FULLmLL
log posterior (marginal) likelihood $mLL: -260.3361
Hyperparameters for error probability distributions: 1 9 9 1
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
67  selected E-genes:
--> rel : 30  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes

$scores: posterior distributions of local models

Summary of MAP estimates:
all  ..  -> <->
  6   2   3   1
```

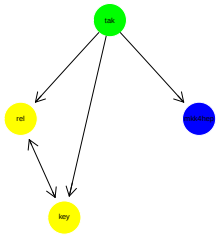

Figure 8: Result of edge-wise learning. Compare this to the result from global search. It looks exactely the same.

2.4 Inference from triples

Edge-wise learning assumes independence of edges. But this is not true in transitively closed graphs, where a direct edge must exist whenever there is a longer path between two nodes. Natural extension of edge-wise learning is inference from triples of nodes [12]. In the package `nem` we do it by

```
> resultTriples <- nem(res.disc$dat,inference="triples",control=hyper, verbose=FALSE

4 perturbed genes -> 4 triples to check (lambda =  0 )

> resultTriples

Object of class  triples

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  FULLmLL
log posterior (marginal) likelihood $mLL: -242.925
Hyperparameters for error probability distributions: 1 9 9 1
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
67  selected E-genes:
--> rel : 30  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

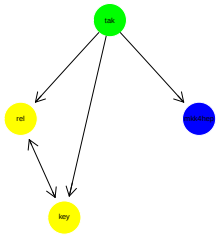

Figure 9: Result of triple learning. Compare this to the result from global search and pairwise learning

2.5 Inference with greedy hillclimbing

Greedy hillclimbing is a general search and optimization strategy known from the literature [14]. Given an initial network hypothesis (usually an empty graph) we want to arrive at a local maximum of the likelihood function by successively adding that edge, which adds the most benefit to the current network's likelihood. This procedure is continued until no improving edge can be found any more.

```
> resultGreedy <- nem(res.disc$dat,inference="nem.greedy",control=hyper, verbose=FAl

Greedy hillclimber for 4 S-genes (lambda = 0 )...

> resultGreedy

Object of class  nem.greedy

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  FULLmLL
log posterior (marginal) likelihood $mLL: -242.925
Hyperparameters for error probability distributions: 1 9 9 1
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
67  selected E-genes:
--> rel : 30  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

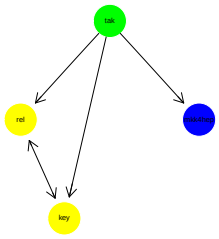

Figure 10: Result of greedy hillclimbing. It is exactly the same as for the exhaustive search.

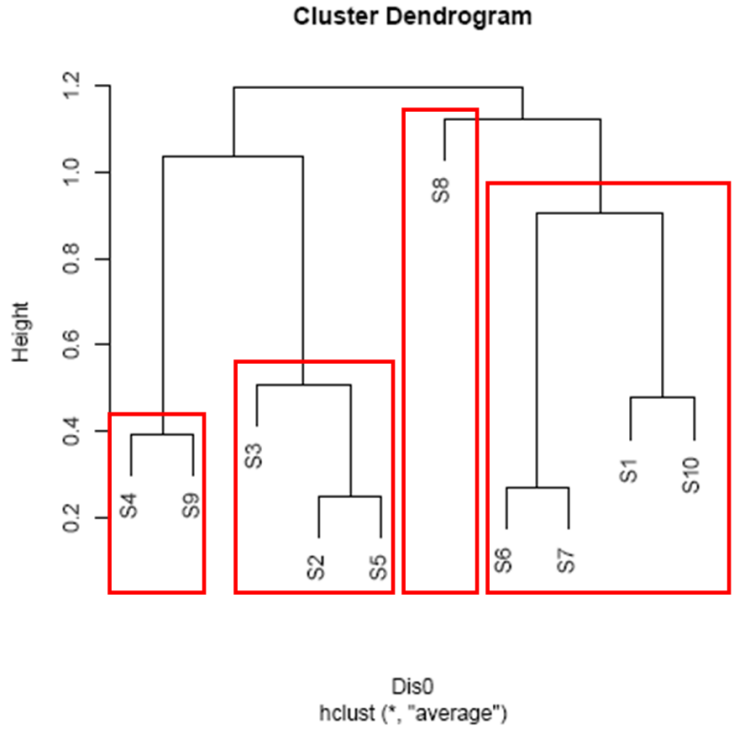

Figure 11: Basic idea of module networks: By successively moving down the cluster hierarchy we identify the clusters (modules) of S-genes, which are marked in red. They contain 4 S-genes at most and can be estimated via exhaustive search.

2.6 Inference with module networks

Rather than looking for a complete network hypothesis at once the idea of the module network is to build up a graph from smaller subgraphs, called *modules* in the following [4, 5]. The module network is thus a divide and conquer approach: We first perform a hierarchical clustering of the preprocessed expression profiles of all S-genes, e.g. via average linkage. The idea is that S-genes with a similar E-gene response profile should be close in the signaling path. We now successively move down the cluster tree hierarchy until we find a cluster with only 4 S-genes at most. Figure 11 illustrates the idea with an assumed network of 10 S-genes. At the beginning we find S_8 as a cluster singleton. Then by successively moving down the hierarchy we identify clusters S_6, S_7, S_1, S_10, S_3, S_2, S_5 and S_4, S_9. All these clusters (modules) contain 4 S-genes at most and can thus be estimated by taking the highest scoring of all possible network hypotheses.

Once all modules have been estimated their connections are constructed.

For this purpose two different approaches have been proposed:

- In the original publication [4] pairs of modules are connected ignoring the rest of the network. That means between a pair of modules all (at most 4096) connection possibilities going from nodes in the first to nodes in the second module are tested. The connection model with highest likelihood is kept. This approach does not guarantee to reach a local maximum of the complete network, but only needs $O(|S - genes|^2)$ likelihood evaluations.
- In [5] a constraint greedy hill-climber is proposed: We successively add that edge between any pair of S-genes being contained in different modules, which increases the likelihood of the complete network most. This procedure is continued until no improvement can be gained any more, i.e. we have reached a local maximum of the likelihood function for the complete network.

In the package `nem` we call the module network by

```
> resultMN <- nem(res.disc$dat,inference="ModuleNetwork",control=hyper, verbose=FALS
> resultMN.orig <- nem(res.disc$dat,inference="ModuleNetwork.orig",control=hyper, ve
```

2.7 Alternating optimization

The alternating optimization scheme was proposed in [15]. In contrast to the original approach by Markowetz et al. [11] there is a MAP estimate of the linking positions of E-genes to S-genes. The algorithm works as follows: Starting with an initial estimate of the linking of E-genes to S-genes from the data, we perform an alternating MAP optimization of the S-genes graph and the linking graph until convergence. As a final step we find a transitively closed graph most similar to the one resulting from the alternating optimization.

```
> resGreedyMAP <- nem(BoutrosRNAiLods, inference="nem.greedyMAP", control=hyper, ver

Alternating optimization for 4 S-genes (lambda = 0 )...

> resGreedyMAP

Object of class  nem.greedyMAP

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  CONTmLLMAP
log posterior (marginal) likelihood $mLL: 254.5843
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
37  selected E-genes:
--> rel : 17  attached E-genes
--> key : 17  attached E-genes
--> tak : 20  attached E-genes
--> mkk4hep : 0  attached E-genes
--> null : 31  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

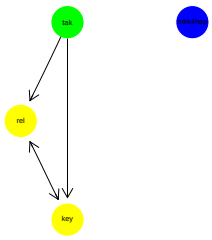

Figure 12: Result of the alternating MAP optimization.

2.8 Incorporating prior Assumptions

2.8.1 Regularization

The `nem` package allows to specify a prior on the network structure itself. This can be thought of biasing the score of possible network hypotheses towards prior knowledge. It is crucial to understand that in principle in any inference scheme there exist two competing goals: Belief in prior assumptions / prior knowledge versus belief in data. Only trusting the data itself may lead to overfitting, whereas only trusting in prior assumptions does not give any new information and prevents learning. Therefore, we need a trade-off between both goals via a regularization constant $\lambda > 0$, which specifies the belief in our prior assumptions. In the simplest case our assumption could be that the true network structure is sparse, i.e. there are only very few edges. More complex priors could involve separate prior edge probabilities (c.f. [4, 5]).

```
> hyper$Pm = diag(4)
> hyper$lambda = 10
> resultRegularization <- nem(res.disc$dat, inference="search", control=hyper, verbo
> resultRegularization

scores for  355  models
--> best model is number  125
Information on this model:
Object of class  score

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  FULLmLL
log posterior (marginal) likelihood $mLL: -265.664
Hyperparameters for error probability distributions: 1 9 9 1
network structure regularization parameter $lambda (default: 0): 10
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
67  selected E-genes:
--> rel : 3  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

In practice we would like to choose a λ in an automated fashion. This leads to an instance of the classical *model selection* problem (e.g. [8]) in statistical

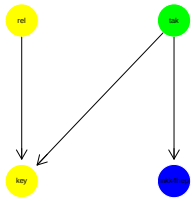

Figure 13: Result of module network learning with regularization towards sparse graph structures ($\lambda = 10$).

learning. One way of dealing with it is to tune λ such that the *Bayesian information criterion* (BIC)

$$BIC(\lambda, \Phi_{opt}) = -2 \log P(D|\Phi_{opt}) + \log(n) d(\lambda, \Phi_{opt}) \tag{2}$$

becomes minimal [8]. Here $d(\lambda, \Phi_{opt})$ denotes the number of parameters in the highest scoring network structure Φ_{opt} haven n S-genes. Here the number of parameters is estimated as:

$$d(\lambda, \Phi) = \sum_{i,j} 1_{|\Phi_{ij} - \hat{\Phi}_{ij}| > 0} \tag{3}$$

where $\hat{\Phi}$ is the prior network.

```
> resultModsel <- nemModelSelection(c(0.01,1,100),res.disc$dat, control=hyper,verbos

Greedy hillclimber for 4 S-genes (lambda = 0.01 )...

Greedy hillclimber for 4 S-genes (lambda = 1 )...

Greedy hillclimber for 4 S-genes (lambda = 100 )...
```

2.8.2 Bayesian Model Selection

Searching for an optimal regularization constant relates to a frequentistic point of view to incorporate prior knowledge. Instead, from a Bayesian perspective one should define a prior on the regularization parameter and integrate it out. Here, this is done by assuming an inverse gamma distribution prior on $\nu = \frac{1}{2\lambda}$ with hyperparameters $1, 0.5$, which leads to a simple

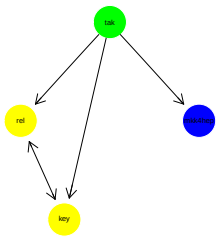

Figure 14: Result of module network learning with regularization towards sparse graph structures and automatic model selection.

closed form of the full prior [5]. An advantage of the Bayesian perspective is that no explicit model selection step is needed. Furthermore, there is evidence, that compared to the frequentistic method the Bayesian approach using the same amount of prior knowledge yields a higher increase of the reconstructed network's sensitivity

```
> hyper$lambda=0        # set regularization parameter to 0
> hyper$Pm        # this is our prior graph structure

     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

> resultBayes <- nem(res.disc$dat, control=hyper, verbose=FALSE) # now we use Bayesi

Greedy hillclimber for 4 S-genes (lambda = 0 )...

> resultBayes

Object of class  nem.greedy

$graph:  phenotypic hierarchy (graphNEL object) with genes
Inference scheme:  FULLmLL
log posterior (marginal) likelihood $mLL: -260.3361
Hyperparameters for error probability distributions: 1 9 9 1
network structure regularization parameter $lambda (default: 0): 0
Prior weight $delta for assigning E-genes to virtual S-gene 'null' (default: 1): 1
```

```
67  selected E-genes:
--> rel : 30  attached E-genes
--> key : 30  attached E-genes
--> tak : 9  attached E-genes
--> mkk4hep : 28  attached E-genes
--> null : 1  attached E-genes

NOTE: One E-gene can be attached to multiple S-genes
```

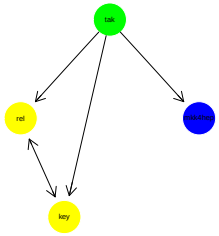

Figure 15: Result of module network learning with a Bayesian network prior favoring sparse graphs.

2.9 Omitting the Data Discretization Step

In general, performing a data discretization on the expression profiles as described in Sec. 1 can be critical, especially, if not both, positive and negative controls are available. An alternative is given by taking the raw p-value profiles obtained from testing for differential gene expression. In this situation we assume the individual p-values in the data matrix to be drawn from a mixture of a uniform, a Beta$(1, \beta)$ and a Beta$(\alpha, 1)$ distribution. The parameters of the distribution are fitted via an EM algorithm [5]. The `nem` package supports such a data format using the options `type = "CONTmLLBayes"` and `type = "CONTmLLMAP"` in the call of the function `nem`. Moreover there is a function `getDensityMatrix`, which conveniently does all the fitting of the p-value densities and produces diagnostic plots into a user specified directory. We always recommend to use the full microarray without any filtering for fitting the p-value densities, since filtering could destroy the supposed form of the p-value distributions.

An alternative to the p-value density (which we do not recommend here from our practical experience), is to use log-odds (B-values), which compare the likelihood of differential expression with the likelihood of non-differential expression of a gene. B-values can be used in the `nem` package in the same way as (log) p-value densities. The inference scheme `type = "CONTmLLBayes"` corresponds to the original one in [11], where linkage positions of E-genes to S-genes are integrated out. The inference scheme `type = "CONTmLLMAP"` was proposed in [15], where we have a MAP estimate of linkage positions.

A further possibility to omit the data discretization step is to calculate the effect probability for each gene based on given the empirical distributions of the controls. Note that for this purpose both, positive and negative controls should be available.

```
> logdensities = getDensityMatrix(myPValueMatrix,dirname="DiagnosticPlots")
> nem(logdensities[res.disc$sel,],type="CONTmLLBayes",inference="search")
> nem(logdensities[res.disc$sel,],type="CONTmLLMAP",inference="search")
> preprocessed <- nem.cont.preprocess(BoutrosRNAiExpression,neg.control=1:4,pos.cont
> nem(preprocessed$prob.influenced,type="CONTmLL",inference="search")
```

2.10 Selection of E-Genes

There exists two mechanisms to select E-genes implemented in `nem`, which can be used complementary to each other [7]:

- A-priori filtering of E-genes, which show a pattern of differential expression, which can be expected to be non-random
- Automated subset selection of most relevant E-genes within the network estimation procedure.

Automated subset selection of most relevant E-genes works as follows: A virtual "null" S-gene is used, to which E-genes are assigned, that are irrelevant. The prior probability for assigning an E-gene to the "null" S-genes is $\frac{\delta}{n}$, where $\delta > 0$ and n is the number of S-genes + 1. Since version > 2.18.0 E-gene selection is done per default. The parameter δ can optionally be tuned via the BIC model selection criterion.

For a given network hypothesis we can also check, which E-Genes had the highest positive contribution to the network hypothesis.

```
> mydat = filterEGenes(Porig, logdensities) # a-priori filtering of E-genes
> hyper$selEGenes = TRUE
> hyper$type = "CONTmLLBayes"
> resAuto = nem(mydat,control=hyper) # use filtered data to estimate a network; peri
> most.relevant = getRelevantEGenes(as(resAuto$graph, "matrix"), mydat)$selected # r
```

2.11 Statistical Stability and Significance

An important step after network inference is some kind of validation. In absence of the true network this step is rather difficult. We have implemented several possible tests, which may also be used in addition to each other [7]:

- Statistical significance: Possibly the least demand we have for our inferred network is, that it should be significantly superior to a random one. We sample N (default 1000) random networks from a null distribution and compare their marginal posterior likelihood to that of the estimated network. The fraction of how often a random network is better than the inferred one yields an exact p-value. An alternative to draw completely random networks, is to permute the node labels of the inferred network. This yields a degree distribution, which is always the same as in the inferred network.
- Bootstrapping: We wish our network to be stable against small changes in our set of E-genes. Therefore, we use bootstrapping: We sample m E-genes with replacement for N times (default 1000) and run the network inference on each bootstrap sample. At the end we count for each edge how often it was inferred in all N bootstrap runs. This yields a probability for each edge. We then may only consider edges with a probability above some threshold (e.g. 50
- Jackknife: We also wish the rest of our network to be stable against removal of S-genes. Therefore, we use the jackknife: Each S-gene is left out once and the network inference run on the other S-genes. At the end we count for each edge, how often it was inferred. This yields a jackknife probability for each edge, which then may be used to filter edges.
- Consensus models: We perform both, bootstrapping and jackknife. At the end we then only keep edges appearing more frequently than some threshold in BOTH procedures.

```
> significance=nem.calcSignificance(disc$dat,res) # assess statistical significance
> bootres=nem.bootstrap(res.disc$dat) # bootstrapping on E-genes
> jackres=nem.jackknife(res.disc$dat) # jackknife on S-genes
> consens=nem.consensus(res.disc$dat) # bootstrap & jackknife on S-genes
```

2.12 Nested Effects Models as Bayesian Networks

Recently it has been shown that under certain assumptions (e.g. acyclicity of the network graph) NEMs can be interpreted in the context of Bayesian networks [16]. This interpretation gives rise to a different formulation of NEMs, which is also accessible within the `nem` package.

```
> hyper$mode="binary_Bayesian"
> resultBN = nem(res.disc$dat, inference="BN.greedy", control=hyper)
```

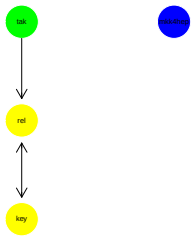

Figure 16: Result of Bayesian network learning

2.13 MC EMiNEM - Learning NEMs via Expectation Maximization and MCMC

In [13] an Expectation Maximization (EM) coupled with a Markov Chain Monte Carlo (MCMC) sampling strategy is proposed. More specifically, a MCMC algorithm (called MCEMiNEM) is used to sample from the distribution of all EM solutions, hence avoiding local optima. MCEMiNEM can be used via:

```
> hyper$mcmc.nburnin = 10 # much to few in practice
> hyper$mcmc.nsamples = 100
> hyper$type = "CONTmLLDens"
> hyper$Pm = NULL
> resulteminem = nem(BoutrosRNAiLods, inference="mc.eminem", control=hyper)
```

```
mc.eminem algorithm
```

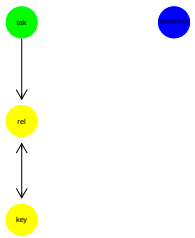

Figure 17: Result of Bayesian network learning

2.14 Dynamic Nested Effects Models

In [1] the authors describe a first extension of NEMs to time series perturbation data. That means after each perturbation gene expression is measured at several time points. The model, called D-NEMs was successfully applied to reverse engineer parts of a transcriptional network steering murine stem cell development [9]. A second model for the same purpose was proposed in [**?**]. Due to the employed likelihood model a highly efficient computation is possible here. It is thus particularly well suited for real world applications and included in to the `nem` package. In addition to the original greedy algorithm used in [3], we here provide a Markov Chain Monte Carlo (MCMC) algorithm for structure learning. It includes a Bayesian way of including prior knowledge by drawing the weight of the structure prior (parameter lambda) itself from an exponential distribution.

```
> data("Ivanova2006RNAiTimeSeries")
> dim(dat)
> control = set.default.parameters(dimnames(dat)[[3]], para=c(0.1,0.1))
> net = nem(dat, inference="dynoNEM", control=control)
> plot.nem(net, SCC=FALSE, plot.probs=TRUE)
```

2.15 Deterministic Effects Propagation Networks

Deterministic Effects Propagation Networks (DEPNs) [6] go in another direction than NEMs and they are designed for a different scenario: The basic assumption is that we measure an unknown signaling network of proteins by performing multiple interventions. These perturbations may affect single proteins at a time or may also be combinatorial, i.e. they affect several proteins at one time. For each protein in the network we monitor the effect of all interventions, typically via Reverse Phase Protein Arrays. Note that each specific intervention not only influences direct targets, but may also cause effects on downstream proteins. We explicitly assume that in one experiment all proteins are unperturbed (e.g. the cells are transfected only with transfection reagent) .

DEPNs infer the most likely protein signaling network given measurements from multiple interventions along a time course. In principle, the model works also with only one or a few time points, although accuracy gets higher with more time points. Moreover, DEPNs can deal with latent network nodes (proteins without measurements) and with missing data. Briefly, the idea is that we have an unknown network graph, where each node can have two states: perturbed or unperturbed. Furthermore, attached to each node we have experimental measurements, which are assumed to come from a Gaussian distribution. The exact form of this distribution depends on the perturbation state of the node (i.e. whether the protein is perturbed or not) and on the time point of measurement.

Given a network hypothesis we can calculate the expected direct and indirect effects of a perturbation experiment. We should repeat at this point that we assume to have one control experiment, where all nodes are known to be unperturbed. Hence, we observe measurements for each node in the network in the perturbed and the unperturbed state for each time point. With this information we can estimate the conditional Gaussian distributions attached to each node and finally the probability of observing the data under the given network hypothesis.

DEPNs are accessible within the `nem` package since version 1.5.4:

```
> data(SahinRNAi2008)
> control = set.default.parameters(setdiff(colnames(dat.normalized),"time"), map=map
> net = nem(dat.normalized, control=control) # greedy hillclimber to find most proba
```

2.16 Infering Edge Directions

It may be interesting to infer edge types (up-regulation, down-regulation) for a given nem model. For an edge a->b we look, whether b goes up or down, if a is perturbed. If it goes up, an inhibition is assumed, otherwise an activation.

```
> resEdgeInf = infer.edge.type(result, BoutrosRNAiLogFC)
> plot.nem(resEdgeInf, SCC=FALSE)
```

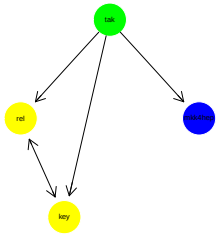

Figure 18: Inferred edge types: blue = inhibition, red = activation

3 Visualization

```
> plotEffects(res.disc$dat,result)
```

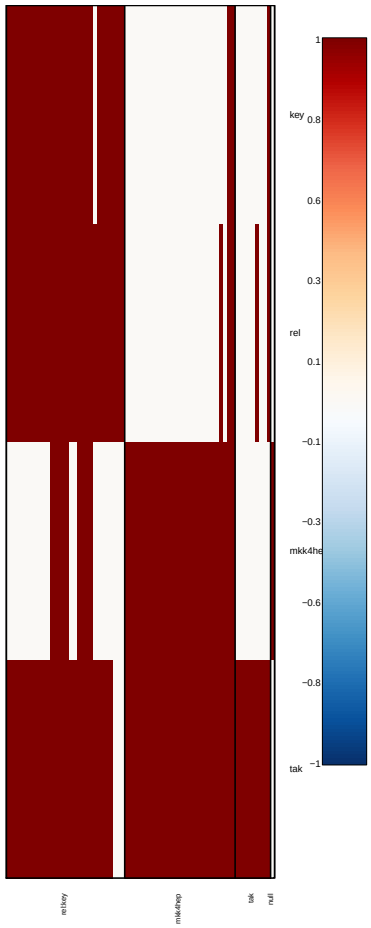

Figure 19: plotting data according to inferred hierarchy

4 Post-processing of results

Combining strongly connected components First, we identify all nodes/genes which are not distinguishable given the data. This amounts to finding the strongly connected components in the graph. Relish and Key are now combined into one node.

```
> result.scc <- SCCgraph(result$graph,name=TRUE)
> plot(result.scc$graph)
```

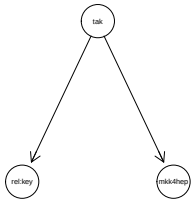

Figure 20: The undistinguishable profiles of *key* and *rel* are summarized into a single node.

Transitive reduction Additionally, in bigger graphs `transitive.reduction` helps to see the structure of the network better. In this simple example there are no shortcuts to remove.

References

[1] B. Anchang, M. J. Sadeh, J. Jacob, A. Tresch, M. O. Vlad, P. J. Oefner, and R. Spang. Modeling the temporal interplay of molecular signaling and gene expression by using dynamic nested effects models. *Proc Natl Acad Sci U S A*, 106(16):6447–6452, Apr 2009.

[2] M. Boutros, H. Agaisse, and N. Perrimon. Sequential activation of signaling pathways during innate immune responses in *Drosophila*. *Developmental Cell*, 3(5):711 – 722, 2002.

[3] H. Frï¿½hlich, P. Praveen, and A. Tresch. Fast and efficient dynamic nested effects models. *Bioinformatics*, 27(2):238–244, Jan 2011.

[4] H. Fröhlich, M. Fellmann, H. Sültmann, A. Poustka, and T. Beißbarth. Large Scale Statistical Inference of Signaling Pathways from RNAi and Microarray Data. *BMC Bioinformatics*, 8:386, 2007.

[5] H. Fröhlich, M. Fellmann, H. Sültmann, A. Poustka, and T. Beißbarth. Estimating Large Scale Signaling Networks through Nested Effect Models with Intervention Effects from Microarray Data. *Bioinformatics*, 24:2650–2656, 2008. doi: 10.1093/bioinformatics/btm634.

[6] H. Fröhlich, O. Sahin, D. Arlt, C. Bender, and T. Beissbarth. Deterministic Effects Propagation Networks for Reconstructing Protein Signaling Networks from Multiple Interventions. *BMC Bioinformatics*, 10:322, 2009.

[7] H. Fröhlich, A. Tresch, and T. Beissbarth. Nested effects models for learning signaling networks from perturbation data. *Biometrical Journal*, 2(51):304 – 323, 2009.

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, NY, USA, 2001.

[9] N. Ivanova, R. Dobrin, R. Lu, I. Kotenko, J. Levorse, C. DeCoste, X. Schafer, Y. Lun, and I. R. Lemischka. Dissecting self-renewal in stem cells with rna interference. *Nature*, 442(7102):533–538, Aug 2006.

[10] F. Markowetz. *Probabilistic Models for Gene Silencing Data*. PhD thesis, Free University Berlin, 2006.

[11] F. Markowetz, J. Bloch, and R. Spang. Non-transcriptional pathway features reconstructed from secondary effects of rna interference. *Bioinformatics*, 21(21):4026 – 4032, 2005.

[12] F. Markowetz, D. Kostka, O. Troyanskaya, and R. Spang. Nested effects models for high-dimensional phenotyping screens. *Bioinformatics*, 23:i305 – i312, 2007.

[13] T. Niederberger, S. Etzold, M. Lidschreiber, K. C. Maier, D. E. Martin, H. FrÃ¶hlich, P. Cramer, and A. Tresch. Mc eminem maps the interaction landscape of the mediator. *PLoS Comput Biol*, 8(6):e1002568, 06 2012.

[14] S. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall Inc., New Jersey, 1995.

[15] A. Tresch and F. Markowetz. Structure Learning in Nested Effects Models. *Statistical Applications in Genetics and Molecular Biology*, 7(1), 2008. in Press.

[16] C. Zeller, H. Fröhlich, and A. Tresch. A bayesian network view on nested effects models. *EURASIP Journal on Bioinformatics and Systems Biology*, 195272, 2009. In Press.

Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 3.4.2 Patched (2017-10-07 r73498), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server 2012 R2 x64 (build 9600)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, utils
- Other packages: BiocGenerics 0.24.0, Rgraphviz 2.22.0, e1071 1.6-8, graph 1.56.0, nem 2.52.0
- Loaded via a namespace (and not attached): RBGL 1.54.0, RColorBrewer 1.1-2, boot 1.3-20, class 7.3-14, compiler 3.4.2, limma 3.34.0, plotrix 3.6-6, statmod 1.4.30, stats4 3.4.2, tools 3.4.2