

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

Florian Markowetz\*      Holger Fröhlich†

November 15, 2007

**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 [6], [5], [7] and [2] we introduced an algorithm to infer non-transcriptional pathway features based on differential gene expression in silencing assays. The method is 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* [1]. 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

---

\*Lewis-Sigler Institute for Integrative Genomics, Princeton, NJ 08544, USA. eMail: [florian@genomics.princeton.edu](mailto:florian@genomics.princeton.edu); URL: <http://genomics.princeton.edu/~florian>

†German Cancer Research Center, Im Neuenheimer Feld 580, 69120 Heidelberg, Germany. eMail: [h.froehlich@dkfz-heidelberg.de](mailto:h.froehlich@dkfz-heidelberg.de)

but without RNAi (positive controls), and 2 replicates each of expression 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 than 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  $\mu_i^+$  be the mean of positive controls for  $E_i$ , and  $\mu_i^-$  the mean of negative controls. To derive binary data  $E_{ik}$ , we defined individual cutoffs for every gene  $E_i$  by:

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

```
> res.disc <- nem.discretize(BoutrosRNAiExpression, neg.control = 1:4,
+   pos.control = 5:8, cutoff = 0.7)
```

discretizing with respect to POS and NEG controls

**Estimating error probabilities** From the positive and negative controls, we can estimate the error probabilities  $\alpha$  and  $\beta$ . The type I error  $\alpha$  is the number of positive controls discretized to state 1, and the type II error  $\beta$  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:

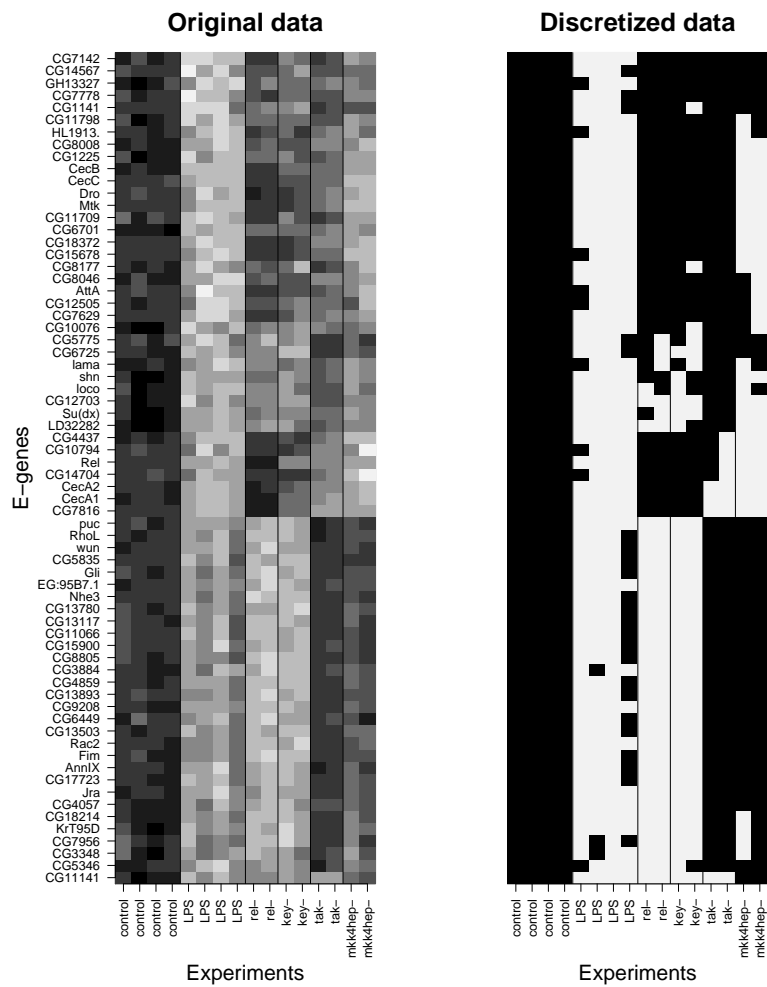


Figure 1: Continuous and discretized data

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

```
      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  $\alpha$  and  $\beta$  (details found in Markowitz *et al.* (2005)) or the full marginal likelihood depending on hyperparameters (details in Markowitz, 2006).

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 [7]
- greedy hillclimbing
- module networks [2]

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 and data, the type of the score ("mLL" or "FULLmLL"), the corresponding parameters (`para`) or hyperparameters (`hyperpara`) and a prior for E-gene positions (`P`).

```
> result <- nem(res.disc$dat, type = "mLL", para = res.disc$para,  
+             inference = "search", verbose = FALSE)  
> result
```

Object of class 'score' generated by 'score()'

```
$graph: phenotypic hierarchy on genes  
$mLL:   a vector of length 355  
$pos:   a list    of length 355  
$mappos: a vector of length 355  
$lambda: 0
```

The output is 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(result, what = "graph")
> plot(result, what = "mLL")
> plot(result, what = "pos")

```

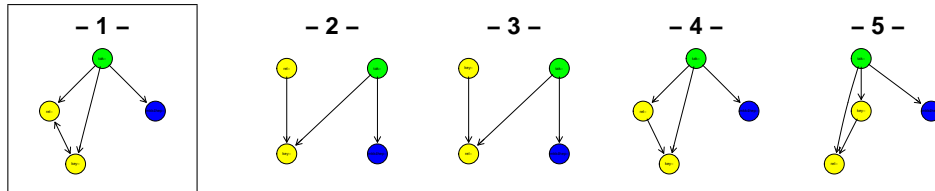


Figure 2: The five silencing schemes getting high scores in Fig. 3. It takes a second to see it, but Nr.2 to 5 are not that different from Nr.1. The main feature, ie. the branching downstream of *tak* is conserved in all of them.

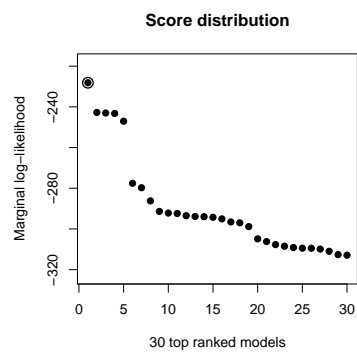


Figure 3: The best 30 scores

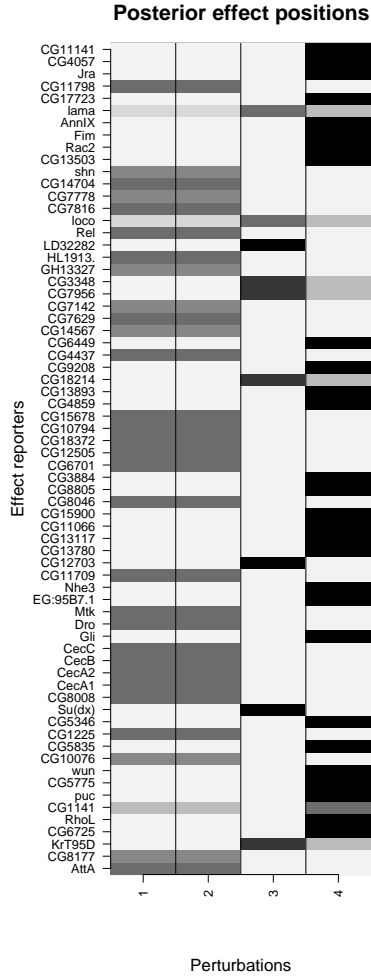


Figure 4: Posterior distributions of E-gene positions given the highest scoring silencing scheme (Nr. 1 in Fig. 2). The MAP estimate corresponds to the row-wise maximum.

## 2.2 Exhaustive search Full marginal likelihood

Additionally to what we did in the paper [6] the PhD thesis [5] contains equations for a “full marginal likelihood” in which error probabilities  $\alpha$  and  $\beta$  are integrated out. This section shows that using this score we learn the same pathways as before.

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

Object of class 'score' generated by 'score()'

```
$graph: phenotypic hierarchy on genes
$mLL: a vector of length 355
$pos: a list of length 355
$mappos: a vector of length 355
$lambda: 0
```

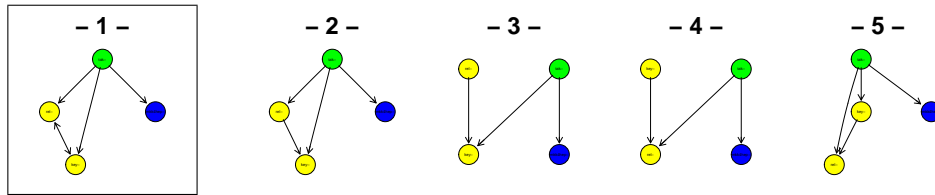


Figure 5: Same topologies as before.

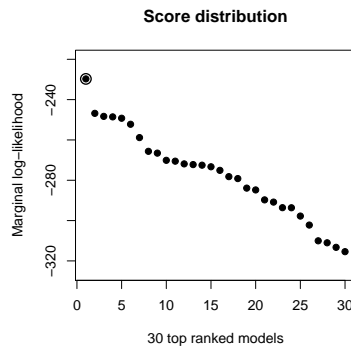


Figure 6: The best 30 scores by full marginal likelihood

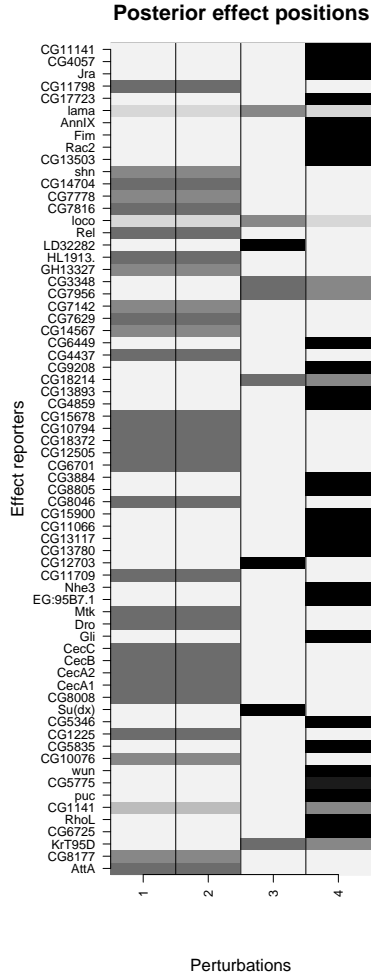


Figure 7: Posterior distributions of E-gene positions given the highest scoring silencing scheme (Nr. 1 in Fig. 5). The MAP estimate corresponds to the row-wise maximum.



## 2.3 Edge-wise learning

Instead of scoring whole pathways, we can learn the model edge by edge [7]. 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$dat, para = res.disc$para, inference = "pairwise",  
+   verbose = FALSE)  
> resultPairs
```

Object of class 'pairwise' generated by 'pairwise.posterior()'

```
$graph: phenotypic hierarchy on genes)  
$scores: posterior distributions of local models
```

Summary of MAP estimates:

```
all  ..  -> <->  
  6   2   3   1
```

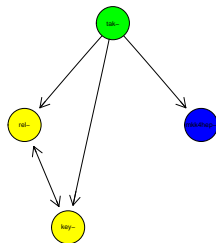


Figure 8: Result of edge-wise learning. Compare this to the result from global search. It looks exactly the same.

## 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 [7]. In the package `nem` we do it by

```
> resultTriples <- nem(res.disc$dat, para = res.disc$para, inference = "triples",  
+ verbose = FALSE)
```

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

```
> resultTriples
```

Object of class 'triples' generated by 'triples.posterior()'

\$graph: phenotypic hierarchy on genes

\$avg: matrix of edge frequencies in triple models

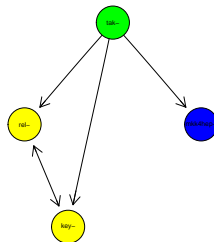


Figure 9: Result of triple learning. Compare this to the result from global search and pairwise learning

## 2.5 Inference with greedy hillclimbing

Greedy hillclimbing is a general search and optimization strategy known from the literature [8]. 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$dat, para = res.disc$para, inference = "nem.greedy",  
+   verbose = FALSE)
```

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

```
> resultGreedy
```

Object of class 'nem.greedy' generated by 'nem.greedy()'

\$graph: phenotypic hierarchy on genes)

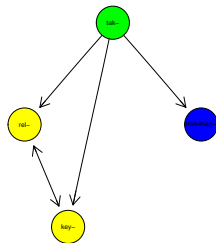


Figure 10: Result of greedy hillclimbing. It is exactly the same as for the exhaustive search.

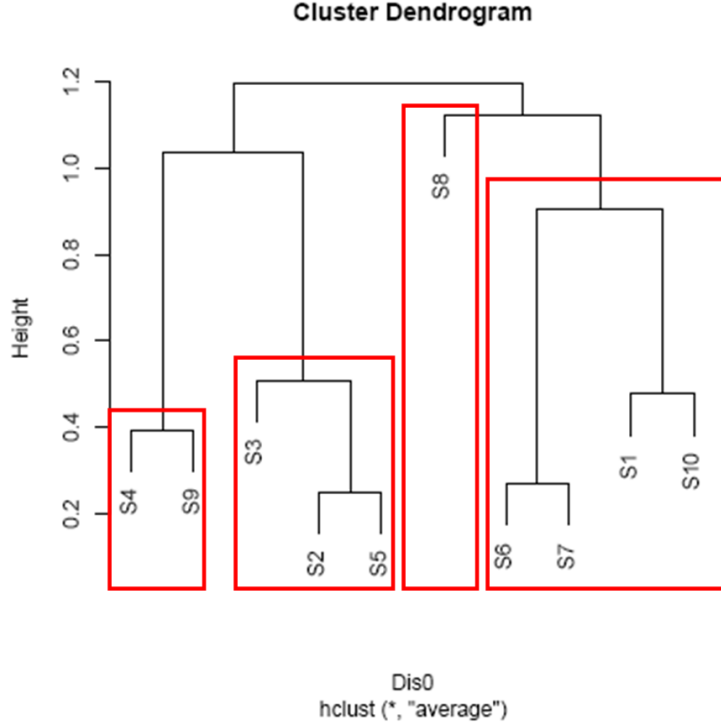


Figure 11: Basic idea of module networks: By successively moving down the cluster hierarchy we identify the clusters (modules) of S-genes, which are marked in red. They contain 4 S-genes at most and can be estimated via exhaustive search.

## 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 [3, 2]. 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 (here: with regard to the Pearson correlation similarity) 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. This is done in a constraint greedy hillclimbing fashion: 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.

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

```
> resultMN <- nem(res.disc$dat, para = res.disc$para, inference = "ModuleNetwork",  
+               verbose = FALSE)
```

Estimating module network of 4 S-genes ( $\lambda = 0$ )...

```
> resultMN
```

Object of class 'ModuleNetwork' generated by 'moduleNetwork()'

\$graph: phenotypic hierarchy on genes)

## 2.7 Incorporating prior Assumptions

### 2.7.1 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. [3]).

```
> resultRegularization <- nem(res.disc$dat, para = res.disc$para,  
+   Pm = matrix(0, ncol = 4, nrow = 4), lambda = 10, inference = "search",  
+   verbose = FALSE)  
> resultRegularization
```

Object of class 'score' generated by 'score()'

```
$graph: phenotypic hierarchy on genes  
$mLL:   a vector of length 355  
$pos:   a list   of length 355  
$mappos: a vector of length 355  
$lambda: 10
```

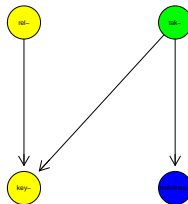


Figure 12: Result of module network learning with regularization towards sparse graph structures ( $\lambda = 10$ ).

In practice we would like to choose a  $\lambda$  in an automated fashion. This leads to an instance of the classical *model selection* problem (e.g. [4]) in statistical learning. One way of dealing with it is to tune  $\lambda$  such that the *Akaike information criterion* (AIC)

$$AIC(\lambda, \Phi_{opt}) = -2 \log P(D|\Phi_{opt}) + 2d(\lambda, \Phi_{opt}) \quad (2)$$

becomes minimal [4]. Here  $d(\lambda, \Phi_{opt})$  denotes the number of parameters (i.e. the number of unknown edges) in the highest scoring network structure  $\Phi_{opt}$ .

```
> resultModsel <- nemModelSelection(c(0.01, 1, 100), res.disc$dat,
+   para = res.disc$para, Pm = matrix(0, ncol = 4, nrow = 4),
+   verbose = FALSE)
```

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 )...

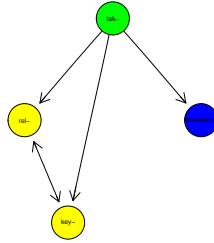


Figure 13: Result of module network learning with regularization towards sparse graph structures and automatic model selection.

### 2.7.2 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 closed form of the full prior [2]. 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

```
> resultBayes <- nem(res.disc$dat, para = res.disc$para, Pm = matrix(0,  
+      ncol = 4, nrow = 4), verbose = FALSE)
```

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

```
> resultBayes
```

Object of class 'nem.greedy' generated by 'nem.greedy()'

\$graph: phenotypic hierarchy on genes)

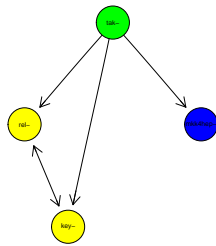


Figure 14: Result of module network learning with a Bayesian network prior favoring sparse graphs.



## 2.8 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  $\text{Beta}(1, \beta)$  and a  $\text{Beta}(\alpha, 1)$  distribution. The parameters of the distribution are fitted via an EM algorithm [2]. The `nem` package supports such a data format using the option `type = "CONTmLLDens"` in the call of the function `nem`. Moreover there is a function `getDensityMatrix`, which conveniently does all the fitting of the  $p$ -value profiles and produces diagnostic plots into a user specified directory.

A second alternative to the  $p$ -value density, is to use the log-odds (B-values), which compare the likelihood of differential expression with the likelihood of non-differential expression of a gene. The `nem` package supports such a data format using the option `type = "CONTmLLRatio"` in the call of the function `nem`.

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$sel, ], type = "CONTmLLDens", inference = "search")
> preprocessed <- nem.cont.preprocess(BoutrosRNAiExpression, neg.control = 1:4,
+   pos.control = 5:8)
> nem(preprocessed$prob.influenced, type = "CONTmLL", inference = "search")
```

## 2.9 Selection of E-Genes

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

- A-priori filtering of E-genes, which show a pattern of differential expression, which can be expected to be non-random
- Selection of most relevant E-genes for a given network hypothesis.

In case of the log-odds inference scheme (`type = "CONTmLLRatio"`) 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 of assigning an E-gene to the "null" S-genes is  $\frac{\delta}{n}$ , where  $\delta > 0$  and  $n$  is the number of S-genes. The parameter  $\delta$  is tuned via the AIC model selection criterion.

In all other cases selection of relevant E-genes is performed via an alternating optimization scheme: First, the network with the highest likelihood for the given set of E-genes is determined. Then, the E-genes with the highest marginal likelihood under that network hypothesis are calculated. In case of the  $p$ -value profile inference scheme (`type = "CONTmLLDens"`) these are simply all E-genes with a positive contribution to the log marginal likelihood. In all other cases by default the top  $10n$  E-genes are selected. With the new set of E-genes a new network is inferred and the whole procedure is repeated until convergence.

```
> mydat = filterEGenes(Porig, logdensities)
> resAuto = nem(mydat, type = "CONTmLLDens", selEGenes = TRUE)
```

## 2.10 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:

- 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 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.
- 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 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.

```
> significance = nem.calcSignificance(res, disc$dat, as(resultGreedy$graph,  
+   "matrix"), resultGreedy$mLL, para = res.disc$para)  
> bootres = nem.bootstrap(res.disc$dat, para = res.disc$para)  
> jackres = nem.jackknife(res.disc$dat, para = res.disc$para)
```

### 3 Visualization

```
> plot.effects(res.disc$dat, result)
```

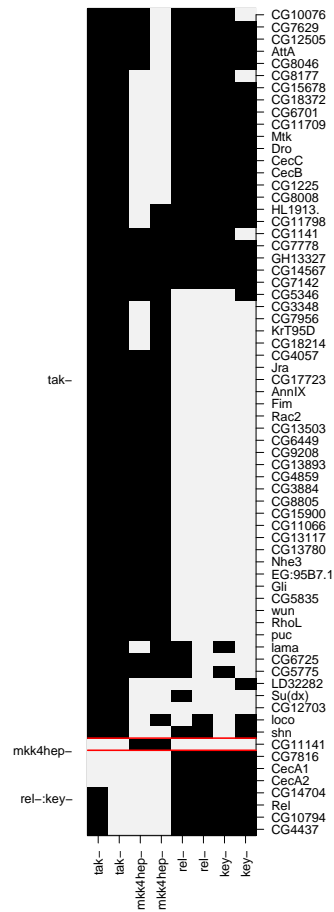


Figure 15: plotting data according to inferred hierarchy

## 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)
```

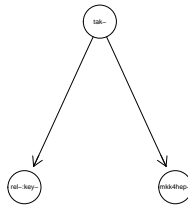


Figure 16: The undistinguishable profiles of *key* and *rel* are summarized into a single node.

**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] 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.
- [2] H. Fröhlich, M. Fellmann, H. Sülthmann, A. Poustka, and T. Beißbarth. Estimating large scale signaling networks through nested effects models from intervention effects in microarray data. In *Proc. German Conf. on Bioinformatics*, pages 45 – 54, 2007.
- [3] H. Fröhlich, M. Fellmann, H. Sülthmann, A. Poustka, and T. Beißbarth. Large scale statistical inference of signaling pathways from rna and microarray data. *BMC Bioinformatics*, 8:386, 2007.
- [4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.
- [5] F. Markowetz. *Probabilistic Models for Gene Silencing Data*. PhD thesis, Free University Berlin, 2006.
- [6] F. Markowetz, J. Bloch, and R. Spang. Non-transcriptional pathway features reconstructed from secondary effects of rna interference. *Bioinformatics*, 21(21):4026 – 4032, 2005.
- [7] F. Markowetz, D. Kostka, O. Troyanskaya, and R. Spang. Nested effects models for high-dimensional phenotyping screens. *Bioinformatics*, 23:i305 – i312, 2007.
- [8] S. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall Inc., New Jersey, 1995.

## Session Information

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

- R version 2.6.0 (2007-10-03), x86\_64-unknown-linux-gnu
- Locale: LC\_CTYPE=de\_DE@euro;LC\_NUMERIC=C;LC\_TIME=de\_DE@euro;LC\_COLLATE=de\_DE@euro
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: bootstrap 1.0-21, class 7.2-37, cluster 1.11.9, e1071 1.5-16, graph 1.16.1, nem 2.2.0, RBGL 1.14.0, Rgraphviz 1.16.0, time 1.0