Boolean Nested Effects Models
Inferring the logical signalling of pathways from indirect measurements and perturbation biology

Martin Pirkl

November 9, 2016

Boolean Nested effects Models (B-NEM) are used to infer signalling pathways. In different experiments (conditions) members of a pathway (S-genes) are stimulated or inhibited, alone and in combination. In each experiment transcriptional targets (E-genes) of the pathway react differently and are higher or lower expressed depending on the condition. From these differential expression B-NEM infers Boolean functions presented as hyper-edges of a hyper-graph connecting parents and children in the pathway. For example if the signal is transducted by two parents A and B to a child C and the signal can be blocked with a knock-down of either one, they are connected by a typical AND-gate. If the signal is still transduced during a single knock-down, but blocked by the double knock-down of A and B, they activate C by an OR-gate. In general the state of child C is defined by a Boolean function

$$f : \{0,1\}^n \rightarrow \{0,1\}\,,\; C = f(A_1, \ldots, A_n)$$

with its parents $A_i, i \in \{1, \ldots, n\}$.

The B-NEM package is based on and uses many low level function of the CellNOptR package of Terfve et al., 2012.

Loading B-NEM

```
## install.packages("devtools")

library(devtools)

install_github("MartinFXP/B-NEM", quiet = T)

library(bnem)

## Loading required package:  CellNOptR
## Loading required package:  RBGL
## Loading required package:  graph
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ, clusterExport,
##    clusterMap, parApply, parCapply, parLapply, parLapplyLB, parRapply,
##    parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##    IQR, mad, xtabs
## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append, as.data.frame,
##    cbind, colnames, do.call, duplicated, eval, evalq, get, grep, grepl,
##    intersect, is.unsorted, lapply, lengths, mapply, match, mget, order, paste,
##    pmax, pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply, setdiff,
##    sort, table, tapply, union, unique, unsplit
## Loading required package: hash
## hash-2.2.6 provided by Decision Patterns
## Loading required package: ggplot2
## Loading required package: RCurl
## Loading required package: bitops
## Loading required package: Rgraphviz
## Loading required package: grid
## Loading required package: XML
##
## Attaching package: 'XML'
## The following object is masked from 'package:graph':
##
##    addNode
## Loading required package: nem
##
## Attaching package: 'nem'
## The following object is masked from 'package:RBGL':
##
##    transitive.closure
## Loading required package: matrixStats
## matrixStats v0.51.0 (2016-10-08) successfully loaded.  See ?matrixStats for help.
## Loading required package: snowfall
## Loading required package: snow
##
## Attaching package: 'snow'
## The following objects are masked from 'package:BiocGenerics':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ, clusterExport,
##    clusterMap, clusterSplit, parApply, parCapply, parLapply, parRapply,
##    parSapply
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ, clusterExport,
##    clusterMap, clusterSplit, makeCluster, parApply, parCapply, parLapply,
##    parRapply, parSapply, splitIndices, stopCluster
## Loading required package: latticeExtra
## Loading required package: lattice
## Loading required package: RColorBrewer
##
## Attaching package: 'latticeExtra'
## The following object is masked from 'package:ggplot2':
##
##    layer
```

Toy example for DAG

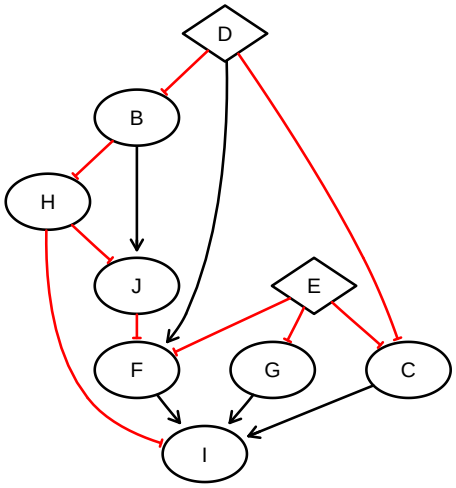
We show how to use B-NEM on a toy pathway presented by a directed acyclic (hyper-)graph (DAG). B-NEM demands several objects as input. The two main objects are the differential gene expression (data) and prior knowledge respectively the search space.

First we create a prior knowledge network (PKN). The PKN will have two S-genes without parents. We define these as stimuli, which means they are set to 1 in experiments, where they are stimulated and 0 otherwise. All other down-stream S-genes have the potential to be inhibited (0). If they are not inhibited their state is calculated according to their parents' state and a given Boolean function.

```
set.seed(2579)
## alternative seed and also a great song:
## set.seed(9247)

## to get the while loop started,
## which makes sure we get a PKN with exactly two stimuli (or what the amount you want):
stimuli <- "dummy"

while(length(stimuli) != 2) {

## random Boolean graph without cycles,
## maximal 25 edges, maximal edge size of 1 (normal graph) and maximal 10 S-genes:
  dnf <- randomDnf(10, max.edges = 25, max.edge.size = 1, dag = T)

## all S-genes:
  cues <- sort(unique(gsub("!", "", unlist(strsplit(unlist(strsplit(dnf, "=")), "\\+")))))

## parents:
  inputs <- unique(unlist(strsplit(gsub("!", "", gsub("=.*", "", dnf)), "=")))

## children:
  outputs <- unique(gsub(".*=", "", dnf))

## parents which are no children are stimuli:
  stimuli <- inputs[which(!(inputs %in% outputs))]

}

inhibitors <- unique(c(inputs, outputs))
## S-genes which are no stimuli are inhibitors
inhibitors <- inhibitors[-which(inhibitors %in% stimuli)]
```

The following figure shows the PKN. Red "tee" arrows depict repression the others activation of the child. The stimulated S-genes are diamond shaped.

```
plotDnf(dnf, stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 9
## Number of Edges = 14
```



In the next step we convert the dnf object into a PKN object and extend it to a Boolean hyper-graph, which represents an apriori restricted search space of Boolean networks.

```
sifMatrix <- NULL

for (i in dnf) {
  inputs2 <- unique(unlist(strsplit(gsub(".*", "", i), "=")))
  output <- unique(gsub(".*=", "", i))
  for (j in inputs2) {
    j2 <- gsub("!", "", j)
    if (j %in% j2) {
      sifMatrix <- rbind(sifMatrix, c(j, 1, output))
    } else {
      sifMatrix <- rbind(sifMatrix, c(j2, -1, output))
    }
  }
}
write.table(sifMatrix, file = "temp.sif", sep = "\t",
            row.names = FALSE, col.names = FALSE, quote = FALSE)
PKN <- readSIF("temp.sif")
## unlink("temp.sif")

## create metainformation (which S-genes are perturbed in which experiments):
CNOlist <- dummyCNOlist(stimuli = stimuli, inhibitors = inhibitors,
                        maxStim = 2, maxInhibit = 1, signals = NULL)

## extend the model:
model <- preprocessing(CNOlist, PKN, maxInputsPerGate=100, verbose = T)

## [1] "The following species are measured: B, C, D, E, F, G, H, I, J"
## [1] "The following species are stimulated: D, E"
## [1] "The following species are inhibited: B, C, F, G, H, I, J"
## [1] "The following species are not observable and/or not controllable: "
```

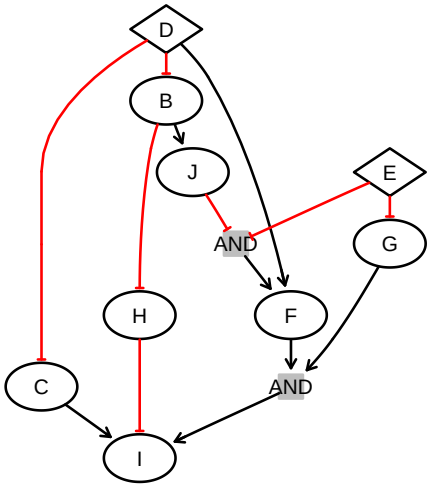
We suggest to take a look at the sif file. In future analyses it is easier to just provide a suitable sif file for the investigated pathway.

In an real application the underlaying ground truth network (GTN) is not known. However in our toy example we define one.

```
## define a bitstring denoting the present and absent edges:
bString <- absorption(sample(c(0,1), length(model$reacID), replace = T), model)

## simulate S-gene states for all possible conditions:
steadyState <- steadyState2 <- simulateStatesRecursive(CNOlist, model, bString)

## we find constitutively active S-genes with the following
## (they are boring, so we avoid them):
ind <- grep(paste(inhibitors, collapse = "|"), colnames(steadyState2))
steadyState2[, ind] <- steadyState2[, ind] + CNOlist@inhibitors

## this while loop makes sure we get a gtn,
## which actually affects all vertices
## and no vertices are constitutively active:
while(any(apply(steadyState, 2, sd) == 0) | any(apply(steadyState2, 2, sd) == 0)) {

  bString <- absorption(sample(c(0,1), length(model$reacID), replace = T), model)

  steadyState <- steadyState2 <- simulateStatesRecursive(CNOlist, model, bString)

  ind <- grep(paste(inhibitors, collapse = "|"), colnames(steadyState2))
  steadyState2[, ind] <- steadyState2[, ind] + CNOlist@inhibitors

}
```

The following figure shows our random GTN, which is a subset of the hyper-graph representing the apriori restricted search space.

```
plotDnf(model$reacID[as.logical(bString)], stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 11
## Number of Edges = 14
```



We use this GTN to simulate data.

```
## the expression data with 10 E-genes for each S-gene and 3 replicates:
exprs <- t(steadyState)[rep(1:ncol(steadyState), 10), rep(1:nrow(steadyState), 3)]
```

```
## we calculate the foldchanges (expected S-gene response scheme, ERS)
## between certain condtions (e.g. control vs stimulation):
ERS <- computeFc(CNOlist, t(steadyState))

# the next step reduces the ERS to a sensible set of comparisons;
## e.g. we do not want to compare stimuli vs inhibition,
## but stimuli vs (stimuli,inhibition):
stimcomb <- apply(expand.grid(stimuli, stimuli), c(1,2), as.character)
stimuli.pairs <- apply(stimcomb, 1, paste, collapse = "_")

## this is the usual setup,
## but "computeFc" calculates a lot more contrasts, which can also be used, if preferred:
ind <- grep(paste(c(paste("Ctrl_vs_", c(stimuli, inhibitors), sep = ""),
                    paste(stimuli, "_vs_", stimuli, "_",
                          rep(inhibitors, each = length(stimuli)), sep = ""),
                    paste(stimuli.pairs, "_vs_", stimuli.pairs, "_",
                          rep(inhibitors, each = length(stimuli.pairs)), sep = "")),
                  collapse = "|"), colnames(ERS))
ERS <- ERS[, ind]

## same as before with the expression values, we have 10 E-genes each and 3 replicates:
fc <- ERS[rep(1:nrow(ERS), 10), rep(1:ncol(ERS), 3)]

## we add some Gaussian noise:
fc <- fc + rnorm(length(fc), 0, 1)

## in real applications some E-genes are negatively regulated,
## hence we flip some foldchanges:
flip <- sample(1:nrow(fc), floor(0.33*row(fc)))
fc[flip, ] <- fc[flip, ]*(-1)

## don't forget to set rownames (usually gene symbols, ensemble ids, entrez ids, ...)
rownames(fc) <- paste(rownames(fc), 1:nrow(fc), sep = "_")
print(fc[1:6, c(1:3,(ncol(fc)-2):ncol(fc))])

##       Ctrl_vs_F  Ctrl_vs_G   Ctrl_vs_J D_E_vs_D_E_H D_E_vs_D_E_C D_E_vs_D_E_I
## B_1  0.57024784  0.2421169 -0.71445606   -0.9938059   -1.6165289   -1.4213850
## C_2 -1.70109567  1.0961857 -0.09249841   -0.1175367   -1.4843078   -1.6976438
## D_3 -1.14748014 -1.7113624  1.38419998   -0.3473269   -0.3694678   -0.8627585
## E_4  1.17990384  0.6426374 -0.24184371    0.5973451   -0.1545769   -0.4779243
## F_5 -1.19006930 -1.1619101  1.85210676    0.1237633   -1.8796660   -0.6152921
## G_6  0.07252035 -0.6971871  1.24181503   -1.8159441    0.5004976   -1.4542246
```

B-NEM uses differential expression between experiments to infer the pathway logics. For example look at the colnames of fc (=foldchanges of E-genes (rows)) and remember that D, E are our stimulated S-genes and the rest possibly inhibited. Thus in the first column of fc we have the contrast F − control. In the control no S-genes are perturbed.

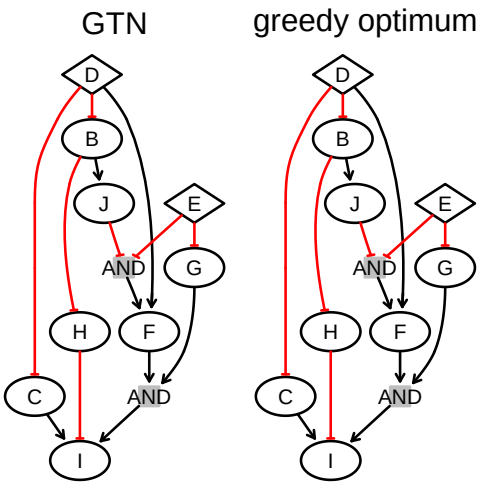
We search for the GTN in our restricted network space. Each network is a sub-graph of the full hyper-graph model$reacID. We initialise the search with a starting network and greedily search the neighbourhood.

```
## we start with the empty graph:
initBstring <- reduceGraph(rep(0, length(model$reacID)), model, CNOlist)
## or a fully connected graph:
```

```
## initBstring <- reduceGraph(rep(1, length(model$reacID)), model, CNOlist)

## paralellize for several threads on one machine or multiple machines
## see package "snowfall" for details
parallel <- 2 # NULL for serialization
## or distribute to 30 threads on four different machines:
## parallel <- list(c(4,16,8,2), c("machine1", "machine2", "machine3", "machine4"))

## greedy search:
greedy <- bnem(search = "greedy",
            fc=fc,
            exprs=exprs, # not used, if fc is defined
            CNOlist=CNOlist,
            model=model,
            parallel=parallel,
            initBstring=initBstring,
            draw = FALSE, # TRUE: draw network at each step
            verbose = FALSE, # TRUE: print changed (hyper-)edges and score improvement
            maxSteps = Inf
            )

## R Version:  R version 3.3.1 (2016-06-21)

## snowfall 1.84-6.1 initialized (using snow 0.4-2):  parallel execution on 2 CPUs.

## Library CellNOptR loaded.

## Library CellNOptR loaded in cluster.

## Library bnem loaded.

## Library bnem loaded in cluster.
##
## Stopping cluster

resString <- greedy$bString
```

We take a look at the efficiency of the search algorithm with sensitivity and specificity of the hyper-eges for the optimized network and the accuracy of its ERS (similar to the truth table). Since several networks produce the same ERS, the found hyper-graph can differ from the GTN and still be 100% accurate.

```
par(mfrow=c(1,2))

## GTN:
plotDnf(model$reacID[as.logical(bString)], main = "GTN", stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 11
## Number of Edges = 14

## greedy optimum:
plotDnf(model$reacID[as.logical(resString)], main = "greedy optimum", stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 11
## Number of Edges = 14
```



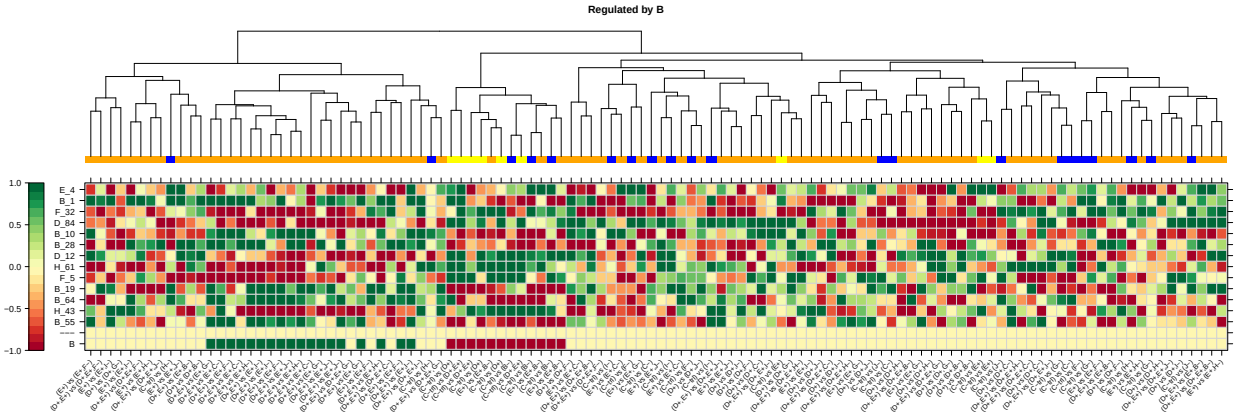

```
## hyper-edge sensitivity and specificity:
print(sum(bString == 1 & resString == 1)/
      (sum(bString == 1 & resString == 1) + sum(bString == 1 & resString == 0)))

## [1] 1

print(sum(bString == 0 & resString == 0)/
      (sum(bString == 0 & resString == 0) + sum(bString == 0 & resString == 1)))

## [1] 1

## accuracy of the expected response scheme (can be high even, if the networks differ):
ERS.res <- computeFc(CNOlist, t(simulateStatesRecursive(CNOlist, model, resString)))
ERS.res <- ERS.res[, which(colnames(ERS.res) %in% colnames(ERS))]
print(sum(ERS.res == ERS)/length(ERS))

## [1] 1
```

In our seeded example 2574 the optimum network is indeed the GTN. If you set the alternative seed at the beginning, neither the network nor the correct ERS is found, if we initialise the search with the empty network. If we start with the fully connected normal graph, the ERS is resolved a 100%, while the greedy optimum still differs from the GTN. Thus the GTN and the greedy optimum are equivalent, but the GTN is larger (more outgoing edges) and thus not preferred.

After optimization we look at the data and how well the greedy optimum explains the E-genes. The lower the score the better the fit.

```
fitinfo <- validateGraph(CNOlist, fc=fc, exprs=exprs, model = model, bString = resString,
                         Sgene = 1, Egenes = 1000, cexRow = 0.8, cexCol = 0.7, xrot = 45,
                         Colv = T, Rowv = T, dendrogram = "both", bordercol = "lightgrey",
                         aspect = "iso", sub = "")

## [1] "1.B: 13"
## [1] "Activated: 6"
## [1] "Inhibited: 7"
## [1] "Summary Score:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.5542 -0.4978 -0.4754 -0.4542 -0.4466 -0.2345
## [1] "Unique genes used: 90 (100 %)"
## [1] "Duplicated genes: 0"
```

```
## [1] "Overall fit:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6368 -0.5190 -0.4505 -0.4327 -0.3601 -0.1497
```



The bottom row shows the ERS of S-gene B and the other rows show the observed response scheme (ORS) of the B-regulated E-genes. Even though the Gaussian noise makes the data look almost random, the greedy optimum is equal to the GTN. Alternatively to the greedy neighbourhood search a genetic algorithm and exhaustive search are available. The exhaustive search is not recommended for search spaces with more than 20 hyper-edges.

```
## genetic algorithm:
genetic <- bnem(search = "genetic",
           fc=fc,
           exprs=exprs,
           parallel = parallel,
           CNOlist=CNOlist,
           model=model,
           initBstring=initBstring,
           popSize = 10,
           stallGenMax = 10,
           draw = FALSE,
           verbose = FALSE
           )

## snowfall 1.84-6.1 initialized (using snow 0.4-2):  parallel execution on 2 CPUs.

## Library CellNOptR loaded.

## Library CellNOptR loaded in cluster.

## Library bnem loaded.

## Library bnem loaded in cluster.
##
## Stopping cluster

resString <- genetic$bString
```

```
## ## exhaustive search:
## exhaustive <- bnem(search = "exhaustive",
##             parallel = parallel,
##             CNOlist=CNOlist,
##             fc=fc,
##             exprs=exprs,
##             model=model
##             )

## resString <- exRun$bString
```

Stimulated and inhibited S-genes can overlap

In this section we show how to use B-NEM, if stimuli and inhibitors overlap. Additionally we want to show that B-NEM can resolve cycles. For this we allow the PKN to have cycles, but no repression, because repression can lead to an unresolvable ERS. See Pirkl et al., 2016 for details.

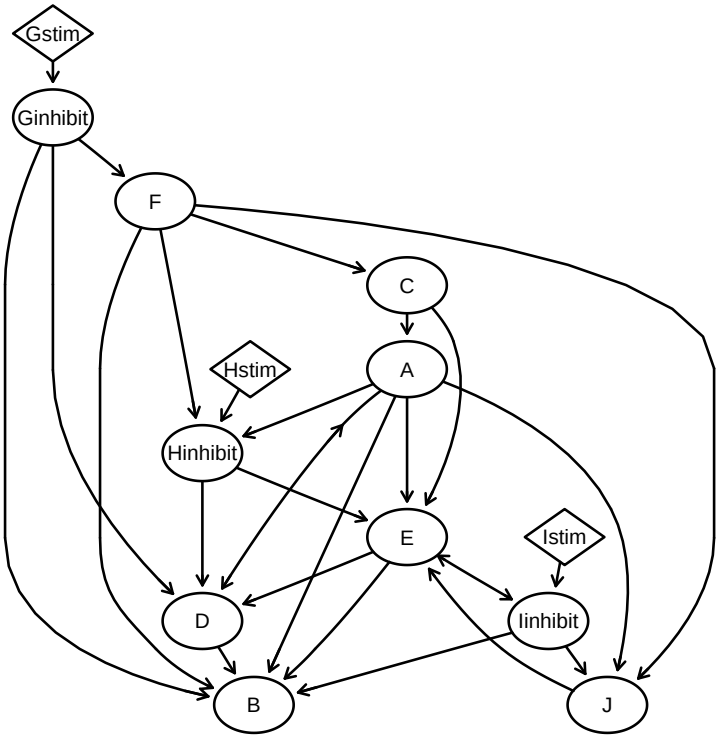
```
## we do not force a DAG but do not allow repression:
dnf <- randomDnf(10, max.edges = 25, max.edge.size = 1, dag = F, negation = F)
cues <- sort(unique(gsub("!", "", unlist(strsplit(unlist(strsplit(dnf, "=")), "\\+")))))
inputs <- unique(unlist(strsplit(gsub("!", "", gsub("=.*", "", dnf)), "=")))
outputs <- unique(gsub(".*=", "", dnf))
stimuli <- c(inputs[which(!(inputs %in% outputs))], cues[sample(1:length(cues), 2)])
inhibitors <- cues
```

We look for stimuli which are also inhibited. For those we add additional stimuli S-genes. The stimuli S-gene (parent) and the inhibitor S-gene (child) are connected by a positive edge.

```
both <- stimuli[which(stimuli %in% inhibitors)]
for (i in both) {
  dnf <- gsub(i, paste(i, "inhibit", sep = ""), dnf)
  dnf <- c(dnf, paste(i, "stim=", i, "inhibit", sep = ""))
  stimuli <- gsub(i, paste(i, "stim", sep = ""), stimuli)
  inhibitors  <- gsub(i, paste(i, "inhibit", sep = ""), inhibitors)
}
```

The next figure shows the cyclic PKN with extra stimuli S-genes. Notice, this way the inhibition of the S-genes overrules the stimulation.

```
plotDnf(dnf, stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 13
## Number of Edges = 28
```



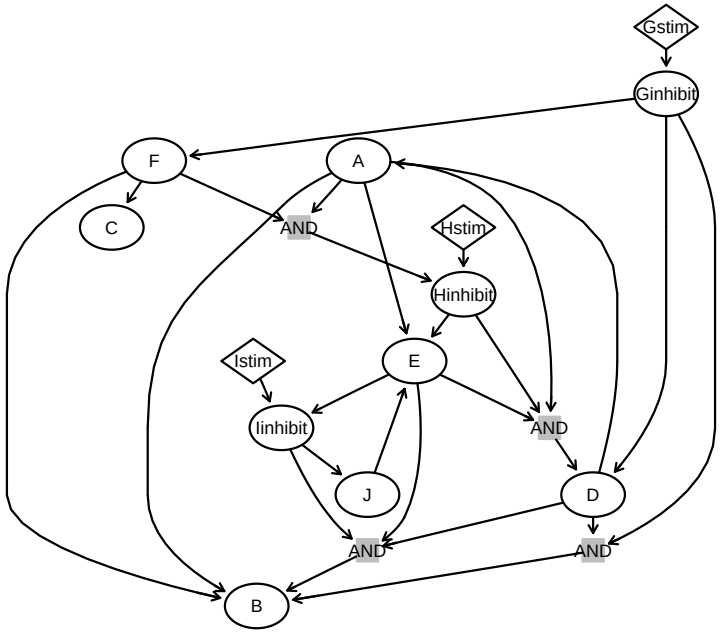
Similar to before we create the full network search space, draw a GTN, simulate data and search for the optimal network.

```
sifMatrix <- NULL
for (i in dnf) {
  inputs2 <- unique(unlist(strsplit(gsub(".*", "", i), "=")))
  output <- unique(gsub(".*=", "", i))
  for (j in inputs2) {
    j2 <- gsub("!", "", j)
    if (j %in% j2) {
      sifMatrix <- rbind(sifMatrix, c(j, 1, output))
    } else {
      sifMatrix <- rbind(sifMatrix, c(j2, -1, output))
    }
  }
}
write.table(sifMatrix, file = "temp.sif", sep = "\t", row.names = FALSE, col.names = FALSE,
            quote = FALSE)
PKN <- readSIF("temp.sif")
## unlink("temp.sif")
CNOlist <- dummyCNOlist(stimuli = stimuli, inhibitors = inhibitors,
                        maxStim = 2, maxInhibit = 1, signals = NULL)
model <- preprocessing(CNOlist, PKN, maxInputsPerGate=100, verbose = F)
```

```
bString <- absorption(sample(c(0,1), length(model$reacID), replace = T), model)
steadyState <- steadyState2 <- simulateStatesRecursive(CNOlist, model, bString)
ind <- grep(paste(inhibitors, collapse = "|"), colnames(steadyState2))
steadyState2[, ind] <- steadyState2[, ind] + CNOlist@inhibitors
while(any(apply(steadyState, 2, sd) == 0) | any(apply(steadyState2, 2, sd) == 0)) {
  bString <- absorption(sample(c(0,1), length(model$reacID), replace = T), model)
```

```
  steadyState <- steadyState2 <- simulateStatesRecursive(CNOlist, model, bString)
  ind <- grep(paste(inhibitors, collapse = "|"), colnames(steadyState2))
  steadyState2[, ind] <- steadyState2[, ind] + CNOlist@inhibitors
}
## we make sure the stimulations work:
bString[grep("stim", model$reacID)] <- 1
bString <- absorption(bString,model)
```

```
plotDnf(model$reacID[as.logical(bString)], stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 17
## Number of Edges = 28
```

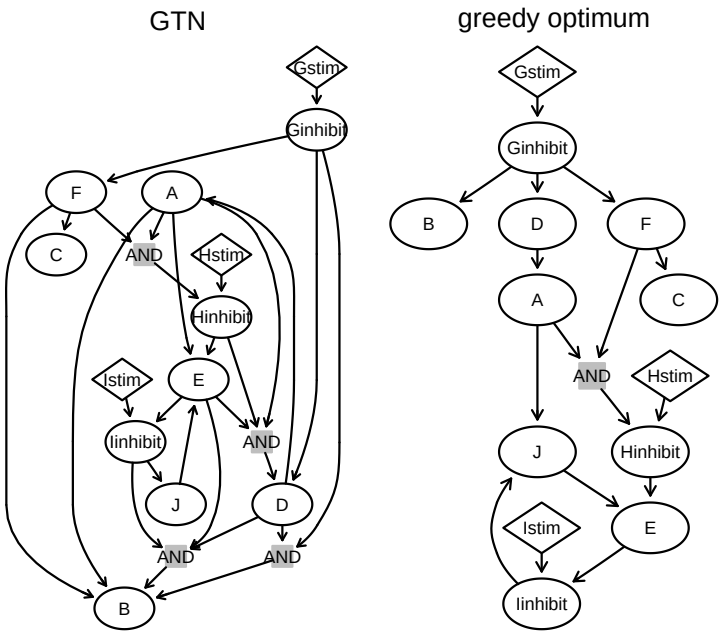



```
exprs <- t(steadyState)[rep(1:ncol(steadyState), 10), rep(1:nrow(steadyState), 3)]
ERS <- computeFc(CNOlist, t(steadyState))
stmcomb <- apply(expand.grid(stimuli, stimuli), c(1,2), as.character)
stimuli.pairs <- apply(stimcomb, 1, paste, collapse = "_")
ind <- grep(paste(c(paste("Ctrl_vs_", c(stimuli, inhibitors), sep = ""),
                    paste(stimuli, "_vs_", stimuli, "_",
                          rep(inhibitors, each = length(stimuli)), sep = ""),
                    paste(stimuli.pairs, "_vs_", stimuli.pairs, "_",
                          rep(inhibitors, each = length(stimuli.pairs)), sep = "")),
                  collapse = "|"), colnames(ERS))
ERS <- ERS[, ind]
fc <- ERS[rep(1:nrow(ERS), 10), rep(1:ncol(ERS), 3)]
fc <- fc + rnorm(length(fc), 0, 1)
flip <- sample(1:nrow(fc), floor(0.33*row(fc)))
fc[flip, ] <- fc[flip, ]*(-1)
rownames(fc) <- paste(rownames(fc), 1:nrow(fc), sep = "_")
print(fc[1:6, c(1:3,(ncol(fc)-2):ncol(fc))])
```

```
##       Ctrl_vs_A  Ctrl_vs_B  Ctrl_vs_C Hstim_Istim_vs_Istim_Hinhibit
## A_1 -0.78149795 -1.0243869 -0.4798684                   -0.45218146
## B_2 -0.01482601 -1.5781127  0.1669494                    0.14834434
## C_3  0.76593851  2.6812805 -0.5003797                   -0.09970571
## D_4 -0.97604903 -0.2445900  1.7634378                   -0.67273006
## E_5 -1.02583577  0.8197866  0.6801132                   -1.07096597
## F_6 -0.82180858  1.0003305  1.1556738                    1.74361097
##     Hstim_Istim_vs_Istim_Iinhibit Hstim_Istim_vs_Istim_J
## A_1                     0.7506221            -0.75686367
## B_2                    -1.1096159             0.68869361
## C_3                    -0.5614034             0.09785174
## D_4                    -0.1764276            -1.34840630
## E_5                    -1.8608468            -1.47816960
## F_6                    -0.8533620             1.09152176
```

```
initBstring <- reduceGraph(rep(0, length(model$reacID)), model, CNOlist)
greedy2 <- bnem(search = "greedy",
            CNOlist=CNOlist,
            fc=fc,
            exprs=exprs,
            model=model,
            parallel=parallel,
            initBstring=initBstring,
            draw = FALSE,
            verbose = FALSE,
            maxSteps = Inf
            )

## snowfall 1.84-6.1 initialized (using snow 0.4-2):  parallel execution on 2 CPUs.

## Library CellNOptR loaded.

## Library CellNOptR loaded in cluster.

## Library bnem loaded.

## Library bnem loaded in cluster.
##
## Stopping cluster

resString2 <- greedy2$bString
```

```
par(mfrow=c(1,2))
plotDnf(model$reacID[as.logical(bString)], main = "GTN", stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 17
## Number of Edges = 28

plotDnf(model$reacID[as.logical(resString2)], main = "greedy optimum", stimuli = stimuli)

## A graphNEL graph with directed edges
## Number of Nodes = 14
## Number of Edges = 16
```




```
print(sum(bString == 1 & resString2 == 1)/
      (sum(bString == 1 & resString2 == 1) + sum(bString == 1 & resString2 == 0)))

## [1] 0.6666667

print(sum(bString == 0 & resString2 == 0)/
      (sum(bString == 0 & resString2 == 0) + sum(bString == 0 & resString2 == 1)))

## [1] 0.9824561

ERS.res <- computeFc(CNOlist, t(simulateStatesRecursive(CNOlist, model, resString2)))
ERS.res <- ERS.res[, which(colnames(ERS.res) %in% colnames(ERS))]
print(sum(ERS.res == ERS)/length(ERS))

## [1] 0.9979757
```

The greedy optimum looks different from the GTN, even though they share a lot, but not all edges (reduced sensitivity and specificity). However the accuracy of the ERS is still 100%.

Pre-attach E-genes

One additional challange for B-NEM compared to methods like CellNetOptimizer is the fact, that B-NEM optimizes the signalling pathway and simulataneously the attachment of the E-genes. However, it is possible to include prior knowledge into the search.

We just have to create a list object, which holds the ERS and prior information about the E-genes.

```
egenes <- list()

for (i in cues) {
    egenes[[i]] <- rownames(fc)[grep(i, rownames(fc))]
}

initBstring <- reduceGraph(rep(0, length(model$reacID)), model, CNOlist)
```

```
greedy2 <- bnem(search = "greedy",
            CNOlist=CNOlist,
            fc=fc,
            exprs=exprs,
            egenes=egenes,
            model=model,
            parallel=parallel,
            initBstring=initBstring,
            draw = FALSE,
            verbose = FALSE,
            maxSteps = Inf
            )

## snowfall 1.84-6.1 initialized (using snow 0.4-2):  parallel execution on 2 CPUs.

## Library CellNOptR loaded.

## Library CellNOptR loaded in cluster.

## Library bnem loaded.

## Library bnem loaded in cluster.
##
## Stopping cluster

resString3 <- greedy2$bString
```

We attach every E-gene to its real parent in the for loop. If an E-gene is only included once in the egenes object, it's position is not learned, but fixed during the optimization of the signalling pathway. Alternatively, we can include one E-gene several times for just a subset of S-genes. This way S-genes, which do not have the E-genes included in their E-gene set are excluded as potential parents.

```
print(sum(bString == 1 & resString3 == 1)/
      (sum(bString == 1 & resString3 == 1) + sum(bString == 1 & resString3 == 0)))

## [1] 0.7222222

print(sum(bString == 0 & resString3 == 0)/
      (sum(bString == 0 & resString3 == 0) + sum(bString == 0 & resString3 == 1)))

## [1] 0.9912281

ERS.res <- computeFc(CNOlist, t(simulateStatesRecursive(CNOlist, model, resString3)))
ERS.res <- ERS.res[, which(colnames(ERS.res) %in% colnames(ERS))]
print(sum(ERS.res == ERS)/length(ERS))

## [1] 1
```

In our toy example, fixing the correct E-genes to their parents increases the accuracy of the network and even resolves the ERS completely.

Visualizing network residuals

We can also quantify how well the attached E-genes fit to the learned network.

```
residuals <- findResiduals(resString3, CNOlist, model, fc, verbose = F) # verbose = TRUE plots the resi
```

Row denote S-genes in the network. Columns denote Contrasts between two experiments. Green colors in the left matrix show the score improves, if no (0) or a negative (-1) response in the network's ERS is changed to positive (+1). Red colors show a zero changed to positive. The right matrix shows the same for switched +1 and -1.

B-Cell receptor signalling

In this section we analyze the B-Cell receptor (BCR) signalling data. The dataset consists of one stimulated S-gene (BCR), three S-genes with available single inhibitions (Tak1, Pik3, Erk) and three S-genes with up to triple inhibitions.

```
data(bcr)
head(fc)
```

```
##              BCR_vs_BCR_Erk BCR_vs_BCR_Ikk2 BCR_vs_BCR_Ikk2_Jnk BCR_vs_BCR_Ikk2_Jnk_p38
## 1552448_a_at      2.0189589      0.04087375          0.48116528               0.4066738
## 1552623_at        0.9608797     -0.34398873         -0.02334786               0.1531595
## 1554018_at       -2.3657004     -0.86858689         -1.19479814              -1.4878822
## 1554067_at        1.3519440      0.29551415          1.12754720               1.1670031
## 1554413_s_at     -0.5918694     -0.35911498         -1.00349004              -1.3765506
## 1554486_a_at     -0.9312615     -0.70027328         -1.32043743              -1.4572034
##              BCR_vs_BCR_Ikk2_p38 BCR_vs_BCR_Jnk BCR_vs_BCR_Jnk_p38 BCR_vs_BCR_Pi3k
## 1552448_a_at         -0.01923705      0.4281865          0.5206746      0.11262679
## 1552623_at           -0.06199606      0.1208266          0.2395189     -0.05526638
## 1554018_at           -0.76218484     -0.5831519         -0.1231922     -1.86135892
## 1554067_at            0.56968258      0.8040793          0.9559501      0.45530266
## 1554413_s_at         -0.40094183     -0.4542043         -0.5547223     -0.89852331
## 1554486_a_at         -0.97043874     -0.6341476         -0.5632004     -0.07047464
##              BCR_vs_BCR_Tak1 BCR_vs_BCR_p38 Ctrl_vs_BCR
## 1552448_a_at       0.6277650     0.10335388   -1.745880
## 1552623_at         0.2350085    -0.08824606   -1.182828
## 1554018_at        -1.3840218    -0.50495621    2.666336
## 1554067_at         0.4630860     0.74459687   -1.039020
## 1554413_s_at      -0.3275965    -0.04957383    1.569679
## 1554486_a_at      -0.5470610    -0.65896879    1.025045
```

We build a PKN to incorporate biological knowledge and account for missing combinatorial inhibitions.

```
negation <- F # what happens if we allow negation?
sifMatrix <- numeric()
for (i in "BCR") {
  sifMatrix <- rbind(sifMatrix, c(i, 1, c("Pi3k")))
  sifMatrix <- rbind(sifMatrix, c(i, 1, c("Tak1")))
  if (negation) {
    sifMatrix <- rbind(sifMatrix, c(i, -1, c("Pi3k")))
    sifMatrix <- rbind(sifMatrix, c(i, -1, c("Tak1")))
  }
}
for (i in c("Pi3k", "Tak1")) {
  for (j in c("Ikk2", "p38", "Jnk", "Erk", "Tak1", "Pi3k")) {
    if (i %in% j) { next() }
```

```
    sifMatrix <- rbind(sifMatrix, c(i, 1, j))
    if (negation) {
      sifMatrix <- rbind(sifMatrix, c(i, -1, j))
    }
  }
}
for (i in c("Ikk2", "p38", "Jnk")) {
  for (j in c("Ikk2", "p38", "Jnk")) {
    if (i %in% j) { next() }
    sifMatrix <- rbind(sifMatrix, c(i, 1, j))
    if (negation) {
      sifMatrix <- rbind(sifMatrix, c(i, -1, j))
    }
  }
}

write.table(sifMatrix, file = "temp.sif", sep = "\t",
            row.names = FALSE, col.names = FALSE, quote = FALSE)
PKN <- readSIF("temp.sif")
unlink("temp.sif")
```

In the next step, we create the meta information. This ensures, that we simulate all the conditions, which are actually available in the data. Furthermore we build our boolean search space based on the PKN.

```
CNOlist <- dummyCNOlist(stimuli = "BCR",
                        inhibitors = c("Tak1", "Pi3k", "Ikk2", "Jnk", "p38", "Erk"),
                        maxStim = 1, maxInhibit = 3)

model <- preprocessing(CNOlist, PKN)

## [1] "The following species are measured: BCR, Erk, Ikk2, Jnk, Pi3k, Tak1, p38"
## [1] "The following species are stimulated: BCR"
## [1] "The following species are inhibited: Erk, Ikk2, Jnk, Pi3k, Tak1, p38"
## [1] "The following species are not observable and/or not controllable: "
```

In the final step we learn the network with the genetic algorithm and deterministic greedy search.

```
initBstring <- rep(0, length(model$reacID))
ga <- bnem(search = "genetic",
               fc=fc,
               CNOlist=CNOlist,
               model=model,
               parallel=2,
               initBstring=initBstring,
               draw = FALSE,
               verbose = FALSE
               )

## snowfall 1.84-6.1 initialized (using snow 0.4-2):  parallel execution on 2 CPUs.

## Library CellNOptR loaded.

## Library CellNOptR loaded in cluster.

## Library bnem loaded.
```

```
## Library bnem loaded in cluster.
##
## Stopping cluster
```

```
print(min(ga$scores))
```

```
## [1] 0.2757441
```

```
initBstring <- rep(0, length(model$reacID))
greedy <- bnem(search = "greedy",
               fc=fc,
               CNOlist=CNOlist,
               model=model,
               parallel=2,
               initBstring=initBstring,
               draw = FALSE,
               verbose = FALSE
               )
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2):  parallel execution on 2 CPUs.
```

```
## Library CellNOptR loaded.
```

```
## Library CellNOptR loaded in cluster.
```

```
## Library bnem loaded.
```

```
## Library bnem loaded in cluster.
##
## Stopping cluster
```

```
print(min(greedy$scores[[1]]))
```

```
## [1] 0.2757441
```

```
par(mfrow=c(1,2))
plotDnf(PKN$reacID, main = "PKN", stimuli = "BCR")
```

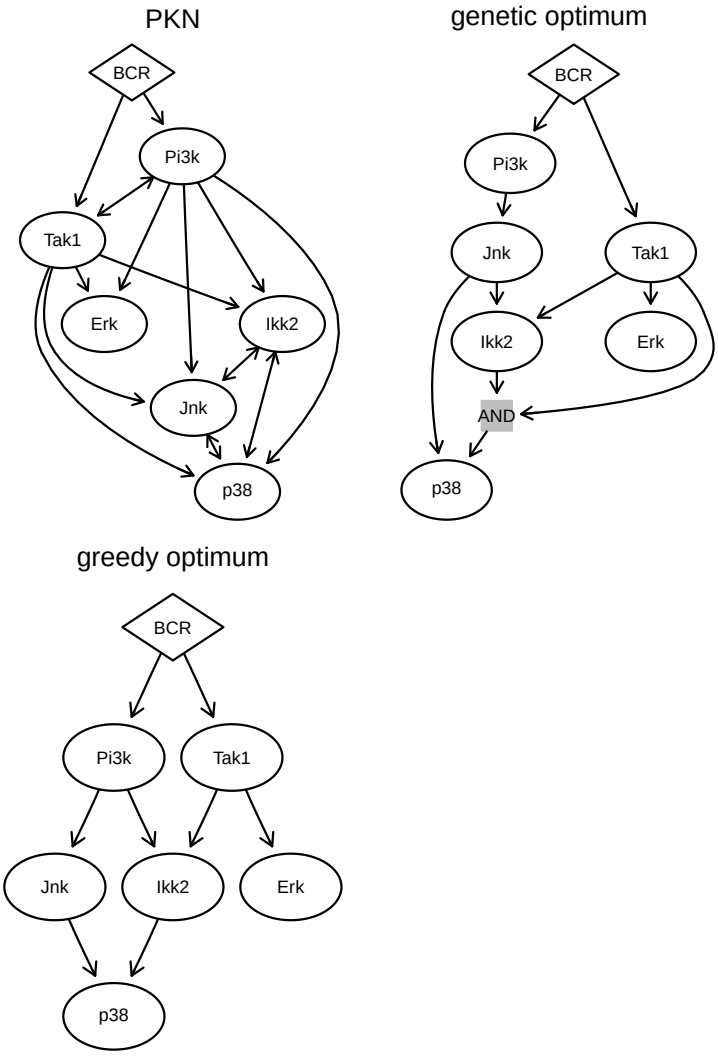
```
## A graphNEL graph with directed edges
## Number of Nodes = 7
## Number of Edges = 18
```

```
plotDnf(ga$graph, main = "genetic optimum", stimuli = "BCR")
```

```
## A graphNEL graph with directed edges
## Number of Nodes = 8
## Number of Edges = 10
```

```
plotDnf(greedy$graph, main = "greedy optimum", stimuli = "BCR")
```

```
## A graphNEL graph with directed edges
## Number of Nodes = 7
## Number of Edges = 8
```




```
sessionInfo()
```

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.11.5 (El Capitan)
##
## locale:
## [1] C/UTF-8/C/C/C/C
##
## attached base packages:
## [1] grid      parallel  stats     graphics  grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
##  [1] bnem_0.99.0         latticeExtra_0.6-28 RColorBrewer_1.1-2  lattice_0.20-34
##  [5] snowfall_1.84-6.1   snow_0.4-2          matrixStats_0.51.0  nem_2.46.0
##  [9] CellNOptR_1.18.0    XML_3.98-1.4        Rgraphviz_2.16.0    RCurl_1.95-4.8
## [13] bitops_1.0-6        ggplot2_2.1.0       hash_2.2.6          RBGL_1.48.1
## [17] graph_1.50.0        BiocGenerics_0.18.0 knitr_1.14          devtools_1.12.0
##
```

```
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.7          BiocInstaller_1.22.3 formatR_1.4          git2r_0.15.0
##  [5] highr_0.6            plyr_1.8.4           class_7.3-14         tools_3.3.1
##  [9] boot_1.3-18          digest_0.6.10        statmod_1.4.26       evaluate_0.10
## [13] memoise_1.0.0        gtable_0.2.0         curl_2.2             e1071_1.6-7
## [17] withr_1.0.2          httr_1.2.1           stringr_1.1.0        stats4_3.3.1
## [21] R6_2.2.0             plotrix_3.6-3        limma_3.28.21        magrittr_1.5
## [25] scales_0.4.0         colorspace_1.2-7     stringi_1.1.2        munsell_0.4.3
```

References:

Pirkl, Martin, Hand, Elisabeth, Kube, Dieter, & Spang, Rainer. 2016. Analyzing synergistic and non-synergistic interactions in signalling pathways using Boolean Nested Effect Models. *Bioinformatics*, 32(6), 893–900.

Pirkl, Martin. 2016. Indirect inference of synergistic and alternative signalling of intracellular pathways. University of Regensburg.

Saez-Rodriguez, Julio, Alexopoulos, Leonidas G, Epperlein, Jonathan, Samaga, Regina, Lauffenburger, Douglas A, Klamt, Steffen, & Sorger, Peter K. 2009. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. Mol Syst Biol, 5, 331.

C Terfve, T Cokelaer, A MacNamara, D Henriques, E Goncalves, MK Morris, M van Iersel, DA Lauffenburger, J Saez-Rodriguez. CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. BMC Systems Biology, 2012, 6:133.