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

Martin Pirkl

August 29, 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.

Loading B-NEM

```
X11.options(type="Xlib")

## install.packages("bnem_1.0.tar.gz")

install.packages("devtools")

## Installing package into '/Users/mpirkl/Library/R/3.3/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/2f/mtjcsd910hz9xnn3shkljg_8005f53/T//RtmpIrICE9/downloaded_packages

library(devtools)

install_github("MartinFXP/B-NEM/package")

## Error in curl::curl_fetch_disk(url, x$path, handle = handle):  Timeout was reached

library(bnem)
```

A simple toy example

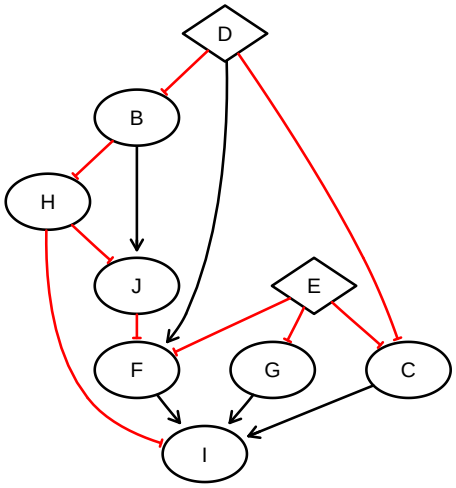
We show how to use B-NEM on a toy example. B-NEM demands several objects as input. The two main objects are the differential gene expression (data) and prior knowledge.

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 the 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 (not necessary)
stimuli <- "dummy"

while(length(stimuli) != 2) {

## random Boolean graph without cycles, maximal 25 edges, maximal edge size of 1 (normal graph) and max
  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 and extend it to a Boolean hyper-graph, which includes all functions allowed by the PKN.

```
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 = FALS
PKN <- readSIF("temp.sif")
## unlink("temp.sif")

## create dummy metainformation:
CNOlist <- dummyCNOlist(stimuli = stimuli, inhibitors = inhibitors, maxStim = 2, maxInhibit = 1, signals

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

## [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: "
```

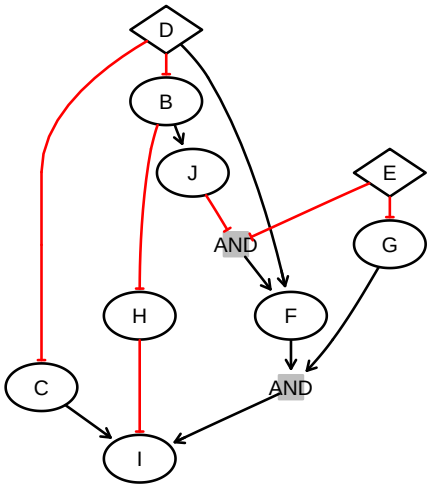
We suggest to take a look at the sif file. In future analyses it is easier to just provide a suitable sif file for the investigated pathway.

In a real world application the underlaying real 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 folloing:
steadyState2[, grep(paste(inhibitors, collapse = "|"), colnames(steadyState2))] <- steadyState2[, grep(p

## this while loop makes sure we get a gtn which actually affects all vertices and no vertices are const
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)

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

}
```

The following figure shows our random GTN, which is a subset of the complete search space (model based on the PKN).

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

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



We now use this GTN to simulate data.

```
## the expression data with 10 E-genes for each S-gene and 3 replicates (not used):
exprs <- t(steadyState)[rep(1:ncol(steadyState), 10), rep(1:nrow(steadyState), 3)]

## we calculate the foldchanges or expected S-gene response scheme (ERS) between certain condtion (e.g.
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 stimul
```

```
stimuli.pairs <- apply(apply(expand.grid(stimuli, stimuli), c(1,2), as.character), 1, paste, collapse =

## this is the usual setup, but "computeFc" calculates a lot more contrasts, which can also be used if
ERS <- ERS[, grep(paste(c(paste("Ctrl_vs_", c(stimuli, inhibitors), sep = ""), paste(stimuli, "_vs_", st

## 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)

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

## don't forget to set rownames (usually gene 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.

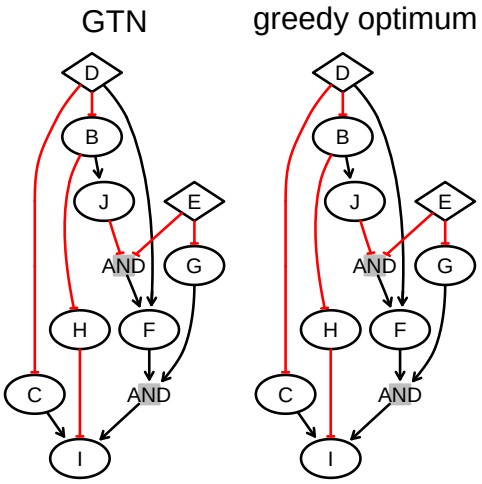
We search for the GTN in our restricted network space. Each network is a sub-graph of the full hyper-graph model$reacID. We initialise the search with a starting network and greedily search the neighbourhood.

```
## start with empty graph:
initBstring <- reduceGraph(rep(0, length(model$reacID)), model, CNOlist)
## initBstring <- reduceGraph(rep(1, length(model$reacID)), model, CNOlist)

## paralellize for several threads on one machine or multiple machines. See package "snowfall" for deta
parallel <- 2 # 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,
            verbose = FALSE,
            maxSteps = Inf
            )
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-1):  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 can now take a look at the efficiency of the search algorithm with sensitivity and specificity of the optimal found network and the accuracy of its ERS (similar to the truth table). Since several network 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

## optimum found:
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
```



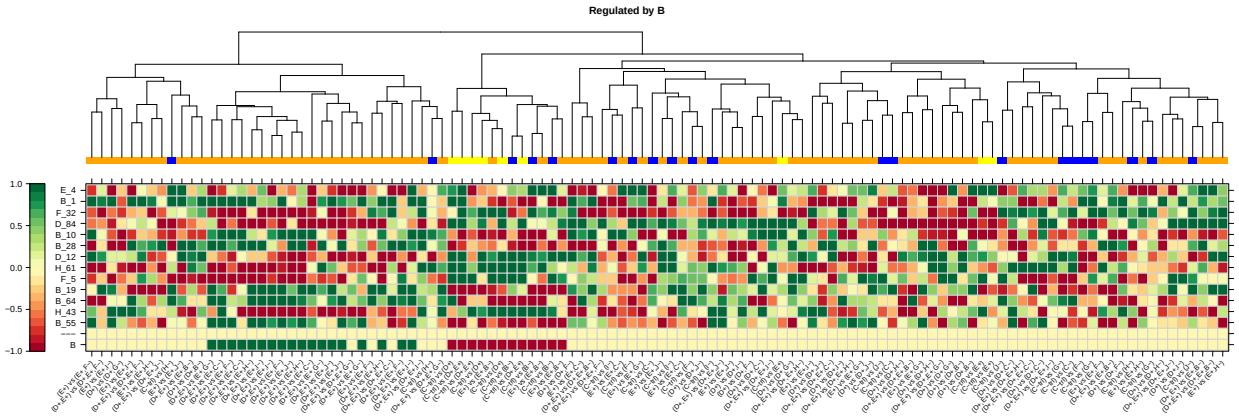

```
## hyper-edge sensitivity and specificity:
print(sum(bString == 1 & resString == 1)/(sum(bString == 1 & resString == 1) + sum(bString == 1 & resStr

## [1] 1

print(sum(bString == 0 & resString == 0)/(sum(bString == 0 & resString == 0) + sum(bString == 0 & resStr
```

```
## [1] 1

## accuracy of expected response scheme from learned network should be high even though the network can
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 with the empty network at the start. If you start with the PKN, the ERS is resolved completely while the network is still not the GTN. Thus the GTN and the optimum are equivalent, but the GTN is larger and thus not preferred.

After optimization you can look at the data and how well your networks 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, Ege

## [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, we still found the GTN. Alternatively to the greedy neightbourhood search a genetic algorithm and exhaustive search are also 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,
           graph = FALSE,
           verbose = FALSE
           )

## snowfall 1.84-6.1 initialized (using snow 0.4-1):  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 when stimuli and inhibitors overlap. 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.

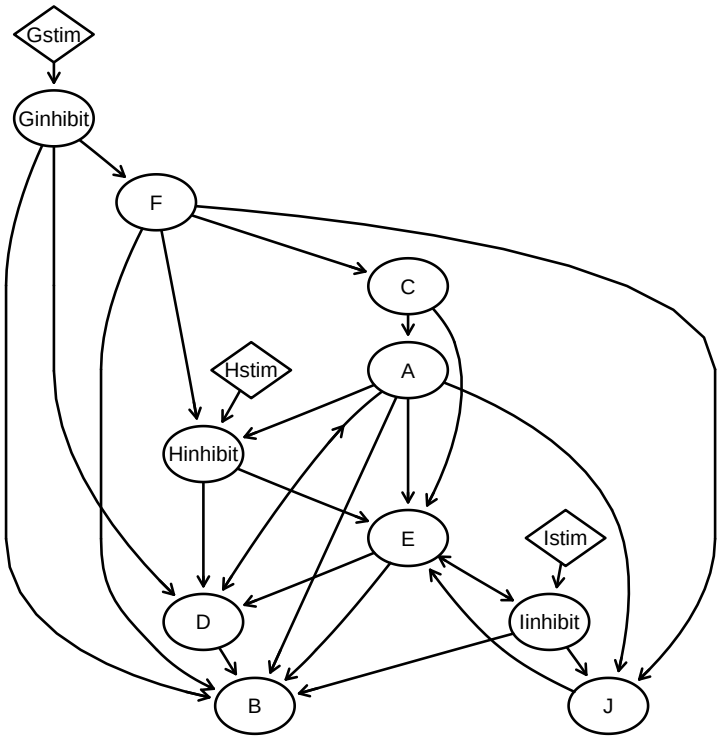
```
## we do not force a DAG but do not allow repression:
dnf <- randomDnf(10, max.edges = 25, max.edge.size = 1, dag = F, negation = F)
cues <- sort(unique(gsub("!", "", unlist(strsplit(unlist(strsplit(dnf, "=")), "\\+")))))
inputs <- unique(unlist(strsplit(gsub("!", "", gsub("=.*", "", dnf)), "=")))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

Now 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
```



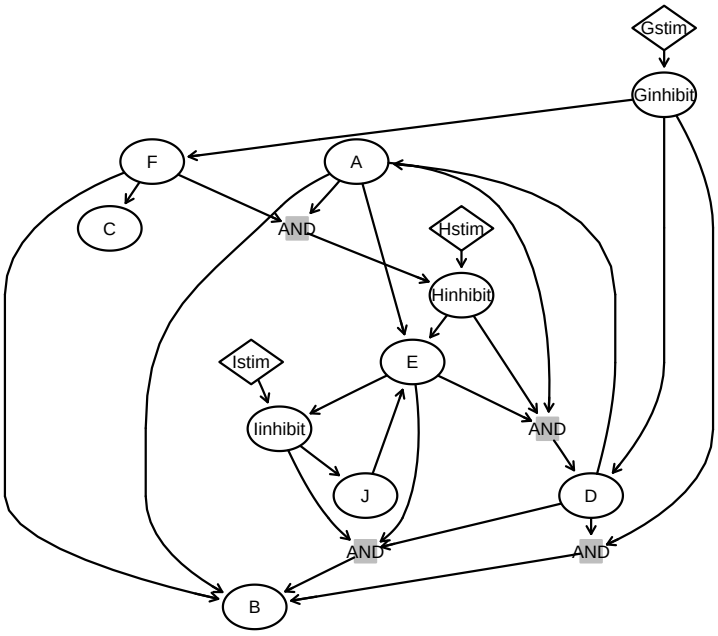
Similar to before we create the full network search space, draw a GTN, simulate data and search for the optimal network.

```
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 = FALS
PKN <- readSIF("temp.sif")
## unlink("temp.sif")
CNOlist <- dummyCNOlist(stimuli = stimuli, inhibitors = inhibitors, maxStim = 2, maxInhibit = 1, signals
model <- preprocessing(CNOlist, PKN, maxInputsPerGate=100)

## [1] "The following species are measured: A, B, C, D, E, F, Ginhibit, Gstim, Hinhibit, Hstim, Iinhibit
## [1] "The following species are stimulated: Gstim, Hstim, Istim"
## [1] "The following species are inhibited: A, B, C, D, E, F, Ginhibit, Hinhibit, Iinhibit, J"
## [1] "The following species are not observable and/or not controllable: "
```

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



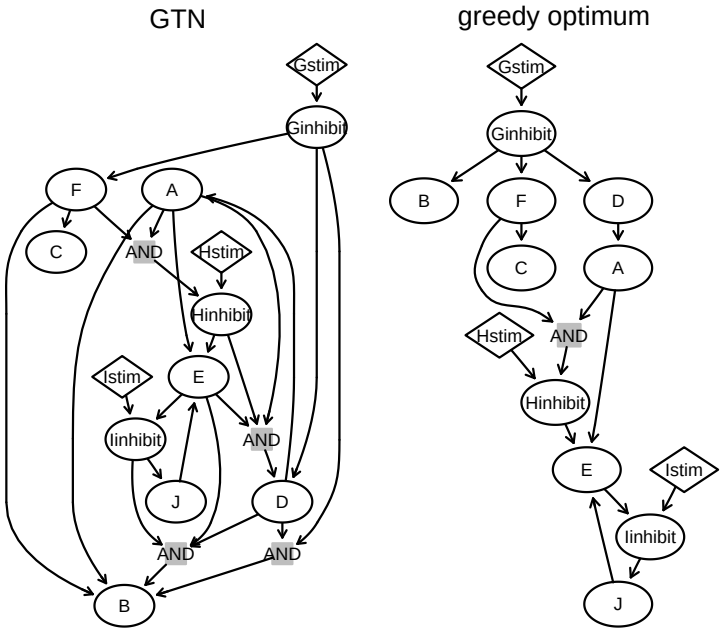

```
exprs <- t(steadyState)[rep(1:ncol(steadyState), 10), rep(1:nrow(steadyState), 3)]
ERS <- computeFc(CNOlist, t(steadyState))
stimuli.pairs <- apply(apply(expand.grid(stimuli, stimuli), c(1,2), as.character), 1, paste, collapse =
ERS <- ERS[, grep(paste(c(paste("Ctrl_vs_", c(stimuli, inhibitors), sep = ""), paste(stimuli, "_vs_", st
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_Hstim_Istim_Hinhibit
## A_1 -0.78149795 -1.0243869 -0.4798684                          0.2478244
## B_2 -0.01482601 -1.5781127  0.1669494                          0.7255230
## C_3  0.76593851  2.6812805 -0.5003797                         -0.8308681
## D_4 -0.97604903 -0.2445900  1.7634378                          1.2054856
## E_5 -1.02583577  0.8197866  0.6801132                         -1.9870181
## F_6 -0.82180858  1.0003305  1.1556738                         -0.8102499
##     Hstim_Istim_vs_Hstim_Istim_Iinhibit Hstim_Istim_vs_Hstim_Istim_J
## A_1                         -1.09111396                   -0.1834281
## B_2                          2.19723830                    0.3406081
## C_3                         -0.02646857                    0.1008377
## D_4                         -0.86552369                   -0.3583840
## E_5                          0.76103492                   -0.4754561
## F_6                         -1.15428818                    1.2788998
```

```
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 0.4-1):  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 & resS

## [1] 0.7222222

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

## [1] 0.9912281

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] 1
```

The optimal network looks very different and has lower sensitivity and specificity. However the accuracy of the ERS is still 100%.

```
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] latticeExtra_0.6-28 RColorBrewer_1.1-2  lattice_0.20-33     bnem_1.0
##  [5] snowfall_1.84-6.1   snow_0.4-1          matrixStats_0.50.2  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 devtools_1.12.0     knitr_1.14
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.6       formatR_1.4       git2r_0.15.0      highr_0.6         plyr_1.8.4
##  [6] class_7.3-14      tools_3.3.1       boot_1.3-18       digest_0.6.10     statmod_1.4.25
## [11] evaluate_0.9      memoise_1.0.0     gtable_0.2.0      curl_1.2          e1071_1.6-7
## [16] withr_1.0.2       httr_1.2.1        stringr_1.0.0     stats4_3.3.1      R6_2.1.2
## [21] plotrix_3.6-3     tcltk_3.3.1       limma_3.28.17     magrittr_1.5      scales_0.4.0
## [26] colorspace_1.2-6 stringi_1.1.1     munsell_0.4.3
```

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), 893900.