CellNOpt tutorial

A. Gabor & A. V. Ponce

1/16/2020

Introduction

The goal of this tutorial is to introduce the CellNOpt framework (Terfve et al 2012) and in particular the CNORode package.

If you have the following issue:

> During startup - Warning messages: 1: Setting LC_CTYPE failed, using "C"
> 2: Setting LC_COLLATE failed, using "C"
> 3: Setting LC_TIME failed, using "C"
> 4: Setting LC_MESSAGES failed, using "C"
> 5: Setting LC_PAPER failed, using "C"

evaluate
`defaults write org.R-project.R force.LANG en_US.UTF-8`
in terminal and then restart RStudio.

CellNOptR

CellNOpt is a software used for creating logic-based models of signal transduction networks using different logic formalisms (Boolean, Fuzzy, or differential equations). CellNOpt uses information on signaling pathways encoded as a Prior Knowledge Network, and trains it against high-throughput biochemical data to create cell-specific models.

These cell specific models can be used,for example, to understand the different signaling patterns among cell-lines or patients or to predict drug response.

Dependencies

```r
# installs devtools package if not already installed
if(!require("devtools")) install.packages('devtools')

# installs CellNOptR and CNORode from GitHub:
if(!require("CellNOptR")) devtools::install_github('saezlab/CellNOptR')
if(!require("CNORode")) devtools::install_github('saezlab/CNORode')
```

If you dont have `devtools` and cannot install it, then

1. please visit the https://github.com/saezlab/CellNOptR and https://github.com/saezlab/CNORode websites,
2. download the toolboxes by clicking "Clone or download" then "Download Zip"
3. Unzip the files
4. In RStudio run:
   `install.packages("../CellNOptR-master", repos = NULL, type = "source")`
   `install.packages("../CNORode-master", repos = NULL, type = "source")`

```r
library(CellNOptR)
library(CNORode)
```

Data

CellNOpt uses a prior knowledge network stated as an interaction file to build a Boolean logic model.

TASK 1: check the format of the SIF file, in `data/tutorial_1_network.sif`: You can do it in a text editor, open it in RStudio or running:

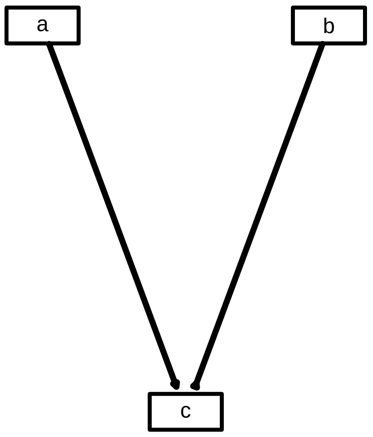
```r
writeLines(readLines("./data/tutorial_1_network.sif"))
```

```
## a 1 c
## b 1 c
```

We find 2 lines that describes 2 interactions between nodes a, b and c. Both node a and node b can activate node c.

Large networks are complicated to check, therefore **TASK 2** Visualise the SIF file in CellNOptR:

```r
model <- readSIF("./data/tutorial_1_network.sif")
plotModel(model)
```



The graph shows the 2 interactions as expected.

The network is already converted to a network object:

```r
print(model)
```

```
## $reacID
## [1] "a=c" "b=c"
##
## $namesSpecies
## [1] "a" "b" "c"
##
## $interMat
##   a=c b=c
## a  -1   0
## b   0  -1
## c   1   1
##
## $notMat
##   a=c b=c
## a   0   0
## b   0   0
## c   0   0
```

- *reacID* enumerates the edges of the network.
- *nameSpecies*: contains the nodes
- *interMat*: is an interaction matrix between nodes and edges
- *notMat*: shows inhibitor edges (none in this model)

TASK 3: check the format of the MIDAS (*Minimum Information for DataAnalysis in Systems Biology*) file, in `data/tutorial_1_data.csv` (best in Excel):

```
writeLines(readLines("./data/tutorial_1_data.csv"))
```

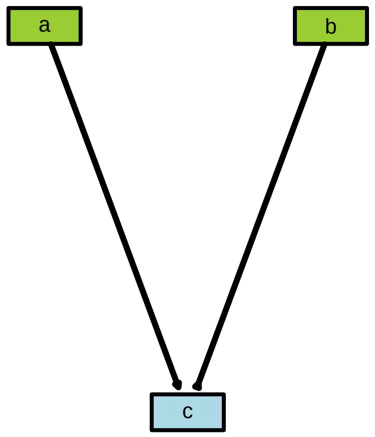
```
## TR:CL:CellLine,TR:a,TR:b,DA:c,DV:c
## 1,0,0,0,0
## 1,0,1,0,0
## 1,1,0,0,0
## 1,1,1,0,0
## 1,0,0,30,0
## 1,0,1,30,0
## 1,1,0,30,0
## 1,1,1,30,1
```

Each row of the MIDAS file encodes a measurement. Column notations:

- *TR*: treatment
- *DA*: time of data acquisition
- *DV*: measured value of the node

TASK 4: Create a CNOlist object from the MIDAS data file and annotate the network

```
cnodata <- CNOlist("./data/tutorial_1_data.csv")
plotModel(model = model, CNOlist = cnodata)
```



Inputs (a and b) are highlighted with green, measured nodes are with blue.

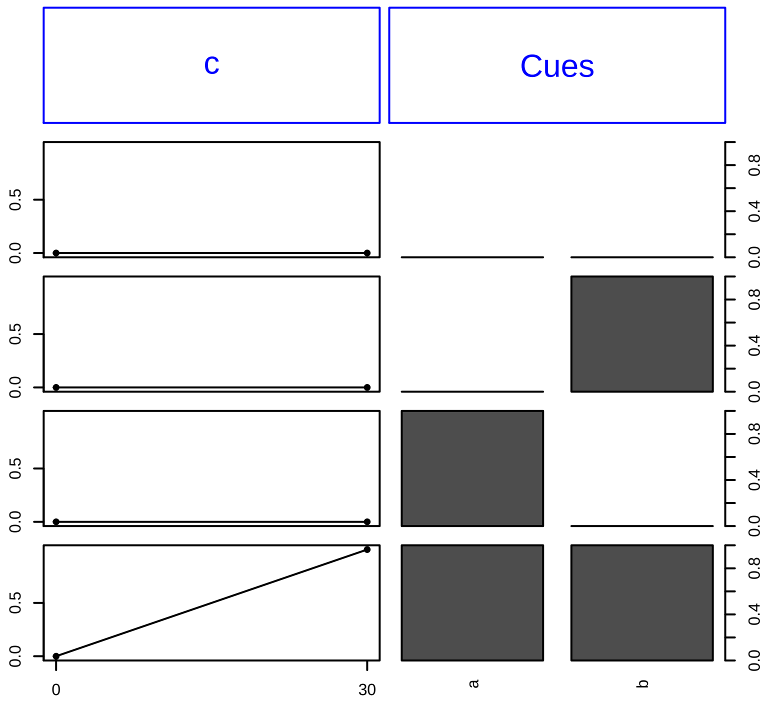
TASK 5: Print and visualise the data object

```
print(cnodata)
```

```
## class: CNOlist
## cues: a b
## inhibitors:
```

```
## stimuli: a b
## timepoints: 0 30
## signals: c
## variances: c
## --
## To see the values of any data contained in this instance, just use the
## appropriate getter method (e.g., getCues(cnolist), getSignals(cnolist), ...
```

```r
plot(cnodata)
```



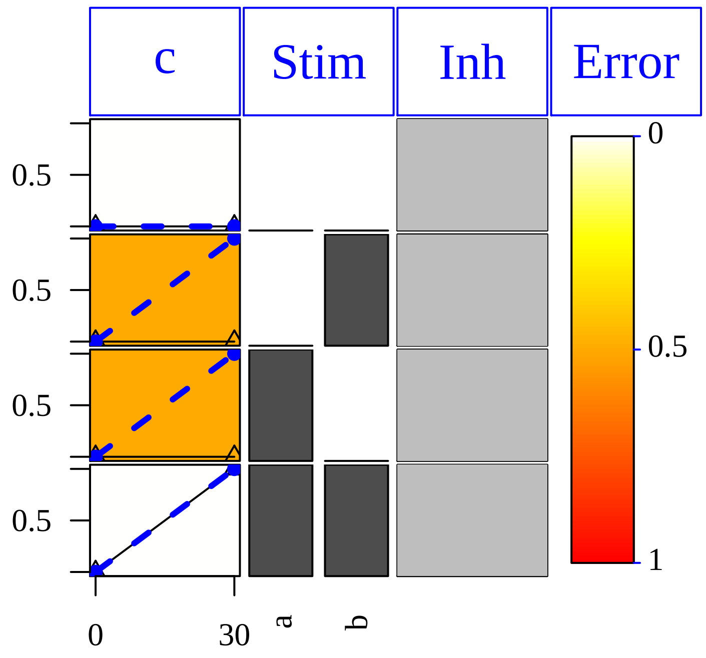
The figure shows an experiment in each line.
Perturbations/Cues (a and b) are 1 (on) or 0 (off). Node C is activated only in the last condition, where both A and B are activated.

Model building

TASK 6: Simulate the model and compare it to the experimental data:

```r
edges = c("a=c" = 1,
          "b=c" = 1)

sim_res <- cutAndPlot(cnodata, model, list(edges))
```



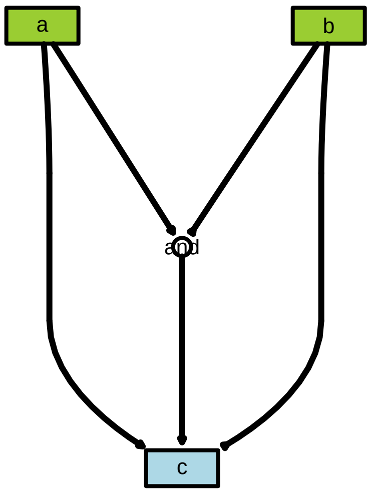
The model predicts an increase of node C if any of A or B increased, i.e. both A and B can activate C.

How do we fix it?

```
prep_model = preprocessing(data = cnodata,
                           model = model,
                           cutNONC = TRUE,  # cut non-controllable subnetwork
                           compression = TRUE, # compress if possible
                           expansion = TRUE, # expand OR gates
                           verbose = TRUE)
```

```
## [1] "The following species are measured: c"
## [1] "The following species are stimulated: a, b"
## [1] "The following species are inhibited: "
## [1] "The following species are not observable and/or not controllable: "
```

```
plotModel(prep_model,cnodata)
```



The preprocessing steps included an **AND** gate between the inputs:

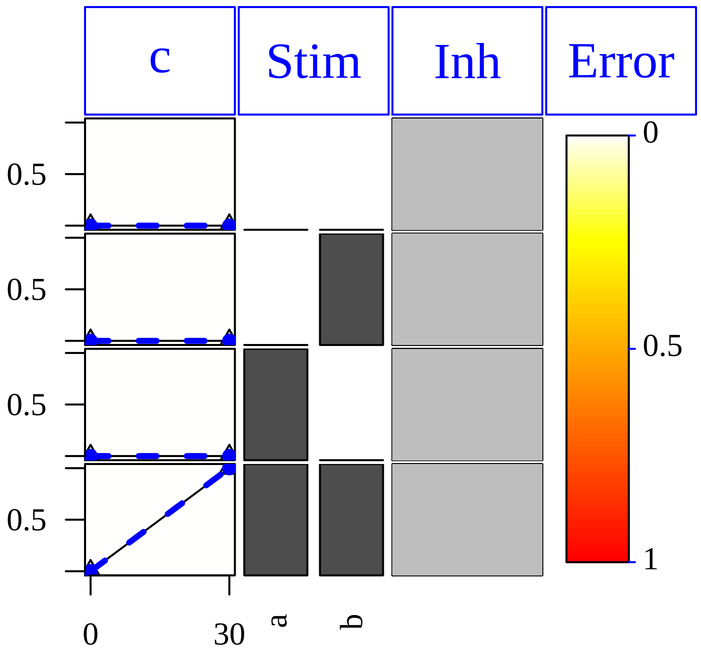
```
print(prep_model$reacID)
```

```
## [1] "a=c"   "b=c"   "a+b=c"
```

Let’s fix the model to match the measured data:

```
# we turn off the a=c and b=c edges:
edges = c("a=c" = 0,
          "b=c" = 0,
          "a+b=c" = 1)

sim_res_and <- cutAndPlot(cnodata, prep_model, list(edges))
```


c
Stim
Inh
Error
a
b