In Silico Systems Biology: Scripting with CellNOptR

Aidan MacNamara

June 25, 2013

Contents

1 Background

CellNOptR is a software package that trains the topology of a PSN to experimental data by the criterion of minimizing the error between the data and the logic model created from the PSN. In CellNOptR, the starting network based on prior knowledge is called the Prior Knowledge Network (PKN). This PKN is preprocessed before training by compression and expansion. The compression step of CellNOptR is a method of reducing the complexity of a logic model by removing nodes that have no effect on the outcome of simulation. The expansion step subsequently includes all possible hyperedges in the model. The model is trained by minimizing a bipartite function that calculates the mismatch between the logic model and experimental data (mean squared error) while penalizing model size. This minimization can be solved using different strategies, from simple enumeration of options for small cases, to stochastic optimization algorithms such as genetic algorithms.

The R version is available on Bioconductor and has a number of added features that allows the user to run different variations of logic modeling within the same framework of model calibration. These variations include steady state

to discrete time Boolean modeling, fuzzy logic and logic ODEs, all of which will be discussed in turn below.

2 Preprocessing

First off, load the necessary libraries, these can be downloaded from Bioconductor, using:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Package Name")
```

```
library(CellNOptR)

## Loading required package:  RBGL
## Loading required package:  graph
## Warning:  package 'graph' was built under R version 3.0.1
## 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
## Warning:  package 'Rgraphviz' was built under R version 3.0.1
## Loading required package:  grid

library(CNORdt)

## Loading required package:  abind

library(CNORode)

## Loading required package:  genalg

library(CNORfuzzy)

## Loading required package:  nloptr
```

Then, create a directory where you can perform your analysis, then set it as your working directory.

```
dir.create("CNOR_analysis")
setwd("CNOR_analysis")
```

To illustrate the variety of logic modeling approaches, we will use an imaginary but biologically plausible prior knowledge network (PKN). This network includes a subset of intracellular signaling networks known to be activated downstream of EGF and TNF stimulation. This is loaded as 'ModelPB':

```
data(modelPB, package = "CNORdt")
```

The in silico data replicates biologically plausible behavior that has been seen in such networks, such as the transient behavior of ERK activation and the oscillatory dynamics of NFκB translocation from the cytoplasm to the nucleus. This is loaded as 'CNOlistPB':

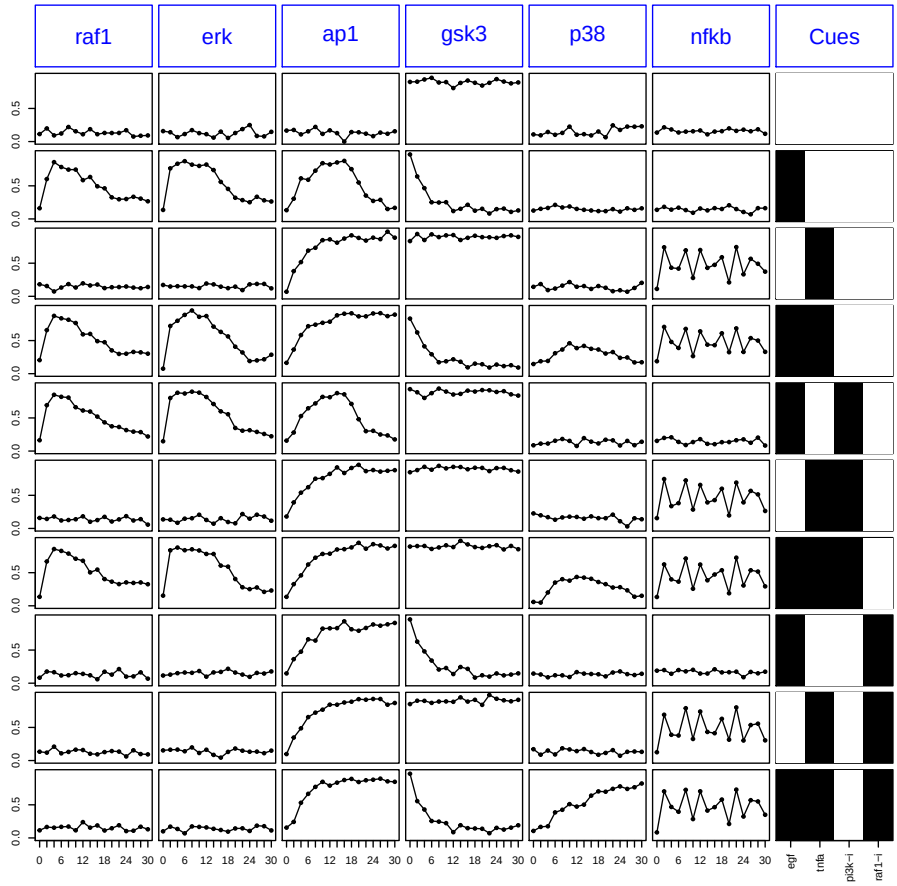
```
data(CNOlistPB, package = "CNORdt")
```

A CNOlist is the central data object of CellNOptR and the add-on packages below. It is the object that contains measurements of elements of a prior knowledge network under different combinations of perturbations of other nodes in the network. A CNOlist comprises the following fields: namesSignals, namesCues, namesStimuli and namesInhibitors, which are vectors holding the names of the measured, stimulated and inhibited species respectively. The fields valueCues (and its derivatives valueStimuli and valueInhibitors) are boolean matrices that contain for each condition (row) a 1 when the corresponding cue (column) is present, and a zero otherwise. You can have a look at your data and the CNOlist format by typing:

```
CNOlistPB
```

The data can also be visualized using:

```
plotCNOlist(CNOlistPB)
```



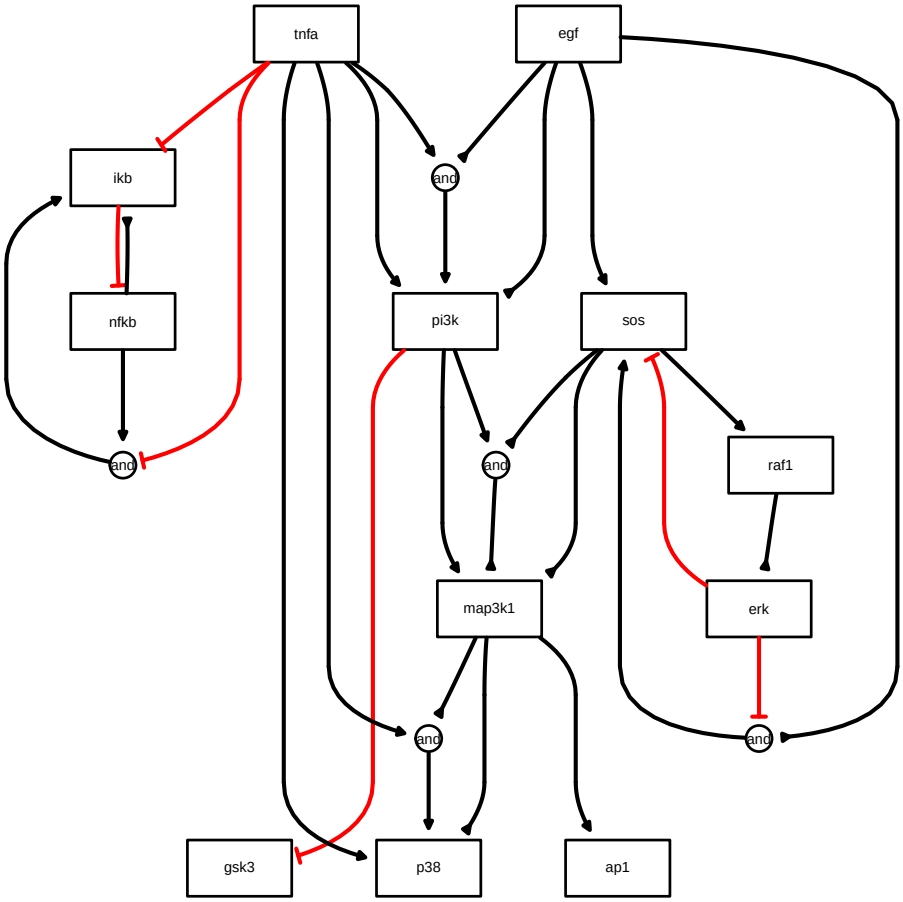
The full details of preprocessing the model can be found in the *CellNOptR* package (the vignette gives a comprehensive explanation):

```
browseVignettes(package = "CellNOptR")
```

```
model = preprocessing(CNOlistPB, modelPB)

## [1] "The following species are measured: raf1, erk, ap1, gsk3, p38, nfkb"
## [1] "The following species are stimulated: egf, tnfa"
## [1] "The following species are inhibited: pi3k, raf1"
## [1] "The following species are not observable and/or not controllable: p90rsk, creb"

plotModel(model)
```



3 Steady State

This is essentially the same example seen for CytoCopteR. With the *in silico* data as our starting point, the PKN is trained using the steady state model formalism at $t_1 = 10$ minutes.

```
# what time point is 'steady state' in the data?
t = 10
CNOlistSS = CNOlistPB
tIndex = which(CNOlistSS$timeSignals == t)
# make a new CNOlist with a single measurement
# time point
CNOlistSS$timeSignals = c(0, t)
CNOlistSS$valueSignals = list(t0 = CNOlistPB$valueSignals[[1]],
    CNOlistPB$valueSignals[[tIndex]])
```

```
opt1 <- gaBinaryT1(CNOlist = CNOlistSS, model = model,
    verbose = FALSE, maxTime = 60)

cutAndPlot(CNOlistSS, model, bStrings = list(opt1$bString))
```

4 Two time points (or additional steady state)

It is quite common in signaling networks to observe a transient behavior where a species is quickly activated and subsequently deactivated. Such a dynamic obviously can not be captured with a steady state approach where only one time point is considered. Therefore in the above section, this issue was avoided by only modeling "fast events" i.e. the activation phase of the signal propagation. However, when information about more than one time point is available and such a fast activation followed by slow deactivation (or indeed any combination of slower and faster processes) is observed, then it is possible to also capture these processes while keeping the simplifying assumption of steady states. In essence, it is assumed that multiple pseudo-steady states reflect the mechanisms that are acting at different time scales and they can be optimized independently. We will illustrate this with the CellNOptR implementation for two time scales, but the approach is extendable to more than two time points.

```
t = c(10, 30)
CNOlistSS2 = CNOlistPB
tIndex = which(CNOlistSS2$timeSignals == t[1])
tIndex[2] = which(CNOlistSS2$timeSignals == t[2])

# make a new CNOlist with 2 time points
CNOlistSS2$timeSignals = c(0, t)
CNOlistSS2$valueSignals = list(t0 = CNOlistPB$valueSignals[[1]],
    CNOlistPB$valueSignals[[tIndex[1]]], CNOlistPB$valueSignals[[tIndex[2]]])

opt2a <- gaBinaryT1(CNOlist = CNOlistSS2, model = model,
    maxTime = 60, verbose = FALSE)

# optimise T2
opt2b <- gaBinaryT2(CNOlist = CNOlistSS2, bStringT1 = opt2a$bString,
    model = model, maxTime = 60, verbose = FALSE)

cutAndPlot(CNOlistSS2, model, list(opt2a$bString, opt2b$bString))
```

5 Synchronous multiple time point simulation with CNORdt

CNORdt introduces some variation in how time is handled in the model. Instead of simulating and fitting data at steady states, it is capable of fitting time course data by using an additional model parameter together with a synchronous updating scheme.

CNORdt introduces a scaling parameter that defines the time scale of the Boolean synchronous simulation. Where each "tick" (t) (or simulation step) is the synchronous updating of all nodes in the model according to their inputs at $t-1$, the scaling parameter defines the "tick" frequency relative to the time scale of the real data. Although this is a crude approach (i.e. it implies a single rate across all reactions), it allows us to fit a synchronous Boolean simulation to data. Hence, all data points can be fitted to the model and hyperedges that cause feedback in the model can be included, which allows the model to reveal more complex dynamics such as oscillations.

More information can be found in the Bioconductor vignette:

```
browseVignettes(package = "CNORdt")
```

The following is an example using CNORdt:

```
opt3 <- gaBinaryDT(CNOlist = CNOlistPB, model = model,
    boolUpdates = 30, maxTime = 60, lowerB = 0.8, upperB = 10)

cutAndPlotResultsDT(CNOlist = CNOlistPB, bString = opt3$bString,
    model = model, boolUpdates = 30, lowerB = 0.8,
    upperB = 10)
```

6 Constrained fuzzy logic with CNORfuzzy

Fuzzy logic is another logic modeling formalism that allows for intermediate levels of activation. Briefly, the relationships (or transfer functions) between nodes in CNORfuzzy are limited to Hill functions. Hence, each transfer function has 2 free parameters: the Hill coefficient n, which controls the steepness of the function, and the sensitivity parameter k, which determines the midpoint of the function (i.e. the value of the input that produces half the maximal output). By varying these 2 parameters, linear, sigmoidal and step-like dynamics can be produced that are good approximations to protein-protein interactions and enzymatic reactions. Full details can be found by the searching for the vignette in R:

```
browseVignettes(package = "CNORfuzzy")
```

The following wrapper function finds the optimized model:

```
# run the wrapper first
optCFL = CNORwrapFuzzy(data = CNOlistSS, model = model)

# summarize results
summary = compileMultiRes(list(optCFL), show = TRUE)

# plot
plotMeanFuzzyFit(0.001, summary$allFinalMSEs, list(optCFL))
```

CNORode allows for continuous states and time by encoding the logic model as a set of ODEs. The transfer functions between nodes are Hill functions that allow for a wide variety of dynamics (to obtain a good fit to data, you may need to increase the number of optimization iterations: 'iters'):

```
initParams = createLBodeContPars(model, random = TRUE)
odeParams = parEstimationLBodeGA(CNOlistPB, model,
    initParams, iters = )

# plot
plotLBodeFitness(CNOlistPB, model, odeParams, iters = 100)
```