Training Signalling Pathway Maps to Biochemical Data with Logic-Based Ordinary Differential Equations

David Henriques[1] and Thomas Cokelaer *[2]

[1]Instituto de Investigaciones Marinas-CSIC, Vigo, Spain and European Bioinformatics Institute, Saez-Rodriguez group, Cambridge, United Kingdom
[2]European Bioinformatics Institute, Saez-Rodriguez group, Cambridge, United Kingdom

September 7, 2012

Contents

1 Introduction

Mathematical models are used to understand protein signalling networks so as to provide an integrative view of pharmacological and toxicological processes at molecular level. *CellNOptR* [**?**] is an existing package (see http://bioconductor.org/packages/release/bioc/html/CellNOptR.html) that provides functionalities to combine prior knowledge network (about protein signalling networks) and perturbation data to infer functional characteristics (of the signalling network). While *CellNOptR* has demonstrated its ability to infer new functional characteristics, it is based on a boolean formalism where protein species are characterised as being fully active or inactive. In contrast, logic-based ordinary differential equations allow a quantitive description of a given Boolean model.

The method used here was first published by Wittmann et al. **??** by the name of odefy. For a detailed description of the methodology the user is adressed to **??** and for a published application example **??**.

This package implements the Odefy method and focus mainly extending the CellNOptR capabilities in order to simulate and calibrate logic-based ordinary differential equation model. We provide direct and easy to use interface to optimization methods available in R such as eSSR (enhanced Scatter Search Metaheuristic for R) and an R genetic algorithm implementation by the name of genalg in order to perform parameter estimation. Additionally we were specially carerful in tackling the main computanional bottlenecks by implementing CNORode simulation engine in the C languange using the CVODES library.

This brief tutorial shows how to use CNORode using as a starting point a Boolean model and a dataset consisting on a time-series of several proteins.

2 Installation

CNORode depends on *CellNOptR* and its dependencies (bioconductor packages), which can be installed in R. In order to install *CellNOptR*, open a R session and type:

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

*cokelaer@ebi.ac.uk

It may take a few minutes to install all dependencies if you start from scratch (i.e, none of the R packages are installed on your system). Then, you can install *CNORode* similarly:

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

These two packages depends on other R packages (e.g., *RBGL*, *nloptr*), which installation should be smooth. Note, however, that there is also an optional dependency on the *Rgraphviz* package, whose compilation may be tricky under some systems such as Windows (e.g., if the graphviz library is not installed or compiler not compatible). Next release of Rgraphviz shoudl fix this issue. Meanwhile, if Rgraphviz cannot be installed on your sytem, you should still be able to install *CellNOptR* and *CNORode* packages and to access most of the functionalities of these packages. Note also that under Linux system, some of these packages necessitate the R-devel package to be installed (e.g., under Fedora type *sudo yum install R-devel*).

Furthermore, for parameter estimation we recommend the use of eSSR. This is algorithm is part of the MEIGOR toolbox. Although to the date, this package is not available in any oficial repository, it can be downloded from http://www.ebi.ac.uk/saezrodriguez/cno/meigo/.

In order to install MEIGOR do:

```
install.packages("~/MEIGOR_0.99.1_svn2120.tar.gz",type="source")
```

Note " /" should point to the directory where the file is saved.

Finally, once *CNORode* is installed you can load it by typing:

```
library(CNORode)
```

3 Quick Start

In this section, we provide a quick example on how to use CNORode to find the set of continous parameters which minimize the squared difference between a model simulation and the experimental data.

Since here we will not be modifying the model structure as opposed to *CellNOptR* we will use a model that already contains AND type gates. Such model can be for instance the result of calibrating a *prio knowledge network*(PKN) with *CellNOptR*. Please note a PKN can also be used as Boolean model which will contain only OR type gates.

Detailed information about the model used here (ToyModelMMB_FeedbackAnd) and additional models can be found at:

http://www.ebi.ac.uk/~cokelaer/cno/doc/sampleModels/

The example used here is shipped with the R package. In order to load the data and model you should type the following commands:

```
library(CNORode)
model=readSIF(system.file("/doc/ToyModelMMB_FeedbackAnd.sif",package="CNORode"));
cno_data=readMIDAS(system.file("/doc/ToyModelMMB_FeedbackAnd.csv",package="CNORode"));
cnolist=makeCNOlist(cno_data,subfield=FALSE);
```

The structure from the CNOlist and the Model object is exactly the same as used in the *CellNOptR* and therefore for a detailed explanation about these structure we direct the reader to the *CellNOptR* manual.

In order to simulate the model and perform parameter estimation we first need to create a list with the ODE parameters associated with each dynamic state as described in []. Each dynamic state will have a τ parameter, as many n and k parameters as inputs. Although the default is to use the normalized Hill function

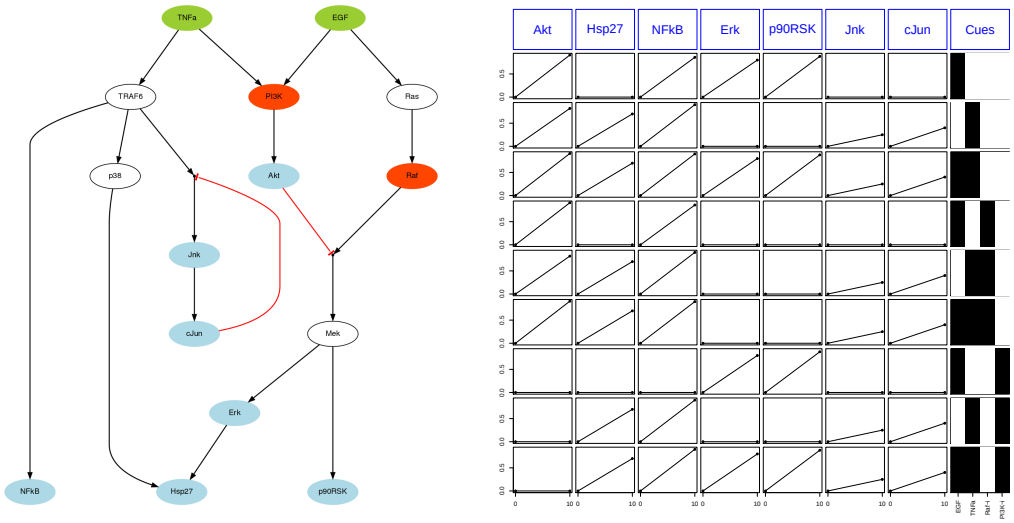

Figure 1: The used model(left panel). A plot from the data, resulting from the *plotCNOlist* function (right panel).

it also possible to use the standard Hill or even not to use any transfer function. To illustrate shape of the equations associated to each dynamic state and the meaning of each parameter will show the differential of Mek.

$$\dot{MEK} = \left(1 - \frac{Akt^{n_1}/({k_1}^{n_1} + Akt^{n_1})}{1/({k_1}^{n_1} + 1)}\right)\left(\frac{Raf^{n_2}/({k_2}^{n_2} + Raf^{n_2})}{1/({k_2}^{n_2} + 1)}\right)$$

```
ode_parameters=createLBodeContPars(model, LB_n = 1, LB_k = 0.1, LB_tau = 0.01,
 UB_n = 5, UB_k = 0.9, UB_tau = 10, default_n = 3, default_k = 0.5,
 default_tau = 1, LB_in = c(), UB_in = c(), opt_n = TRUE, opt_k = TRUE,
 opt_tau = TRUE, random = FALSE)
```

4 Detailled example

4.1 The PKN model and data

The *CellNOptR* package contains a data set that is more realistic, which is part of the network analysed in [**?**] and comprises 40 species and 58 interactions in the PKN. This network was also used for the signaling challenge in DREAM4 (see http://www.the-dream-project.org/). The associated data was collected in hepatocellular carcinoma cell line HepG2 [**?**]. The prior knowledge network is presented in Figure **??**. In this section, we will proceed to the same analysis as above taking more time to understand how to set the parameters and chose the proper threshold.

4.2 Parameters

As mentioned earlier the ToyModel is a very simple example: the Genetic Algorithm converge quickly even with small population and only one instance of optimisation suffices to get the optimal model. The DREAM case is mode complex. We will need a more thorough analysis. First, let us look at the parameters in more

details. The following sample codes shows what are the parameters that a user can change. Let us start with the Genetic Algorithm parameters.

First, we use set a list of default parameters (line 2). We could keep the default values but to show how to change them, let us manually set the population size (line 4), the maximum time for a Genetic Algorithm optimisation (line 5), the maximum number of generation (line 6) and the maximum number of stall generation (line 7). Note that care must be taken on the lower and upper cases names (a non homogeneous caps convention is used!).

Next, let us look at the fuzzy logic parameters. There are three types: *Type1Funs*, *Type2Funs* and *ReductionThreshold*. In the code below, we set the *Type1Funs* parameters. It contains the parameter of the Hill transfer functions. It is a matrix of n transfer functions times the 3 parameters g, n and k. The parameter g is the gain of the transfer function (set to 1). k is the sensitivity parameter which determines the midpoint of the function. n is the Hill coefficient, which determines the sharpness of the sigmoidal transition between the high and low output node values (see Figure **??**-a for a graphical representation).

Note that the last value of n is set to 1.01 because a Hill coefficient n of 1 is numerically unstable. Note also that 68.5095 is the maximum k value to be used.

The parameters *Type2Funs* set transfer functions that connects stimuli to downstream species. They are used so that these species can be connected with different transfer functions if desired. There is no need to change these transfer function parameters except for the number of rows by changing *nrow* to a differnt value. Note that *nrow* must be consistent (identical) for the *Type1Funs* and *Type2Funs* parameters.

ReductionThresh is a list of threshold to be used during the reduction step. This vector is used for instance in Figure **??** to set the x-axis.

Finally, you can also set the optimisation parameters used in the refinement step, which affects the duration of the simulation significantly. As compared to the default parameter, we reduce the maxtime:

See the appendix **??** for a detailled list of the parameters used in this package.

4.3 Analysis

Once the parameters are set, similarly to Section **??**, we perform the analysis using the *CNORwrapFuzzy* function. However, this time we set N to a value greater than 1 to use several runs, as recommended in the general case of complex models.

Note that the analysis with complex model and as many as 7 transfer functions could be quite long to compute. An upper time estimation is $n \times (GA_{maxT} + O_{maxT} \times (L+1))$ where gA_{maxT} is the maximum time spend in the genetic algorithm optimisation (*paramsList$maxTime*), o_{maxT} is the maximum time for the optimisation in the refinement step (*paramsList$optimisation$maxtime*) and L is the number of reduction threshold (*paramsList$redThres*).

The previous sample code calls the function *compileMultiRes* that combines together the different optimisations inside the variable *summary*. In addition there is a plot generated (see Figure **??**) that helps on chosing the parameter for the next function (*plotMeanFuzzyFit*). Indeed, we want to obtain a model that achieves a minimum MSE while keeping the number of parameters small. A compromise has to be found according to the value of the Reduction threshold. This is done by looking at Figure **??** and chosing a threshold before the mean MSE starts to increase significantly. In our example, the reduction threshold should be around 10^{-2}. Let us plot the results for two different threshold. First, let us use the optimal threshold:

This function creates the network resulting from the training a cFL model to data in multiple runs. The weights of the edges are computed as the mean across models using post refinement threshold to choose reduced refined model resulting from each run. As with *writeNetwork* (in CellNOptR), this function maps back the edges weights from the optimised (expanded and compressed) model to the original model. Note that the mapping back only works if the path has length 2 at most (i.e., you have node1-comp1-comp2-node2, where comp refer to nodes that have been compressed). This function saves several files with the tag *output_dream*:

References

[1] C. Terfve. CellNOptR: R version of CellNOpt, boolean features only. R package version 1.2.0, (2012) http://www.bioconductor.org/packages/release/bioc/html/CellNOptR.html

[2] L.G. Alexopoulos, J. Saez-Rodriguez, B.D. Cosgrove, D.A. Lauffenburger, P.K Sorger.: Networks inferred from biochemical data reveal profound differences in toll-like receptor and inflammatory signaling between normal and transformed hepatocytes. Molecular & Cellular Proteomics: MCP **9**(9), 1849–1865 (2010).

[3] M.K. Morris, I. Melas, J. Saez-Rodriguez. Construction of cell type-specific logic models of signalling networks using CellNetOptimizer. *Methods in Molecular Biology: Computational Toxicology*, Ed. B. Reisfeld and A. Mayeno, Humana Press.

[4] M.K. Morris, J. Saez-Rodriguez, D.C. Clarke, P.K. Sorger, D.A. Lauffenburger. Training Signaling Pathway Maps to Biochemical Data with Constrain ed Fuzzy Logic: Quantitative Analysis of Liver Cell Responses to Inflammatory Stimuli. *PLoS Comput Biol.* 7(3) (2011) : e1001099.

[5] J. Saez-Rodriguez, L. Alexopoulos, J. Epperlein, R. Samaga, D. Lauffenburger, S. Klamt and P.K. Sorger. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular Systems Biology*, 5:331, 2009.

A Default parameters

Parameter name	Type	Default values	Description
Objective Function Parameters			
sizeFac	positive real	0	Each input to a logic gate is penalized by this amount (i.e., a two-input AND gate is penalized by this factor twice). Must be zero for reliable training.
NAFac	positive real	0	Penalty assigned to nodes that are not calculable in the simulation. Nodes might not be calculable because they oscillate due to a feed back loop. (value of 1 is the largest possible error between the simulation and data).
Genetic Algorithm Parameters			
PopSize	positive integer	50	Number of individuals tested at each generation.
Pmutation	positive real	0.5	probability that one bit/number in each individual is randomly changed (at each generation).
MaxTime	positive integer	180	stop criteria based on a maximum amount of time (seconds).
maxGens	positive integer	500	stop criteria based on a maximum number of generations.
StallGenMax	positive integer	100	stop criteria based on a constant objective function for that number of generations.
SelPress	positive real ≥ 1	1.2	If fitness is assigned according to the rank, this number is used in the calculation of fitness to increase the speed of loss of diversity and thus, convergence.
elitism	positive integer	5	Number of individuals retained for the successive generation.
RelTol	positive real	0.1	All solutions found by the GA within this fraction of the best solution are returned.
verbose	boolean	FALSE	

Parameter name	**Type**	**Description**	**Default values**
Fuzzy Parameters			
Type1Funs	a $w \times 3$ matrix where w is the number of transfer functions	g=(1,1,1,1,1,1,1), n=(3,3,3,3,3,3,1.01), k=(0.2,0.3,0.4,0.55,0.72, 1.03,68.5098)	The first column contains the gain (g), the second the Hill coefficient (n) and the third the sensitivity parameter (k).
Type2Funs	a $w \times 3$ matrix	g=(0.2,0.3,0.4, 0.5,0.6,0.7,0.8), n=(1,1,1,1,1,1,1), k=(1,1,1,1,1,1,1)	transfer functions that GA chooses from for relating the stimuli inputs to outputs. Same format as Type1Funs.
RedThresh	vector of positive real number	c(0, 0.0001, 0.0005, 0.001, 0.003,0.005, 0.01)	Reductions thresholds used during reduction step. If the reduction threshold is too high (greater than 0.01), empty models may be returned, resulting in a failure of the reduction and refinement stages
DoRefinement	boolean	TRUE	
Optimisation Refinement Parameters			
algorithm	string	NLOPT_LN_SBPLX	optimisation algorithm (nloptr package)
xtol_abs	positive real	0.001	stop criteria based on the absolute error tolerance
maxeval	positive integer	1000	stop criteria based on the maimum number of evaluations
maxtime	positive integer	5*60	stop criteria based on a maximum amount of time (seconds)