Training Signalling Pathway Maps to Biochemical Data with Constrained Fuzzy Logic using CNORfuzzy

Melody K.Morris[1] and Thomas Cokelaer *[2]

[1]Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge MA, U.S.
[2]European Bioinformatics Institute, Saez-Rodriguez group, Cambridge, United Kingdom

October 1, 2012

Contents

1 Introduction **1**

2 Installation **2**

3 Quick Start **2**

4 Detailled example **5**

4.1 The PKN model and data 5
4.2 Parameters 6
4.3 Analysis 7

A Default parameters **12**

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* [1] 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, the Constraint Fuzy Logic formalism [4] implemented in this package (called *CNORfuzzy*) generalises the boolean logic to model quantitative data.

The constrained Fuzzy Logic modelling (also denoted cFL) is fully described in [4]. It was first implemented in a Matlab toolbox *CellNOpt* (available at http://www.ebi.ac.uk/saezrodriguez/software.html#CellNetOptimizer). More information about the methods and application of the Matlab pipeline can be found in references [3, 4].

In this document, we show how to use the *CNORfuzzy* on biological model/data sets. Since *CNORfuzzy* and this tutorial use functions from *CellNOptR*, it is strongly recommended to read the *CellNOptR* tutorial before carrying on this tutorial.

*cokelaer@ebi.ac.uk

2 Installation

CNORfuzzy depends on *CellNOptR* and its dependencies (bioconductor packages) and *nloptr*, which can be installed in R. 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 *CNORfuzzy* similarly:

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

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 *CNORfuzzy* 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*).

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

```
library(CNORfuzzy)
```

3 Quick Start

In this section, we will show you how to run the pipeline to optimise a set of model and data and how to get the optimised model using the constrained fuzzy logic.

As in *CellNOptR*, there is a function that does most of the job for you, which is called *CNORwrapFuzzy*. We will detail this function step by step in the next section but for now, let us see how to obtain an optimised model in a few steps. First, we need a model and a data set. We will use the same toy model as in *CellNOptR*:

```
library(CNORfuzzy)
data(CNOlistToy, package="CellNOptR")
data(ToyModel, package="CellNOptR")
```

The object ToyModel is a data frame that contains the Prior Knowledge Network (PKN) about the model. For instance, it contains a list of all reactions (see Table 1). A graphical representation of the model is shown in Figure 1. See *CellNOptR* tutorial for more details [1].

	1	2	3	4
1	EGF=Ras	TRAF6=p38	p38=Hsp27	!Akt=Mek
2	EGF=PI3K	TRAF6=Jnk	PI3K=Akt	Mek=p90RSK
3	TNFa=PI3K	TRAF6=NFkB	Ras=Raf	Mek=Erk
4	TNFa=TRAF6	Jnk=cJun	Raf=Mek	Erk=Hsp27

Table 1: The ToyModel object contains a prior knowledge network with 16 reactions stored in the field *ToyModel$reacID*. There are other fields such as *namesSpecies* or *interMat* that are used during the analysis.

The object *CNOlistToy* is a CNOlist 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 names of the signals, cues, stimuli and inhibitors and is used to represent the PKN model with colored nodes as in Figure 1.

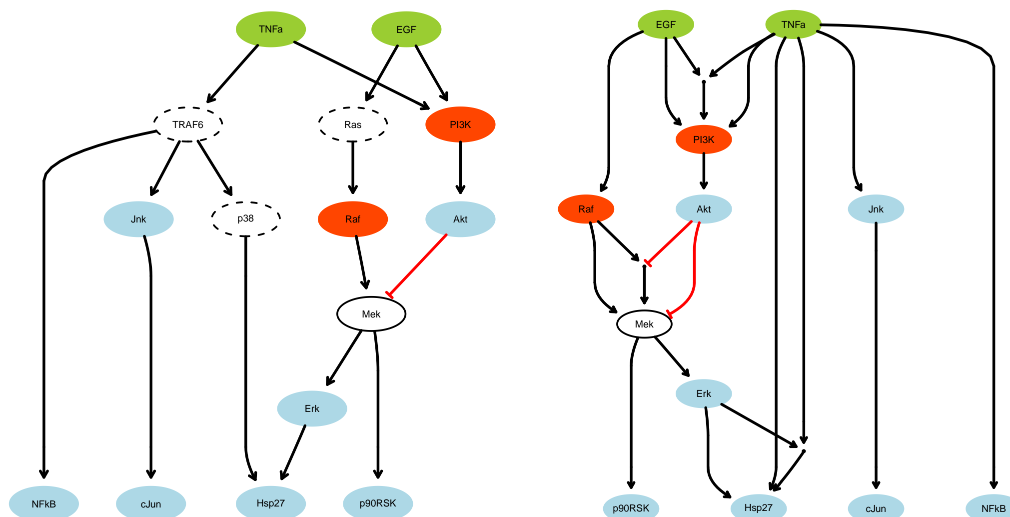

Figure 1: The original PKN model (left panel). The colors indicate the signals (green), readouts (blue) and inhibitors (red) as described by the data. The dashed white nodes shows species that can be compressed. The right panel is the compressed and expanded model as used by the analysis (See reference [1] for details).

Note that in CellNOptR version above 1.3.28, a new class called *CNOlist* is available. We strongly recommend to use it since future version of *CellNOptR* and *CNORfuzzy* will use this class instead of the list returned by *makeCNOlist*. You can easily convert existing CNOlist (like the R data set called CNOlistToy built with *makeCNOlist*).

```
data(CNOlistToy, package="CellNOptR")
CNOlistToy = CNOlist(CNOlistToy)
print(CNOlistToy)
```

```
class: CNOlist
cues: EGF TNFa Raf PI3K
inhibitors: Raf PI3K
stimuli: EGF TNFa
timepoints: 0 10
signals: Akt Hsp27 NFkB Erk p90RSK Jnk cJun
```

You can also visualise the CNOlist using the *plot* method (see Figure 2) that will produce a plot with a subplot for each signal (column) and each condition (row), and an image plot for each condition that contains the information about which cues are present (last column). See *CellNOptR* tutorial for more details [1].

```
# with the old CNOlist (output of makeCNOlist), type
data(CNOlistToy, package="CellNOptR")
plotCNOlist(CNOlistToy)
# with the new version, just type:
CNOlistToy = CNOlist(CNOlistToy)
plot(CNOlistToy)
```

Next, we set up a list of parameters that are related to (i) the genetic algorithm used in the optimisation step, (ii) the transfer functions associated to fuzzy logic formalism, (iii) a set of optimisation parameters related to the fuzzy logic. The list of parameters is also used to store the Data and Model objects. There is a function that will help you managing all the parameters, which is called *defaultParametersFuzzy* and is used as follows:

```
paramsList = defaultParametersFuzzy(CNOlistToy, ToyModel)
paramsList$popSize = 50
paramsList$maxGens = 50
paramsList$optimisation$maxtime = 30
```

In the next section, we will show how to set up the parameters more specifically. Here, we reduced the default values of some parameter to speed up the code and also because the algorithm converges quickly for this particular example.

Once we have the data, model and parameters, we can optimise the model against the data. This is done thanks to the function *CNORwrapFuzzy*. In principle, as we will see later the fuzzy approach requires to run several optimisations. Therefore, you need to loop over several optimisations before getting the final results. This is done with the following code:

```
N = 1
allRes = list()
paramsList$verbose=TRUE
for (i in 1:N){
     Res = CNORwrapFuzzy(CNOlistToy, ToyModel, paramsList=paramsList)
     allRes[[i]] = Res
 }
```

As you can see, we set *N=1* because in the case of the ToyModel a single optimisation suffices. We provide this sample code that is generic enough to be used with more complex data sets (see next section for a more complex example).

After each optimisation, the results are saved in a temporary object *Res* that is appended to a list *allRes* that stored all the results. Each variable *Res* stores the reduced and refined models [4] that are used by the *compileMultiRes* function to build up a summary:

```
summary = compileMultiRes(allRes,show=FALSE)
```

Note that we set *show=FALSE* because the resulting plot is meaningless for this example. The next section shows an example with the option *show=TRUE* and describes the resulting plot.

Finally, we produce a plot of our analysis that is similar to the one produce by the plotCNOList function (left panel in Figure 2), except that the simulated data is overlaid in dashed lines and the background color indicates the difference between the simulated and experimental data (right panel in Figure 2). The plot is generated with the following command:

```
plotMeanFuzzyFit(0.1, summary$allFinalMSEs, allRes,
     plotParams=list(cmap_scale=0.5, cex=.9, margin=0.3))
```

The next section will explain how to chose the first argument, which is arbitrary set to 0.1 in this example.

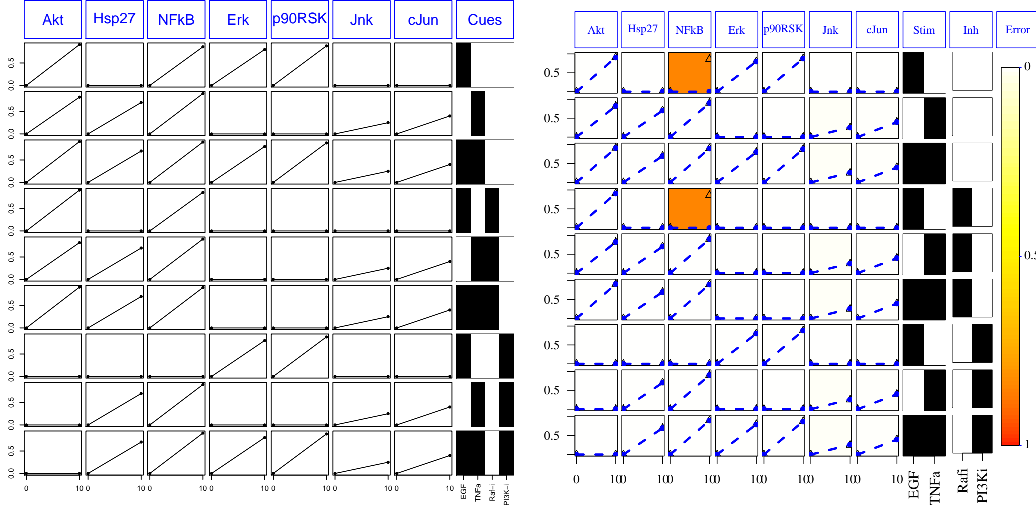

Figure 2: The actual data are plotted with *plotCNOlist* from the *CellNOptR* package (left panel). The results of simulating the data with our best model compared with the actual data are plotted with *plotMeanFuzzyFit* (right panel) where the simulated data is overlaid in dashed blue lines and the background indicates the absolute difference between model and data. The red boxes indicates a missing link as explained in [3]. The colors convention is as follows: greener=closer to 0 difference, redder=closer to 100% difference. Here the light pink boxes indicates a difference about 50%.

The results shown in Figure 2 shows a very good agreement between the final model selected by the fuzzy logic approach and the experimental data except for the case of NFkB specie. It has been found that this is related to a missing link between NFkB and PI3K species in the PKN model [3]. Yet, the overall results is better that the one obtained with the boolean approach [1, 3].

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 [5] 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 [2]. The prior knowledge network is presented in Figure 3. 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.

```
library(CNORfuzzy)
data(DreamModel, package="CellNOptR")
data(CNOlistDREAM, package="CellNOptR")
```

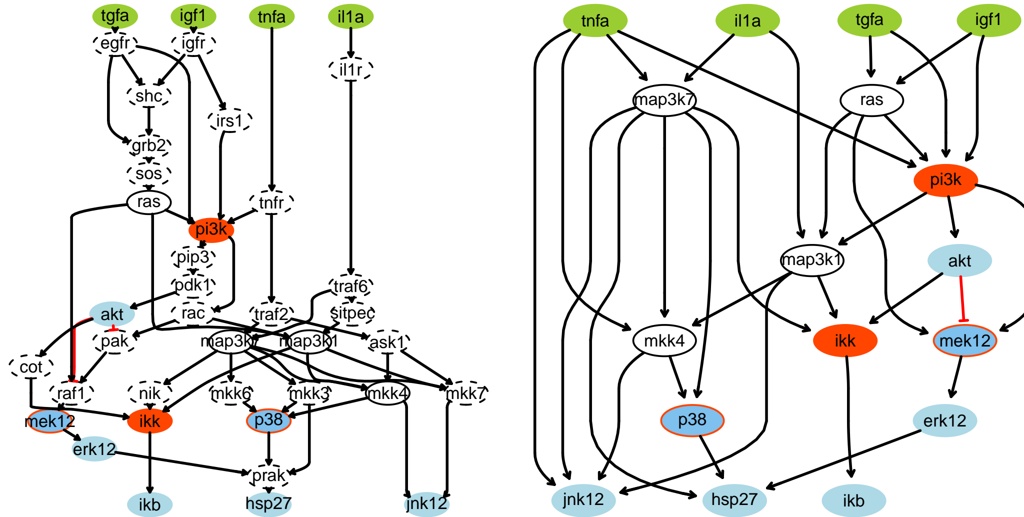

Figure 3: The left panel shows the original PKN model (DREAM data). Right panel shows the compressed and expanded model. See caption of Figure 1 for the color code.

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.

```
# Default parameters
paramsList = defaultParametersFuzzy(CNOlistDREAM, DreamModel)
# Some Genetic Algorithm parameters
paramsList$popSize      = 50
paramsList$maxTime      = 5*60
paramsList$maxGens      = 200
paramsList$stallGenMax  = 50
paramsList$verbose      = FALSE
```

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 6-a for a graphical representation).

```
# Default Fuzzy Logic Type1 parameters (Hill transfer functions)
nrow = 7
```

```
paramsList$type1Funs = matrix(data = NaN,nrow=nrow,ncol=3)
paramsList$type1Funs[,1] = 1
paramsList$type1Funs[,2] = c(3, 3, 3, 3, 3, 3, 1.01)
paramsList$type1Funs[,3] = c(0.2, 0.3, 0.4, 0.55, 0.72,1.03, 68.5098)
```

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.

```
# Default Fuzzy Logic Type2 parameters
nrow = 7
paramsList$type2Funs = matrix(data = NaN,nrow=nrow,ncol=3)
paramsList$type2Funs[,1] = seq(from=0.2, to=0.8, length=nrow)
#paramsList$type2Funs[,1] = c(0.2,0.3,0.4,0.5,0.6,0.7,0.8)
paramsList$type2Funs[,2] = 1
paramsList$type2Funs[,3] = 1
```

ReductionThresh is a list of threshold to be used during the reduction step. This vector is used for instance in Figure 4 to set the x-axis.

```
paramsList$redThres = c(0, 0.0001, 0.0005, 0.001,  0.003, 0.005, 0.01)
```

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:

```
paramsList$optimisation$algorithm = "NLOPT_LN_SBPLX"
paramsList$optimisation$xtol_abs = 0.001
paramsList$optimisation$maxeval =  10000
paramsList$optimisation$maxtime = 60*5
```

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

4.3 Analysis

Once the parameters are set, similarly to Section 2, 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.

```
N = 10
allRes = list()
for (i in 1:N){
     Res = CNORwrapFuzzy(CNOlistDREAM, DreamModel, paramsList=paramsList,
         verbose=TRUE)
     allRes[[i]] = Res
 }
summary = compileMultiRes(allRes, show=TRUE)
```

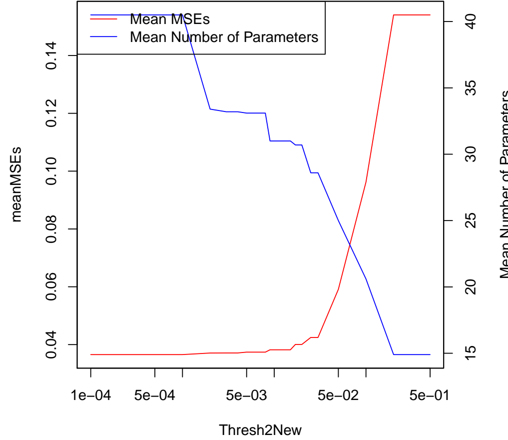

Figure 4: This plot is produced with *compileMultiRes* function. The threshold should be chosen before the mean MSE increases that is around 0.01 in this example. This threshold is used in the final analysis as an argument to the function *plotMeanFuzzyFit*.

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 4) 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 4 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:

```
plotMeanFuzzyFit(0.01, summary$allFinalMSEs, allRes)
```

and second, a threshold that would lead to a model with smaller parameters but with a larger mean MSE:

```
plotMeanFuzzyFit(0.5, summary$allFinalMSEs, allRes)
```

Results are shown in Figure 5. Finally, we can outputs some files using the following command:

```
writeFuzzyNetwork(0.01, summary$allFinalMSEs, allRes, "output_dream")
```

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*:

1. output_dream_PKN.dot a dot file representing the optimised model. Once processed with the executable *dot*), the results is shown in Figure 7.

2. output_dream_nodesPKN.NA store the nature of the nodes (compressed, signals, ...) and can be used with cytoscape.

3. output_dream_TimesPKN.EA stores the reactions and associated weights to be used in cytoscape

4. output_dream_PKN.sif

Figure 5: Simulated and experimental fit using a threshold of 0.01 (left panel) and 0.5 (right panel). See text to decide on how to choose the threshold.

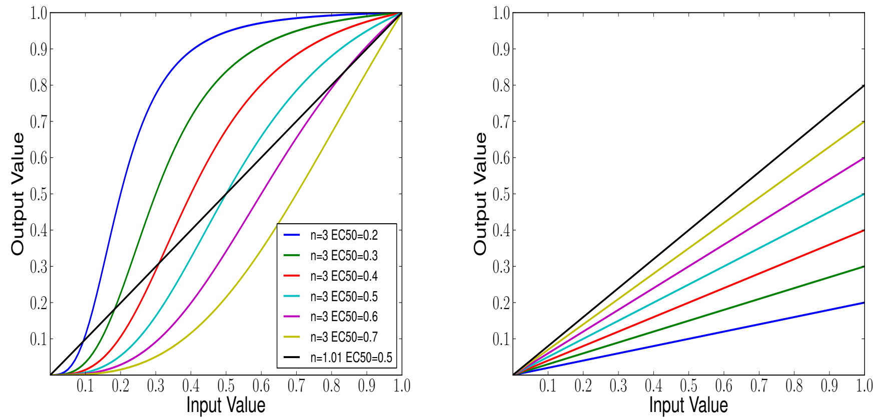

Figure 6: (a) Normalised Hill function generated using the *Type1Funs* default parameters. The default values of n are shown in the caption. The default values of k are chosen so that is correspond to the EC50 values shown in the caption. (b) Transfer function generated using the *Type2Funs* default parameters.

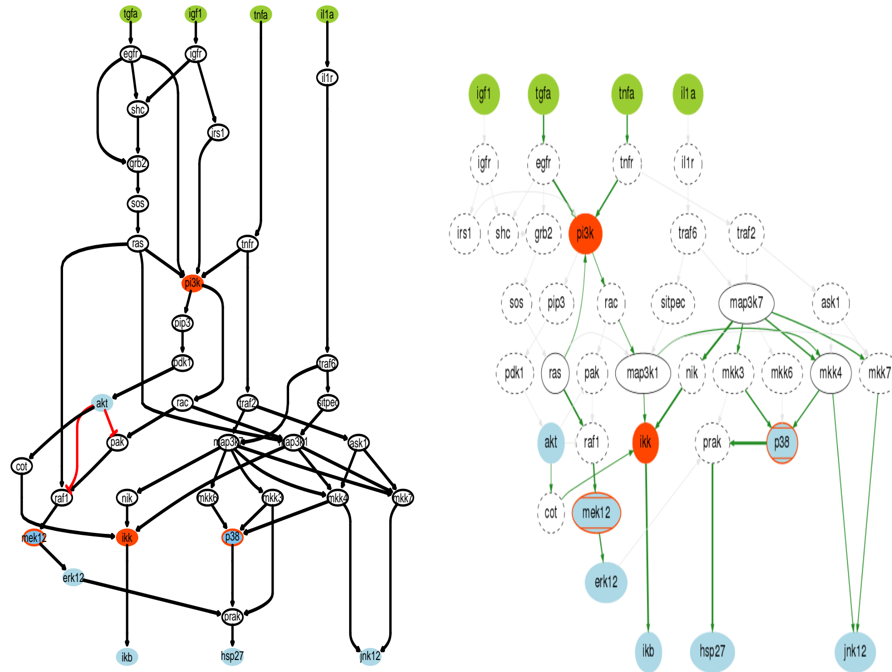

Figure 7: When calling *writeFuzzyNetwork*, the original model is saved in a new file (.sif file in the left panel) as well as the optimised model on the data resulting from the analysis that is saved in a .dot file (right panel) where grey edges means no link. Note that the right panel wrongly labelled some edges in grey (e.g., no link out of IL1a; this is related to a current issue in the mapback step of the optimised model on the original one, which should be fixed in a future version). Other files useful to import in CytoScape are also saved. See text for details.

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), 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)