Integrating literature-constrained and data-driven inference of signalling networks with *CNORfeeder*
(version 1.0.0 and above)

November 1, 2022

Contents

1 Overview

This vignette shows how to install and use CNORfeeder. CNORfeeder is an R package wich allows to integrate information derived from prior knowledge about signalling networks, with information derived strictly from data using reverse engineering methods. The aim is to overcome limitations due to links missing in the prior knowledge becuase of uncompleteness of databases or lack of biological knowledge about certain processes. In particular, CNORfeeder is developed to be combined with the Bioconductor package CellNOptR which allows to train a prior knowledge network to data in order to obtain the logic model with best fit the data. The integration of literature-constrained and data-driven inference of signalling networks allows also to overcome limitations of strictly data-driven reverse-engineering methods, which are generally of limited biological interpretability being restricted to measured and perturbed nodes, thus overlooking the rest of the network. The name CNORfeeder refers to the ability of the package to feed the prior knowledge network in a combined pipeline with CellNOptR (also called CNOR).

2   Installation

This software is written in the R language, so in order to use it you will need to have R installed on your computer. For more information and download of R, please refer to http://www.r-project.org/. For more information about how to install R packages, please refer to http://cran.r-project.org/doc/manuals/R-admin.html#Installing-packages. This package relies on several Bioconductor packages (CellNOptR, RBGL, graph, minet) and CRAN packages (igraph catnet). The following code installs the *CNORfeeder* dependencies and suggested packages that are on Bioconductor website, open an R session and type:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("RBGL","graph","minet","CellNOptR","igraph","catnet"))
```

Before starting this tutorial you also need to install the package *CNORfeeder*. You can either install *CNORfeeder* from Bioconductor by typing:

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

or from a tar ball as follows:

```
> install.packages("path_to_CNORfeeder/CNORfeeder_1.0.0.tar.gz",
+     repos=NULL, type="source")
```

or, using the R GUI by clicking on "Packages & Data" then "Package installer", then choosing "local source" from the dropdown menu, clicking "install", choosing *CNORfeeder_1.0.0.tar.gz* and finally clicking "open".

A series of books about R can be found on the R project website (http://www.r-project.org/), and many tutorials are available on the internet. If you are a complete beginner, all you need to know is that by typing "?nameOfFunction" you get the help page about the function that you are interested in.

3   Introduction

The package *CNORfeeder* permits to extend a network derived from literature with links derived strictly from the data via various inference methods using information on physical interactions of proteins to guide and validate the integration of links, as described in [1]. The package is designed to be integrated with *CellNOptR*, a package described in [4] and based on methods described in [3] (see http://www.cellnopt.org for further details on the project). The integrated pipeline is illustrated in Figure 1 where steps performed by *CNORfeeder* are represented in cyan boxes and the ones performed by *CellNOptR* are in magenta boxes.

A Data are used to infer a strictly data-driven network (DDN) using reverse-engineering methods (as for now FEED, ARACNe, CLR and Bayesian networks are implemented);

B the prior knowledge network (PKN) is compressed according to the data (green, red and blue nodes are respectively stimulated, inhibited and measured), removing non-identifiable nodes (dashed ones);

C the compressed network is integrated with the DDN (blue links are obtained from the DDN and black ones from the PKN);

D information derived from protein-protein interaction network (PIN) is used to support and prioritize integrated links;

E the integrated network is used as input for the training: in trained model, thick black lines denote interactions (and gates) in final model, and light-grey links present in integrated network but not in trained model.
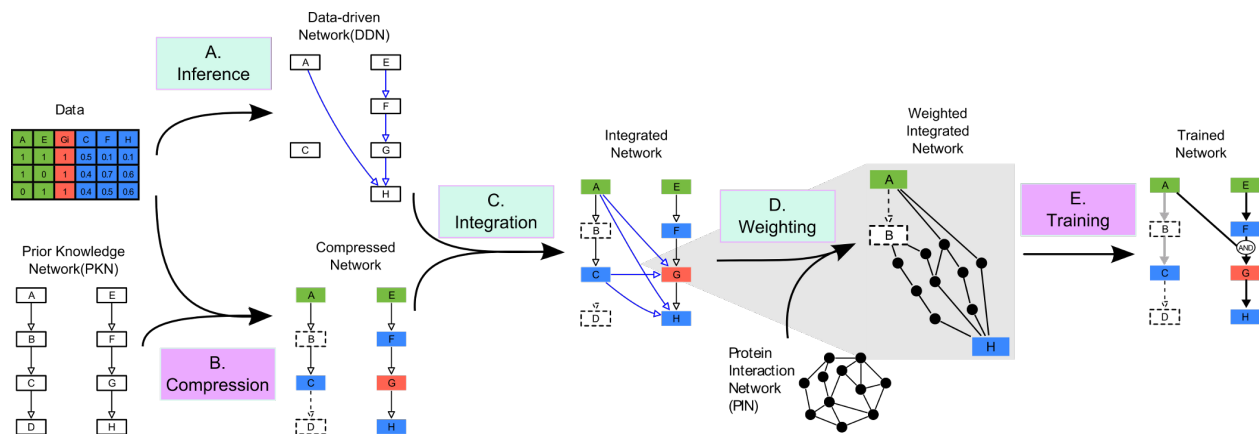
Figure 1: Integrated *CNORfeeder* - *CellNOptR* pipeline. Figure taken from [1].

4   Example

We will illustrate the use of package *CNORfeeder* showing, step by step, the analysis described in [1]. It is performed on a real data set, which is a part of the network analysed in [3] and comprises 40 species and 58 interactions in the PKN. This network was also used for the signaling challenge in DREAM 4 (see http://www.the-dream-project.org/). The associated data was collected in hepatocellular carcinoma cell line HepG2 (see [2]). Phosphorylation of seven proteins (akt, erk12, ikb, jnk12, p38, hsp27, mek12) is measured 30 min after combinatorial stimulation with four ligands (tnfa, il1a, igf1, tgfa) and four inhibitors (pi3k, ikk, p38, mek12). The level of phosphorylation of proteins is measured using the Luminex xMAP assay and provides a value of the phosphorylation in arbitrary units, that can be used to compare values at two conditions. In our case we compare the values between 0 and 30, and this change is a proxy of the induced activation of the corresponding protein. The normalization of this data to a value between 0 and 1 is achieved using a method based on a set of thresholds as described in [3].

First of all we load the library

```
> library(CNORfeeder)
```

and the data

```
> # load the data already formatted as CNOlist
> data(CNOlistDREAM,package="CellNOptR")
> # load the model (PKN) already in the CNO format
> data(DreamModel,package="CellNOptR")
```

For additional information about how to read the data see *CellNOptR* documentation.

4.1   A. Inference - CNORfeeder

Different methods can be used to reverse-engineer a network purely from data. CNORfeeder includes a native method, FEED, developed specifically to infer signalling network from perturbation experiments (that is an improved version of the method described in [5]). Since we believe that different inference methods can provide complementary insights into the same problem, other three approaches have been imported from other packages: two methods based on mutual information (i.e. CLR and ARACNE from the R package *minet*) and one based on Bayesian networks (from the R package *catnet*).

The three functions provided to infer a network strictly from data are the following:

1. *makeBTables* can be used to infer a cause-effect network interpreted as Boolean tables using the method FEED [5];

2. *MIinference* allows to use the R package *minet* ([6]) to infer networks based on mutual information, i.e. ARACNE and CLR methods;

3. *Binference* allows to use the R package *catnet* (http://cran.r-project.org/web/packages/catnet/index.html) to derive Bayesian networks;

In this example, the FEED method (descirbed in [5]) is applied. The inferred DDN is encoded in Boolean tables. For each protein, a Boolean table is inferred having one columns for each stimulus and one row for each inhibitor. If a stimulus produces a significant effect on the activity level of the protein this is codified with a 1 in the corresponding column, if also the inhibitor affects the protein there is a 2 in the corresponding cell. The sign of the regulation is coded in separate tables. Significance is assessed taking into account the experimental error. Parameter k is the parameter which determines the threshold of significancy of the effect of stimuli and inhibitors, being multiplied by the measurement error in order to assess the relevance of a link. Tuning parameter k allows to adjust the sparsity of the network.

```
> BTable <- makeBTables(CNOlist=CNOlistDREAM, k=2, measErr=c(0.1, 0))
```

Please note that *makeBTables* can handle multiple time-point data and links are added if their effect is significant in at least one time point. As for *MIinference* and *Binference* only one steady state data shoud be provided for each experimental condition. The resulting network is shown in Figure 2, upper panel. FEED method also allows to rank links according to the upper limit value of parameter k allowing the presence of the link.

```
> Lrank <- linksRanking(CNOlist=CNOlistDREAM, measErr=c(0.1, 0), savefile=FALSE)
```

4.2 B. Compression - CellNOptR

The preprocessing step includes the compression of the network to exclude non-identifiable nodes; the full details of preprocessing the model can be found in the *CellNOptR* package (see vignette for comprehensive explanation).

```
> model<-preprocessing(data=CNOlistDREAM, model=DreamModel)
```

4.3 C. Integration - CNORfeeder

The inferred DDN is used to integrate the PKN in order to include links that are missing in the prior knowledge but seems to be supported by data. We have to take into account that networks inferred directly from data (DDNs) are limited only to measured, inhibited and stimulated nodes, thus a link in the DDN can correspond to multiple links in th PKN.

```
> modelIntegr <- mapBTables2model(BTable=BTable,model=model,allInter=TRUE)
```

```
> modelIntegr$reacID[modelIntegr$indexIntegr]
```

The resulting network model can be plotted as follows (Figure 2, lower panel), using the *indexIntegr* argument to highlight in purple integrated links:

```
> plotModel(model=modelIntegr, CNOlist=CNOlistDREAM, indexIntegr=modelIntegr$indexIntegr)
```
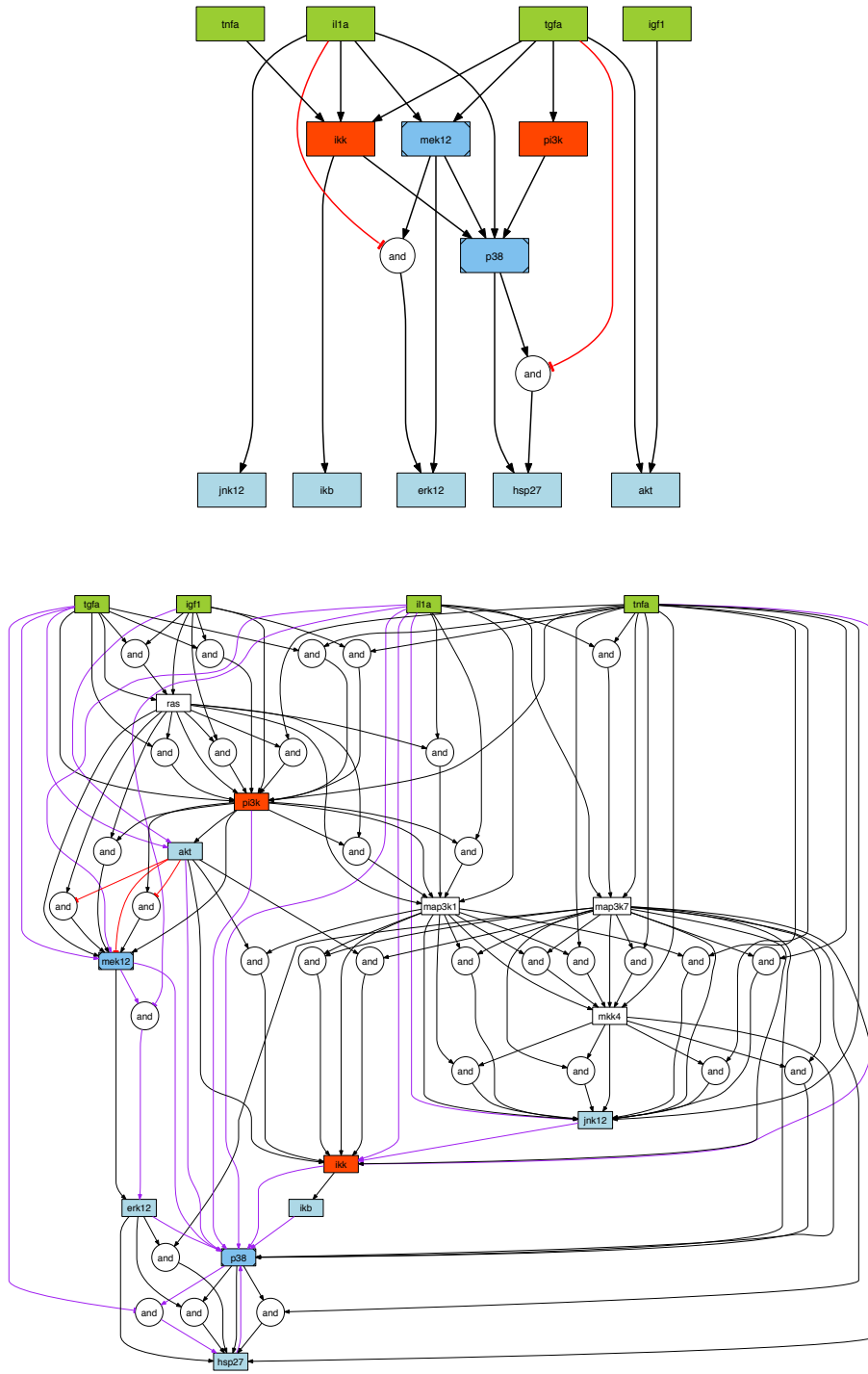
Figure 2: *Upper panel*: data-drive network (DDN) inferred using FEED method. *Lower panel*: integrated network, the prior knowledge network is intepreted as a logic model with all possible AND and OR gates, links added based on data-driven reverse-engineering methods are highlighted in purple.

4.4   D. Weighting - CNORfeeder

Links inferred based only on data are, in principle, less reliable than those derived from prior knowledge, thus we allow to weight added links differently. The attribute *integrFac* is the multiplicative factor to differently prioritize integrated links during the optimization (next step). A higher value of *integrFac* forces the genetic algorothm used for optimization, to strongly prioritize links from the PKN and to limit the choice of integrated links, even if their presence in the network model improves the fit of the model to the data. If *integrFac* is set equal to 1, all links have the same weight and the optimization algorithm will select those links which minimize the fit, disregarding if they are derived from data or from prior knowledge.

```
> modelIntegrWeight <- weighting(modelIntegr=modelIntegr, PKNmodel=DreamModel,
+                                CNOlist=CNOlistDREAM, integrFac=10)
```

This function return the same model with an additional field that is the weight assigned to each link, thus preparing it for the optimization (see next section).

Information derived from protein-protein networks (PIN) can also be used to differently prioritize integrated links, with the basic idea that a shorter path in the PIN corresponds to a more reliable link in the PKN. A PIN (PPINigraph) is provided with the package and can be passed to the *PPI* attribute, as shown in the example, in order to differently weight links (note that Uniprot IDs must be provided, see *UniprotIDdream* as an example of the list structure to be used).

```
> data(PPINigraph,package="CNORfeeder")
> data(UniprotIDdream,package="CNORfeeder")
> modelIntegrWeight <- weighting(modelIntegr=modelIntegr, PKNmodel=DreamModel,
+                                CNOlist=CNOlistDREAM, integrFac=10,
+                                UniprotID=UniprotIDdream, PPI=PPINigraph)
```

Users can include their own interaction network passing it as argument *PPI* to the function *weighting*. The interaction network should be passes as an igraph where proteins are nodes and edges represent the interactions (see *weighting* manual for further details).

4.5   E. Training - CellNOptR

The integrated network can be optimized using *CellNOptR*; the function *gaBinaryT1W* of *CNORfeeder* package is a slightly modified version of *gaBinaryT1* function of package *CellNOptR* in order to take into account the different weigths that can be assigned to each link. A bipartite objective function is used to balance fit and size, that is to find the model which best describe data with the minumum number of links (see [3] and the vignette of *CellNOptR* for a comprehensive description).

```
> # training to data using genetic algorithm (run longer to obtain better results)
> DreamT1opt<-gaBinaryT1W(CNOlist=CNOlistDREAM, model=modelIntegrWeight,
+                         maxGens=2, popSize=5, verbose=FALSE)
```

The optimal model and the simulated and real data (shown in Figure 5) can be visualized using the following commands of *CellNOptR*:

```
> # model
> plotModel(model=modelIntegrWeight, CNOlist=CNOlistDREAM, bString=DreamT1opt$bString)
> # data
> cutAndPlotResultsT1(model=modelIntegrWeight, CNOlist=CNOlistDREAM,
+                     bString=DreamT1opt$bString)
```

When the integrated network is trained to the data, the model which best describes the data with the minimum number of links is selected. Integrated links which improves the fit to the data with respect to the PKN alone are plausible links that where missing in the PKN. The functionality of this connections could be further experimentally tested, for example, using methods as RNA interference to knock out the gene corresponding to the upstream protein and looking at the effect on the downstream protein.
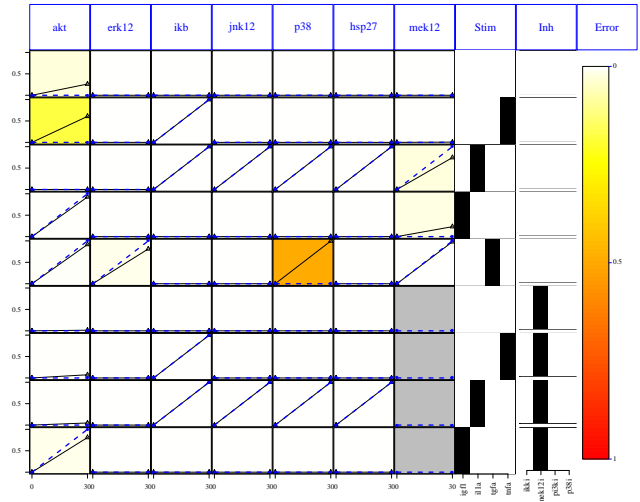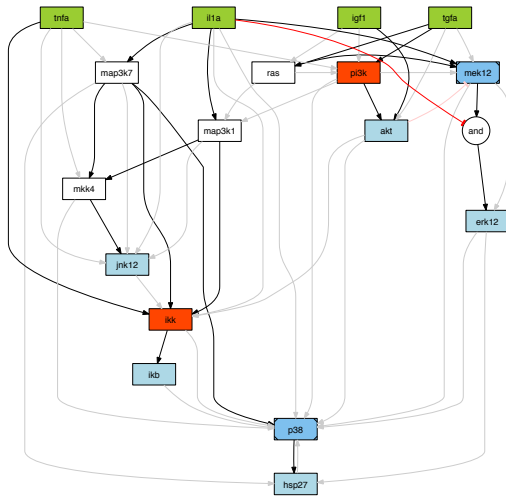
Figure 3: *Left panel*: optimal model. *Right panel*: fit to experimental data.
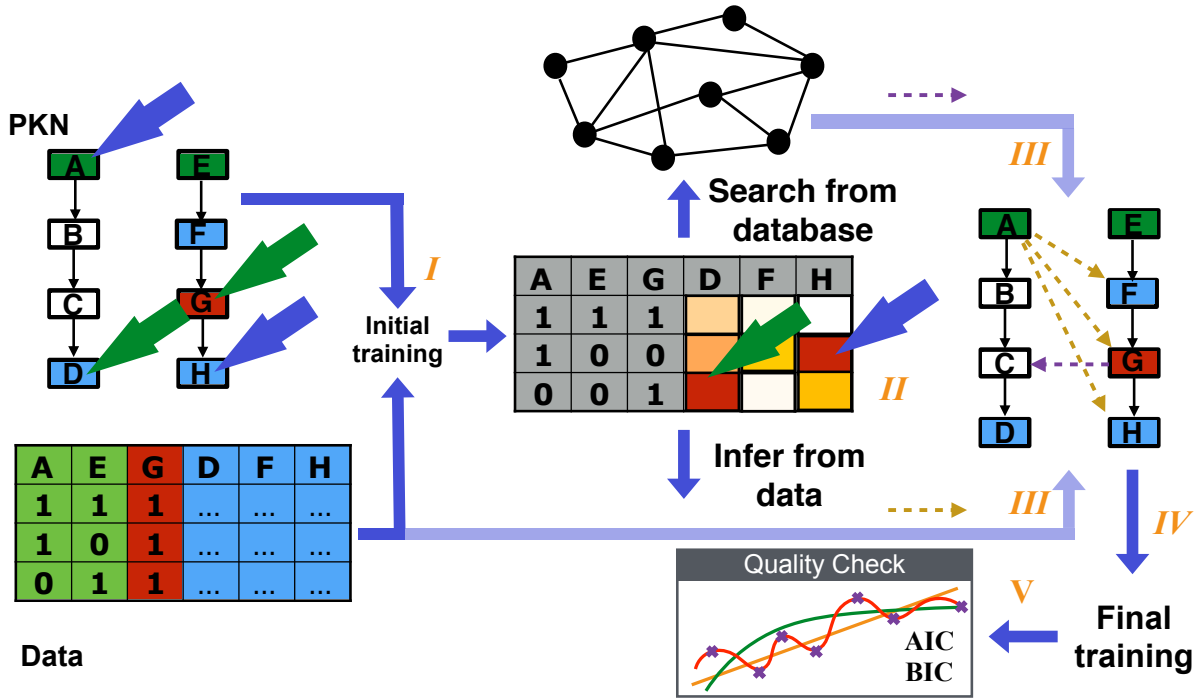
Figure 4: *Dynamic-Feeder* pipeline.

5 Dynamic Feeder

Dynamic-Feeder represents an extension of the CNORfeeder implementation described in the chapters above by inferring new possible functional interactions missing in the PKN for dynamic models casted as logic-based Ordinary Differential Equations (ODEs) from time-series data.

Through the figure below 4, we show a step by-step summary of the Dynamic-Feeder pipeline and apply it to a small toy example [7] which was generated manually based on the network contained in [8]. This network was designed to output a variety of dynamics (oscillations, transience etc.).

5.1 I. Loading the toy example and performing the initial ODE analysis

Loading the necessary packages and the toy example containing 16 interactions and 15 species (of which 2 are stimulated, 2 are inhibited and 7 are measrued). Data was generated in-silico across each measuremnt at each perturbation experimental condition for 3 time-points (0, 10, 30).

```
> # loading the necessary packages
> library(CellNOptR)
> library(MEIGOR)
> library(CNORode)
> library(doParallel)
> library(readr)
> library(infotheo)
> library(igraph)
> library(OmnipathR)
> library(CNORfeeder)
> # loading the model
> data(ToyModel_Gene, package="CNORfeeder")
> # loading the data
> data(CNOlistToy_Gene, package="CNORfeeder")
> # plotting the model and the data
> plotModel(model = model, CNOlist = cnolist)
> plotCNOlist(CNOlist = cnolist)
```
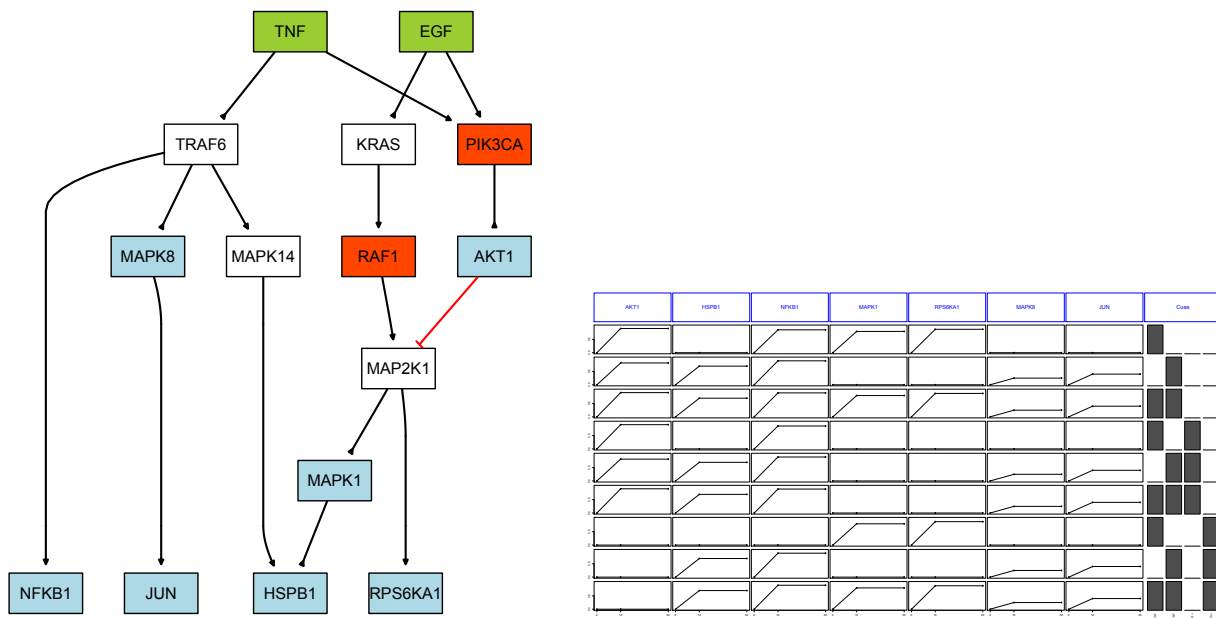

Figure 5: *Left panel*: initial model. *Right panel*: in-silico data.

Optionally, the user can load pathway resources. Below, we show how we can load from *OmniPath* [9] (by using *OmniPathR* package) interactions from three other resources: *SignaLink3*, *PhosphoSite* and *Signor*. We only retain those directed interactions which either are activatory or inhibitory. Besides the resource, users can choose to have interactions based on number of references, resources, etc. For more about this, please visit https://github.com/saezlab/OmnipathR.

```
> ## Loading database
> data(database, package="CNORfeeder")
> ## Alternatively, users can download the database from Omnipath and retain only
> ## activatory/inhibitory interactions as shown below.
> ## Please note that OmniPath is in continuous update and new links are
> ## continuously added
>
> # interactions <- import_Omnipath_Interactions(filter_databases=
> #                                                  c("SignaLink3",
> #                                                    "PhosphoSite",
> #                                                    "Signor"))
> # interactions = interactions[which(interactions$is_stimulation+interactions$is_inhibition==1), ]
> # database = matrix(data = , nrow = nrow(interactions), ncol = 3)
> # database[, 1] = interactions$source_genesymbol
> # database[which(interactions$is_stimulation==1), 2] = "1"
> # database[which(interactions$is_inhibition==1), 2] = "-1"
> # database[, 3] = interactions$target_genesymbol
```

Now, we perform the initial ODE analysis and generate the simulation results of our optimized model:

```
> # set initial parameters (here parameters k and tau are optimised and n fixed to 3)
> ode_parameters=createLBodeContPars(model, LB_n = 1, LB_k = 0,
+                                     LB_tau = 0, UB_n = 3, UB_k = 1,
+                                     UB_tau = 1, default_n = 3,
+                                     default_k = 0.5, default_tau = 0.01,
+                                     opt_n = FALSE, opt_k = TRUE,
+                                     opt_tau = TRUE, random = TRUE)
> ## Parameter Optimization
> # essm
> paramsSSm=defaultParametersSSm()
> paramsSSm$local_solver = "DHC"
> paramsSSm$maxtime = 60;
> paramsSSm$maxeval = Inf;
> paramsSSm$atol=1e-6;
> paramsSSm$reltol=1e-6;
> paramsSSm$nan_fac=1000;
> paramsSSm$dim_refset=30;
> paramsSSm$n_diverse=1000;
> paramsSSm$maxStepSize=Inf;
> paramsSSm$maxNumSteps=10000;
> paramsSSm$transfer_function = 4;
> paramsSSm$lambda_tau=0.1
> paramsSSm$lambda_k=0.01
> paramsSSm$bootstrap=F
> paramsSSm$SSpenalty_fac=0
> paramsSSm$SScontrolPenalty_fac=0
> ## Training of the initial model
> opt_pars=parEstimationLBode(cnolist, model, method="essm",
+                             ode_parameters=ode_parameters, paramsSSm=paramsSSm)
> simData = plotLBodeFitness(cnolist = cnolist, model = model,
+                             ode_parameters = opt_pars, transfer_function = 4)
```

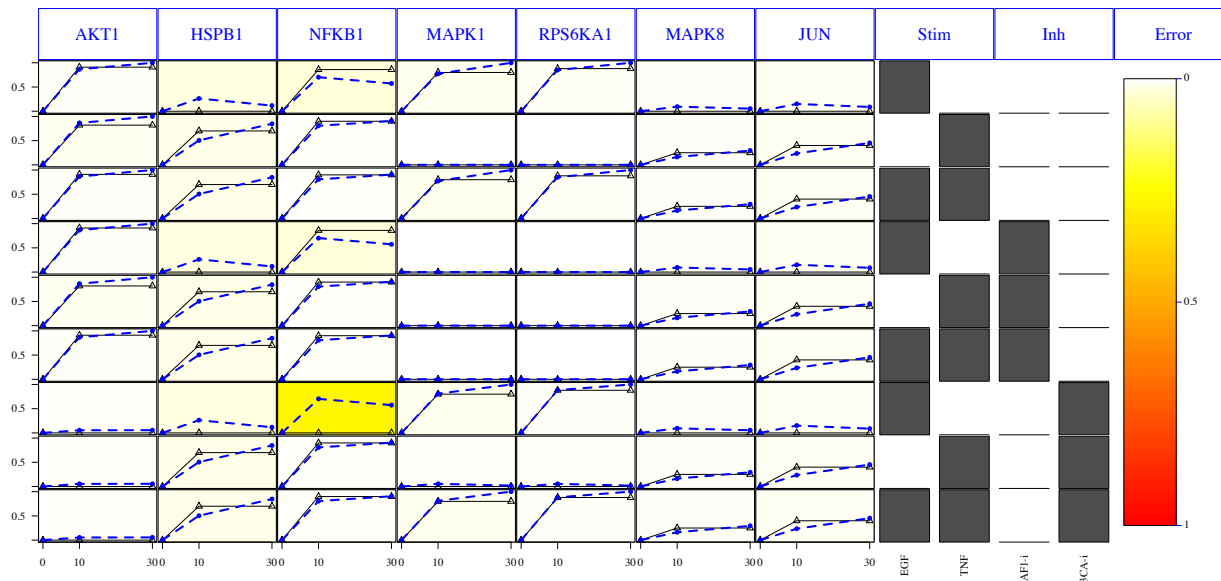The figure below 6 shows how our optimized model fits the data.

Figure 6: *Initial Fit*

5.2   II. Identifying poorly-fitted measurements

As we can notice, our model performs worse for the NFKB1 measurement when EGF and PIK3CA are being perturbed. As a next step, we then try to identify poorly-fitted measurements. A measurement is considered as poorly-fitted on a specific experimental condition, if the deviation between experimental data and prediction of the initial model is higher than a threshold parameter *mseThresh* which can be controlled by the user. On this case we have:

```
> # Identifying the mis-fits (measurements with mse worse than 0.05)
> indices = identifyMisfitIndices(cnolist = cnolist, model = model,
+                                  simData = simData, mseThresh = 0.05)
```

The *indices* object which is returned represents a list of indices telling which measurement and at which experimental condition is poorly predicted by our model. It also contains an *mse* matrix indicating the misfit between our initial model predictions for each observation (quantifide by the Mean Squared Error value).

Alternatively, users can bypass the initial training of the model and add new links through the FEED or database search regardless how well somee measurements were already fitted simply by setting sim-Data=NULL. In this case:

```
> # Identifying the mis-fits (measurements with mse worse than 0.05)
> indices = identifyMisfitIndices(cnolist = cnolist, model = model, simData = NULL)
```

5.3   III. Integration

After identifying the poorly-fitted measurements, we then try to find possible links between corresponding cues and measurements and which can potentially identify mising mechanisms which yielded the poor prediction. The user can either do a search from the *database* object, derivel the links from data-alone based on the *FEED* algorithm, or both. Below we show an example about how we can infer interactions from both approaches and show the new expanded PKN with newly integrated links colored in purple 7:

```
> # interactions from the database and from the FEED algorithm which we want to integrate
> feederObject = buildFeederObjectDynamic(model = model, cnolist = cnolist,
+                                         indices = indices, database = database,
+                                         DDN = TRUE, pathLength = 2) # max path-length=2
>                                                                    # for database search
>
> integratedModel = integrateLinks(feederObject = feederObject, cnolist = cnolist,
+                                  database = database)
> plotModel(model = integratedModel$model, CNOlist = cnolist,
+           indexIntegr = integratedModel$integLinksIdx)
```

5.4 IV. Final fit

Finally, we then optimize the integrated PKN to data and observe the effects of the new links to the fit. In order to avoid overfitting, we penalize the new links with factors higher than the interactions which were already present on the PKN. We show this for two cases:

1. By applying moderate penalty factors over the new links (200 times higher for interactions coming from *FEED* [1] and 10 times higher for interactions coming from *database* compared to the weight applied over the interactions in the PKN) and nodes (10 times higher copared to the weight for the nodes present in the PKN) 8:

```
> # interactions from the database and from the FEED algorithm which we want to integrate
> ode_parameters=createLBodeContPars(integratedModel$model, LB_n = 1,
+                                     LB_k = 0, LB_tau = 0, UB_n = 3,
+                                     UB_k = 1, UB_tau = 1, default_n = 3,
+                                     default_k = 0.5, default_tau = 0.01,
+                                     opt_n = FALSE, opt_k = TRUE,
+                                     opt_tau = TRUE, random = TRUE)
> res1 = runDynamicFeeder(cnolist = cnolist, integratedModel = integratedModel,
+                         ode_parameters = ode_parameters, paramsSSm = paramsSSm,
+                         penFactor_k = 2, penFactorPIN_k = 0.1, penFactor_tau = 1)
> plotLBodeFitness(cnolist = res1$CNOList, model = res1$`Integrated-Model`$model,
+                  ode_parameters = res1$Parameters, transfer_function = 4)
```

2. By applying very high penalty factors over the new links and nodes compared to the weights assigned to the species in the PKN 9:

```
> # interactions from the database and from the FEED algorithm which we want to integrate
> ode_parameters=createLBodeContPars(integratedModel$model, LB_n = 1,
+                                     LB_k = 0, LB_tau = 0, UB_n = 3,
+                                     UB_k = 1, UB_tau = 1, default_n = 3,
+                                     default_k = 0.5, default_tau = 0.01,
+                                     opt_n = FALSE, opt_k = TRUE,
+                                     opt_tau = TRUE, random = TRUE)
> res2 = runDynamicFeeder(cnolist = cnolist, integratedModel = integratedModel,
+                         ode_parameters = ode_parameters, paramsSSm = paramsSSm,
+                         penFactor_k = 10000, penFactorPIN_k = 10000,
+                         penFactor_tau = 10000)
> plotLBodeFitness(cnolist = res2$CNOList, model = res2$`Integrated-Model`$model,
+                  ode_parameters = res2$Parameters, transfer_function = 4)
```

We notice that for the case when we apply higher penalty factors, their effect over the overall fit get attenuated as it would be expected.
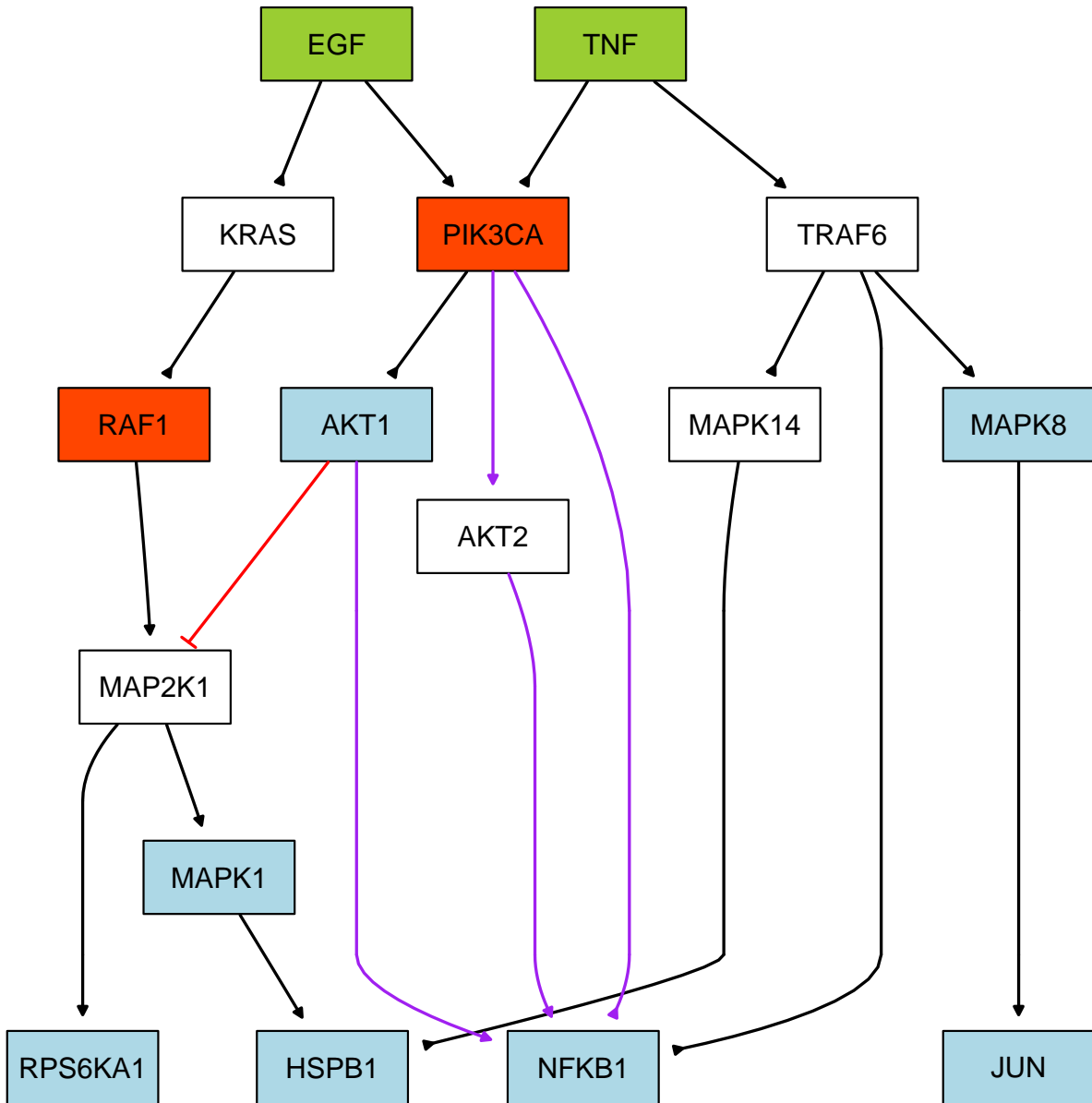
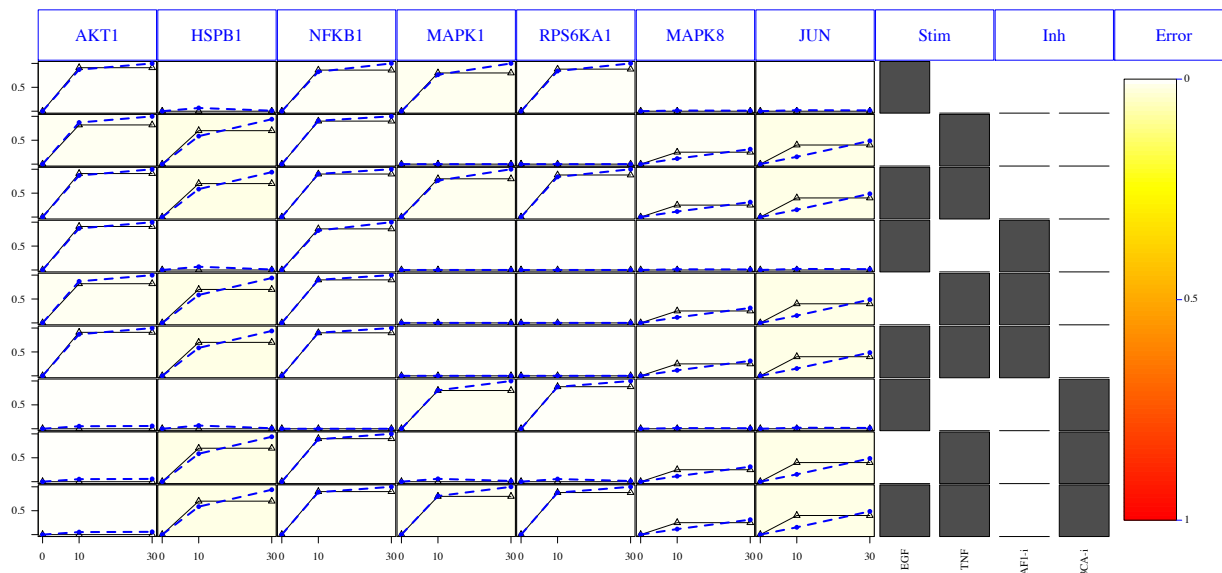Figure 7: *Integrated Model*, newly integrated links shown in purple
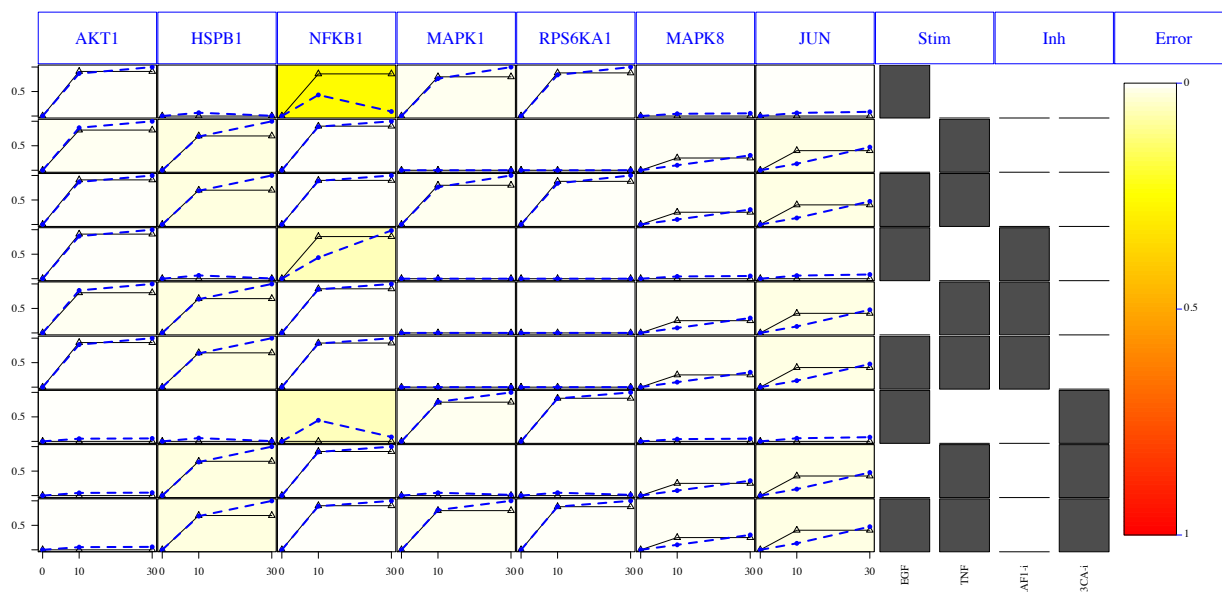
Figure 8: Effects of the new links to the PKN

Figure 9: Effects of the new links to the PKN

14

References

[1] F. Eduati, J. De Las Rivas, B. Di Camillo, G. Toffolo, J. Saez-Rodriguez. Integrating literature-constrained and data-driven inference of signalling networks. *Bioinformatics*, 28:18, 2311–2317, 2012.

[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*, 9:9, 1849–1865, 2010.

[3] J. Saez-Rodriguez, L. Alexopoulos, J. Epperlein, R. Samaga, D. Lauffenburger, S. Klamt, 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.

[4] C. Terfve, T. Cokelaer, D. Henriques, A. MacNamara, E. Goncalves, M. K. Morris, M. van Iersel, D. A. Lauffenburger, J. Saez-Rodriguez. CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. *BMC Syst Biol*, 6:133, 2012.

[5] F. Eduati, A. Corradin, B. Di Camillo, G. Toffolo. A Boolean approach to linear prediction for signaling network modeling. *PLoS ONE*, 5(9): e12789, 2010.

[6] P. E. Meyer, F. Lafitte, G. Bontempi MINET: An open source R/Bioconductor package for mutual information based network inference. *BMC bioinformatics*, 9(1), 2008.

[7] A. MacNamara, C. Terfve, D. Henriques, B.P. Bernabe, J. Saez-Rodriguez State-time spectrum of signal transduction logic models Physical Biology. *Phys Biol.*, 9(4):045003, 2012.

[8] J. Saez-Rodriguez, L.G. Alexoupulos, M. Zhang, M.K. Morris, D.A. Lauffenburger, P.K. Sorger Comparing signaling networks between normal and transformed hepatocytes using discrete logical models. *Cancer Res.*, 15;71(16):5400-11, 2011.

[9] D. Turei, T. Korcsmaros, J. Saez-Rodriguez. OmniPath: guidelines and gateway for literature-curated signaling pathway resources. *Nature Methods*, 13, pages966 967 2016.