CellNOptR

March 24, 2012

`CNORwrap` *CNOR analysis wrapper*

Description

This function is a wrapper around the whole CNOR analysis, it performs the following steps: 1. Plot the CNOlist; 2. Checks data to model compatibility; 3. Find the indices, in the model, of the species that are inhibited/stimulated/measured (signals); 4. Find the indices of the non-osbervables/non-controllables (nonce); 5. Cut the nonc off the model; 6. Recompute the indices; 7. Compress the model;8. Recompute the indices; 9. Expand the gates; 10. Compute the residual error; 11. Prepare for simulation; 12. Optimisation; 13. Plot simulated and experimental results; 14. Plot the evolution of fit; 15. Write the scaffold and PKN; 16. Write the report

Usage

```
CNORwrap(paramsList,Data,Model,Name,NamesData,Time=1)
```

Arguments

`paramsList`	paramsList has entries: Data:the CNOlist, Model:the model; Parameters for the optimisation:sizeFac: default to 1e-04; NAFac: default to 1;PopSize: default to 50; Pmutation: default to 0.5; MaxTime: default to 60; maxGens: default to 500; StallGenMax: default to 100; SelPress: default to 1.2; elitism: default to 5; RelTol: default to 0.1; verbose: default to FALSE (default to true in the functions used by CNORwrap but CNORwrap sets them to false by default).
`Data`	the CNOlist that contains the data that you will use
`Model`	the model that you want to optimise
`Name`	a string that will be used to name the project and all graphs produced
`NamesData`	a list with two elements:CNOlist and Model, each containing a string that is a reference for the user to know which model/data set was used (it will be included in the report)
`Time`	either 1 or 2: by default this is set to 1 because the 2 time points optimisation is not implemented in this version

Details

If you do not provide a parameters list, you can provide only essential elements, and all other parameters will be set to their default values. In this case, you should set paramsList=NA, and provide the following fields: Data, Model, Name, Time.

Value

This function does not return anything, it does the analysis, produces all the plots and puts them in a folder that is in your working directory, and is called "Name".

Author(s)

C. Terfve

Examples

```
#version with paramslist

tmpdir<-tempdir()
setwd(tmpdir)

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

pList<-list(
Data=CNOlistToy,
Model=ToyModel,
sizeFac = 1e-04,
NAFac = 1,
PopSize = 5,
Pmutation = 0.5,
MaxTime = 60,
maxGens = 5,
StallGenMax = 5,
SelPress = 1.2,
elitism = 5,
RelTol = 0.1,
verbose=TRUE)
CNORwrap(
paramsList=pList,
Name="Toy",
NamesData=list(CNOlist="ToyData",Model="ToyModel"),
Data=NA,
Model=NA)

## Not run:
#version with default parameters

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

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

CNORwrap(
paramsList=NA,
```

```
Name="Toy",
NamesData=list(CNOlist="ToyData",Model="ToyModel"),
Data=CNOlistToy,
Model=ToyModel)

## End(Not run)
```

`CNOlistDREAM` *Data used for the DREAM3 challenge*

Description

This data object contains the DREAM data used in the package vignette, already loaded and formatted as a CNOlist object. This is to be used with the model "DreamModel". This is a data collected on HepG2 cells cultivated with or without stimulation of tgfa, ilk, mek12, pi3k and p38, in combination with inhibition of igf1 and/or il1a. Seven phosphoproteins are measured using Luminex xMAP assays: akt, erk12, ikb, jnk12, p38, hsp27 and mek12.

Usage

```
CNOlistDREAM
```

Format

CNOlistDREAM is a list with the fields "namesCues" (character vector), "namesStimuli" (character vector), "namesInhibitors" (character vector), "namesSignals" (character vector), "timeSignals" (numerical vector), "valueCues" (numerical matrix), "valueInhibitors" (numerical matrix), "valueStimuli"(numerical matrix), "valueSignals"(numerical matrix).

Source

This data and model is extracted from the Matlab version of CellNOpt1.0 (http://www.ebi.ac.uk/saezrodriguez/software.html#CellNetOptimizer).

References

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

Prill RJ, Marbach D, Saez-Rodriguez J, Sorger PK, Alexopoulos LG, Xue X, Clarke ND, Altan-Bonnet G, and Stolovitzky G. Towards a rigorous assessment of systems biology models: the DREAM3 challenges. PLoS One, 5(2):e9202, 2010.

CNOlistToy *Toy data*

Description

This data object contains the data associated with the Toy Model example from the package vignette, already loaded and formatted as a CNOlist object.

Usage

```
CNOlistToy
```

Format

CNOlistToy is a list with the fields "namesCues" (character vector), "namesStimuli" (character vector), "namesInhibitors" (character vector), "namesSignals" (character vector), "timeSignals" (numerical vector), "valueCues" (numerical matrix), "valueInhibitors" (numerical matrix), "valueStimuli"(numerical matrix), "valueSignals"(numerical matrix).

Source

This data and model is extracted from the Matlab version of CellNOpt1.0 (http://www.ebi.ac.uk/saezrodriguez/software.html#CellNetOptimizer).

References

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

CellNOptR-package *R version of CellNOptR, boolean features*

Description

This package does optimisation of boolean logic networks of signalling pathways based on a previous knowledge network and a set of data collected upon perturbation of some of the nodes in the network.

Details

Package:	CellNOptR
Type:	Package
Version:	0.99.6
Date:	2011-10-22
License:	Artistic-2.0
LazyLoad:	yes

Author(s)

C.Terfve Maintainer: C.Terfve <terfve@ebi.ac.uk>

References

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

Examples

```
library(CellNOptR)

tmpdir<-tempdir()
setwd(tmpdir)

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#From here there are 2 versions: 1. If you want to set the parameters yourself
pList<-list(
Data=CNOlistToy,
Model=ToyModel,
sizeFac = 1e-04,
NAFac = 1,
PopSize = 10,
Pmutation = 0.5,
MaxTime = 60,
maxGens = 5,
StallGenMax = 5,
SelPress = 1.2,
elitism = 5,
RelTol = 0.1,
verbose=TRUE)

CNORwrap(
paramsList=pList,
Name="Toy",
NamesData=list(CNOlist="ToyData",Model="ToyModel"),
Data=NA,
Model=NA)

#2. If you want to keep the default parameters
## Not run:
CNORwrap(
paramsList=NA,
Name="Toy",
NamesData=list(CNOlist="ToyData",Model="ToyModel"),
Data=CNOlistToy,
Model=ToyModel)

## End(Not run)
```

LiverDREAM *Model used for the DREAM3 challenge*

Description

This data object contains the model used in the package vignette, already loaded and formatted as a Model object. This is to be used with the data in "CNOListDREAM"

Usage

```
DreamModel
```

Format

DreamModel is a list with fields "reacID" (character vector), "namesSpecies" (character vector), "interMat" (numerical matrix), "notMat"(numerical matrix).

Source

This data and model is extracted from the Matlab version of CellNOpt1.0 (http://www.ebi.ac.uk/saezrodriguez/software.html#CellNetOptimizer).

References

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

Prill RJ, Marbach D, Saez-Rodriguez J, Sorger PK, Alexopoulos LG, Xue X, Clarke ND, Altan-Bonnet G, and Stolovitzky G. Towards a rigorous assessment of systems biology models: the DREAM3 challenges. PLoS One, 5(2):e9202, 2010.

ToyModel *Toy mode*

Description

This data object contains the Toy model from the package vignette, already loaded and formatted as a Model object.

Usage

```
ToyModel
```

Format

ToyModel is a list with fields "reacID" (character vector), "namesSpecies" (character vector), "interMat" (numerical matrix), "notMat"(numerical matrix).

Source

This data and model is extracted from the Matlab version of CellNOpt1.0 (http://www.ebi.ac.uk/saezrodriguez/software.html#CellNetOptimizer).

References

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

---

checkSignals *Check the CNOlist and model matching*

Description

This function checks that all the signals in the CNOlist match to species in the model. It also checks that the CNOlist and Model lists have the right format and contain the right fields.

Usage

```
checkSignals(CNOlist, Model)
```

Arguments

CNOlist	A CNOlist structure, as created by makeCNOlist
Model	A model structure, as created by readSif

Details

If the formats are wrong, this function produces an error with an explanation message. If the signals don't match the species, this function produces a warning that explains which signals don't match any species, and advises to remove them (THIS SHOULD BE DONE IMPERATIVELY) which is not done at this point but could be done in the form of this function returning a new CNOlist which would be the original if all ok or a CNOlist with non matching signals removed when necessary. I don't see this as a priority at this stage so if it happens, the user should remove those signals manually from the CNOlist (in both fields valueSignals and namesSignals).

Value

If all ok this function doesn't return anything.

Author(s)

C. Terfve

See Also

makeCNOlist, readSif

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
checkSignals(CNOlistToy,ToyModel)
```

---

compressModel *Compress a model*

Description

This function compresses a model by compressing species that are not signals/inhibited/stimulated and that are not dead ends/in complex logic (i.e. only species with either one input or one output are compressed)/in self loops.

Usage

```
compressModel(Model, indexes)
```

Arguments

Model	a model structure as produced by readSif
indexes	list of indexes of sthe species stimulated/inhibited/measured in the model, as created by indexFinder

Details

Be aware that in the multiple inputs/one output case, if one of the outputs is an '&' gate this function handles it fine as long as it is an '&' with 2 inputs and no more.

Value

a compressed model list, with an additional field called 'speciesCompressed' that contains the names of the species that have been compressed

Author(s)

C.Terfve

See Also

indexFinder, readSif

Examples

```
#load data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyComp<-compressModel(ToyModel,indicesToy)
```

cutAndPlotResultsT1

Plot the results of an optimisation at t1

Description

This function takes a model and an optimised bitstring, it cuts the model according to the bitstring and plots the results of the simulation along with the experimental data.

Usage

```
cutAndPlotResultsT1(Model, bString, SimList, CNOlist, indexList, plotPDF = FALSE
```

Arguments

Model	a model (the full one that was used for optimisation)
bString	a bitstring for T1 as output by gabinaryT1 (i.e. a vector of 1s and 0s)
SimList	a simlist corresponding to the model, as output by prep4Sim
CNOlist	a CNOlist, corresponding to the optimisation one
indexList	an indexList, produced by indexFinder ran on the model and the CNOlist above
plotPDF	TRUE or FALSE; tells whether you want a pdf to be produced or not

Value

This function doesn't return anything, it only plots the graph in your graphic window, and in a pdf file if asked

Author(s)

C.Terfve

See Also

gabinaryT1, prep4Sim

Examples

```
#load data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#pre-process model

checkSignals(CNOlistToy,ToyModel)
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=FALSE)
ToyNCNOcut<-cutNONC(ToyModel,ToyNCNOindices)
indicesToyNCNOcut<-indexFinder(CNOlistToy,ToyNCNOcut)
ToyNCNOcutComp<-compressModel(ToyNCNOcut,indicesToyNCNOcut)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
```

```
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)

#optimise

ToyFields4Sim<-prep4Sim(ToyNCNOcutCompExp)
initBstring<-rep(1,length(ToyNCNOcutCompExp$reacID))
ToyT1opt<-gaBinaryT1(
CNOlist=CNOlistToy,
Model=ToyNCNOcutCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyNCNOcutComp,
initBstring=initBstring,
maxGens=2,
PopSize = 5,
verbose=FALSE)
cutAndPlotResultsT1(
Model=ToyNCNOcutCompExp,
bString=ToyT1opt$bString,
SimList=ToyFields4Sim,
CNOlist=CNOlistToy,
indexList=indicesToyNCNOcutComp,
plotPDF=FALSE)
```

`cutNONC` *Cuts the non-observable/non-controllable species off the model*

Description

This function cuts the non observable and/or non controllable species off the model, and returns a cut model

Usage

```
cutNONC(Model, NONCindexes)
```

Arguments

`Model`	a model structure, as produced by readSif
`NONCindexes`	a vector of indices of species to remove in that model, as produced for example by findNONC

Details

This function takes in a model and a vector of indices of species to remove in that model and it removes those species and any reaction involving them (be aware, if you have x&y=z and x is to be removed, then the function produces y=z, because it works by removing entire rows of the Model matrices and then removes the columns that do not have either an input or an output). This function could actually be used to cut any species, not only NONC species.

Value

a model

Author(s)

C.Terfve

See Also

findNONC, readSif

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=FALSE)
ToyNCNOcut<-cutNONC(ToyModel,ToyNCNOindices)
```

---

expandGates *Expand the gates of a model*

Description

This function takes in a model and splits all AND gates into ORs. In addition, wherever there are more than one and up to 5 reactions coming into one node, it creates all possible ANDs combinations of them, but considering only ANDs with 2 and 3 inputs

Usage

```
expandGates(Model)
```

Arguments

Model	a model structure

Details

This function returns a model with additional fields that help keep track of the processing done on the network. I would advice not to overwrite on the initial model but rather to assign the result of this function to a variable with a different name.

Value

returns a Model, with additional fields:

SplitANDs	list that contains a named element for each AND reac that has been split, and each element contains a vector with the names of the of the reactions that result from the split if nothing was split, this element has the default value $initialReac [1] "split1" "split2"
newANDs	list that contains an element for each new '&' gate, named by the name of this new and reac, and containing a vector of the names of the reactions from which it was created (contains all the reacs in that pool, not the particular ones, this could be improved)

Note

This function could be simplified in the future.

Author(s)

C.Terfve

Examples

```
#load data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#pre-process the model

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=TRUE)
ToyNCNOcut<-cutNONC(ToyModel,ToyNCNOindices)
indicesToyNCNOcut<-indexFinder(CNOlistToy,ToyNCNOcut)
ToyNCNOcutComp<-compressModel(ToyNCNOcut,indicesToyNCNOcut)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)
```

---

findNONC *Find the indexes of the non-observable and non controllable species*

Description

This function finds the indexes of the non-observable and non controllable species and returns the indices, in the model, of the species to remove

Usage

```
findNONC(Model, indexes, verbose)
```

Arguments

Model	a model, as created by readSif
indexes	a list of indices of species measured, stimulated or inhibited, as created by indexFinder
verbose	do you want information about the ncno species printed on the screen? Default to FALSE but we would advise to put it to true the first time that the function is called

Details

This function uses the function floyd.warshall.all.pairs.sp from the package RBGL. Non observable nodes are those that do not have a path to any measured species in the model, whereas non controllable nodes are those that do not receive any information from a species that is perturbed in the data.

Value

a vector of indices of species to remove

Author(s)

C. Terfve

See Also

cutNONC, indexFinder, readSif

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
checkSignals(CNOlistToy,ToyModel)
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=FALSE)
```

---

gaBinaryT1 *Genetic algorithm used to optimise a model*

Description

This function is the genetic algorithm to be used to optimise a model by fitting to data containing one time point.

Usage

```
gaBinaryT1(CNOlist, Model, SimList, indexList, sizeFac = 1e-04, NAFac = 1, initB
```

Arguments

CNOlist	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at t1)
Model	a Model list
SimList	a list that contains additional fields for the simulator, as created by prep4Sim applied to the model above
indexList	a list of indexes of species stimulated/inhibited/signals, as produced by indexfinder applied on the model and CNOlist above
sizeFac	the scaling factor for the size term in the objective function, default to 0.0001
NAFac	the scaling factor for the NA term in the objective function, default to 1
initBstring	an initial bitsring to be tested, should be of the same size as the number of reactions in the model above
PopSize	the population size for the genetic algorithm, default set to 50
Pmutation	the mutation probability for the genetic algorithm, default set to 0.5
MaxTime	the maximum optimisation time in seconds, default set to 60
maxGens	the maximum number of generations in the genetic algorithm, default set to 500

StallGenMax	the maximum number of stall generations in the genetic algorithm, default to 100
SelPress	the selective pressure in the genetic algorithm, default set to 1.2
elitism	the number of best individuals that are propagated to the next generation in the genetic algorithm, default set to 5
RelTol	the relative tolerance for the best bitstring reported by the genetic algorithm, i.e.how different from the best solution can solutions be to be reported as well, default set to 0.1
verbose	logical (default to TRUE) do you want the statistics of each generation to be printed on the screen?

Details

The whole procedure is described in details in Saez-Rodriguez et al. (2009). The basic principle is that at each generation, the algorithm evaluates a population of models based on excluding or including some gates in the initial pre-processed model (this is encoded in a bitstring with contains 0/1 entries for each gate). The population is then evolved based on the results of the evaluation of these networks, where the evaluation is obtained by simulating the model (to steady state) under the various conditions present in the data, and then computing the squared deviation from the data, to which a penalty is added for size of the model and for species in the model that do not reach steady state.

Value

This function returns a list with elements:

bString	the best bitstring
Results	a matrix with columns "Generation","Best_score","Best_bitString","Stall_Generation","Avg_Score_
StringsTol	the bitstrings whose scores are within the tolerance
StringsTolScores	the scores of the above-mentioned strings

Author(s)

C. Terfve

References

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

See Also

GetFit, prep4Sim, indexFinder, simulatorT1

Examples

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

#pre-process model
```

```
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=FALSE)
ToyNCNOcut<-cutNONC(ToyModel,ToyNCNOindices)
indicesToyNCNOcut<-indexFinder(CNOlistToy,ToyNCNOcut)
ToyNCNOcutComp<-compressModel(ToyNCNOcut,indicesToyNCNOcut)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)

#optimise

ToyFields4Sim<-prep4Sim(ToyNCNOcutCompExp)
initBstring<-rep(1,length(ToyNCNOcutCompExp$reacID))
ToyT1opt<-gaBinaryT1(
CNOlist=CNOlistToy,
Model=ToyNCNOcutCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyNCNOcutComp,
initBstring=initBstring,
maxGens=2,
PopSize=5,
verbose=FALSE)
```

`getFit` *Compute the score of a model*

Description

This function computes the value of the objective function for a model and an associated data set, as a sum of a term that computes the fit of model to data, a term that penalises the NA values produced by the model, and a term that penalises increasing size of the model.

Usage

```
getFit(SimResults, CNOlist, Model, indexList, timePoint = c("t1", "t2"), sizeFac
```

Arguments

`SimResults`	matrix of simulated results (the full one as output by the simulator)
`CNOlist`	a CNOlist to compare the simulated results with
`Model`	a model that has already been cut to contain only the reactions in the optimal bitstring
`indexList`	list of indexes as produced by indexFinder
`timePoint`	"t1" or "t2" tells which time point we are looking at. If timePoint=t1 then we will compare the SimResults to the results stored in CNOlist$valueSignals[[2]]. If timePoint=t1 then we will compare the SimResults to the results stored in CNOlist$valueSignals[[2]]
`sizeFac`	weights the penalty for the size of the model, default=0.0001
`NAFac`	weights the penalty for the number of NAs

Details

BE AWARE: contrary to what is done in the Matlab version of CellNOpt, here the simulation results are computed beforehand and the Model that is input into this function is a model that has already been cut i.e. that only contains the reactions present in the optimised model (i.e.should be the same model as the one that you input into the simulator). Also, the SimResults matrix is the full one as output by the simulator, i.e. it contains results for all species in the model, not only the signals

Value

This function returns a single number, the value of the objective function.

Author(s)

C. Terfve

References

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

See Also

gabinaryT1, indexFinder, simulatorT1

Examples

```
#Here we will evaluate the fit of the full initial model,
#without pre-processing or any optimisation

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyFields4Sim<-prep4Sim(ToyModel)
SimResults<-simulatorT1(
CNOlist=CNOlistToy,
Model=ToyModel,
SimList=ToyFields4Sim,
indexList=indicesToy)
Score<-getFit(
SimResults=SimResults,
CNOlist=CNOlistToy,
Model=ToyModel,
indexList=indicesToy,
timePoint="t1")
```

`indexFinder` *Finds the indices, in the Model fields, of the species that are measured/inhibited/stimulated*

Description

This function finds the indices, in the Model fields, of the species that are measured/inhibited/stimulated. It looks for their position in Model$namesSpecies which has the same order as the rows of interMat and notMat, and therefore these indexes can be used there as well.

Usage

```
indexFinder(CNOlist, Model,verbose=FALSE)
```

Arguments

`CNOlist`	a CNOlist structure, as produced by makeCNOlist
`Model`	a model structure, as produced by readSif
`verbose`	do you want information about the cues and signals identities printed on the screen? Default if false but we would advise to set it to true when the function is called for the first time.

Value

a list with fields:

`signals`	vector of indices of the measured species
`stimulated`	vector of indices of the stimulated species
`inhibited`	vector of indices of the inhibited species

Author(s)

C. Terfve

See Also

makeCNOlist, ReadSif

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
```

`makeCNOlist` *Make a CNOlist structure*

Description

This function takes as input the output of readMIDAS and extracts the elements that are needed in a CNO project

Usage

```
makeCNOlist(dataset, subfield)
```

Arguments

`dataset`	output of readMIDAS
`subfield`	TRUE or FALSE, specifies if the column headers contain subfields or not i.e. if I should look for TR:sthg:sthg or just TR:sthg

Details

Be aware that most of the functions in this package, including this one, expect the data to contain measurements at time 0, but these should all be equal to zero according to the normalisation procedure that should be used. Therefore, if you have on time point, the files valueSignals contains two matrices, one for t0 and one for t1.

I there are replicate rows in the MIDAS file (i.e. identical cues and identical time), this function averages the values of the measurements for these replicates.

Value

a CNOlist with fields

`namesCues`	a vector of names of cues
`namesStimuli`	a vector of names of stimuli
`namesInhibitors`	a vector of names of inhibitors
`namesSignals`	a vector of names of signals
`timeSignals`	a vector of times
`valueCues`	a matrix of dimensions nConditions x nCues, with 0 or 1 if the cue is present or absent in the particular condition
`valueInhibitors`	a matrix of dimensions nConditions x nInhibitors, with 0 or 1 if the inhibitor is present or absent in the particular condition
`valueStimuli`	of dimensions nConditions x nStimuli, with 0 or 1 if the stimuli is present or absent in the particular condition
`valueSignals`	a list of the same length as timeSignals, each element containing a matrix of dimensions nConditions x nsignals, with the measurements.

Author(s)

C. Terfve

References

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

See Also

readMIDAS

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)
cpfile<-dir(system.file("ToyModel",package="CellNOptR"),full=TRUE)
file.copy(from=cpfile,to=getwd(),overwrite=TRUE)
dataToy<-readMIDAS(MIDASfile='ToyDataMMB.csv')
CNOlistToy<-makeCNOlist(dataset=dataToy,subfield=FALSE)
```

---

normaliseCNOlist *Normalisation for boolean modelling.*

Description

This function takes in a CNOlist and does the normalisation of the data between 0 and 1, according to two different procedures (see details)

Usage

```
normaliseCNOlist(CNOlist, EC50Data = 0.5, HillCoef = 2, EC50Noise = 0.1, Detecti
```

Arguments

CNOlist	a CNOlist
EC50Data	parameter for the scaling of the data between 0 and 1, default=0.5
HillCoef	Hill coefficient for the scaling of the data, defat to 2
EC50Noise	parameter for the computation of a penalty for data comparatively smaller than other time points or conditions
Detection	minimum detection level of the instrument, everything smaller will be treated as noise (NA), default to 0
Saturation	saturation level of the instrument, everything over this will be treated as NA, default to Inf.
ChangeTh	threshold for relative change considered significant, default to 0
Norm2TorCtrl	"time" or "ctrl": choice of a normalisation method: compute the relative change compared to the control at the same time, or to the same condition and measurement at time 0

Details

The normalisation procedure works as follows: a) every value that is out of the dynamic range of the equipment (as specified by the parameters `Detection` and `Saturation` are set to NA, b) values are transformed to fold changes relative to the same condition at t0 (if `Norm2TorCtrl="time"`) or the control condition (i.e. same inhibitors, no stimuli) at the same time (if `Norm2TorCtrl="ctrl"`), c) the fold changes are transformed with a Hill function (i.e. for each data point `x^HillCoef/((EC50Data^HillCo`
d) a penalty for "noisiness" is computed for each measurement as the value divided by the maximum value for that readout across all conditions and times (excluding values out of the dynamic range), e) the noise penalty is transformed by a saturation function (for each measurement `x/(EC50Noise+x)` where `x=x/max(x)`), f) the noise penalty and Hilled fold changes are multiplied, g) if the fold change is negative and bigger than `ChangeTh`, the resulting product is multiplied by -1, if the fold change is smaller than `ChangeTh` (either positive or negative), it is set to 0. The normalisation procedure applied here is explained in details in Saez-Rodriguez et al. (2009).

As the normalisation procedure works by computing a fold change relative to the same condition at time 0 or the control condition, if the aforementioned conditions have a value of zero (which is not expected with any common biochemical technique), then the fold change calculation will return a "NaN" value. If this is a problem for your particular case then we would suggest putting a dummy, very low value, instead of the zero, or setting that measurement to "NA" in the MIDAS file.

Value

a normalised CNOlist

Author(s)

C. Terfve

References

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

See Also

makeCNOlist

Examples

```
#Load a CNOlist

data(CNOlistToy,package="CellNOptR")

#Replace the values in the list by random values
#(for demonstration purposes, when actually using this function you would simply load a r

CNOlistToy$valueSignals$t0<-matrix(
data=runif(n=(dim(CNOlistToy$valueSignals$t0)[1]*dim(CNOlistToy$valueSignals$t0)[2]),min=
nrow=dim(CNOlistToy$valueSignals$t0)[1],
ncol=dim(CNOlistToy$valueSignals$t0)[2])
CNOlistToy$valueSignals[[2]]<-CNOlistToy$valueSignals[[1]]+matrix(
data=runif(n=(dim(CNOlistToy$valueSignals$t0)[1]*dim(CNOlistToy$valueSignals$t0)[2]),min=
nrow=dim(CNOlistToy$valueSignals$t0)[1],
ncol=dim(CNOlistToy$valueSignals$t0)[2])
```

```
CNOlistToyN<-normaliseCNOlist(
CNOlistToy,
EC50Data = 0.5,
HillCoef = 2,
EC50Noise = 0.1,
Detection = 0,
Saturation = Inf,
ChangeTh = 0,
Norm2TorCtrl = "time")
```

`plotCNOlist` *Plot the data in a CNOlist*

Description

This function plots the data in a CNOlist as a matrix of plots with a row for each condition and a column for each signal, and an extra plot for each row that specifies which cues are present.

Usage

```
plotCNOlist(CNOlist)
```

Arguments

`CNOlist`	a CNOlist

Details

This function can plot the normalised values or the un-normalised ones, it just needs a CNOlist.

Value

This function just produces a plot on your graphics window

Author(s)

C. Terfve

See Also

plotCNOlistPDF, plotCNOlistLarge, plotCNOlistLargePDF

Examples

```
data(CNOlistToy,package="CellNOptR")
plotCNOlist(CNOlistToy)
```

`plotCNOlistLarge` *Plot the data in a CNOlist, for lists with many conditions.*

Description

This function plots the data in a CNOlist as a matrix of plots with a row for each condition and a column for each signal, and an extra plot for each row that specifies which cues are present.

Usage

```
plotCNOlistLarge(CNOlist,nsplit=4)
```

Arguments

`CNOlist`	a CNOlist
`nsplit`	the number of splits in the condition dimension (one new plot window will be produced for each split, i.e. if you have 80 conditions and specify 4 splits you will get 4 plots with 20 conditions each).

Details

This function can plot normalised values or the un-normalised ones, it just needs a CNOlist. This function makes plots of CNOlists that are more readable when many conditions are present in the data. In addition to plotting the conditions divided into multiple plots, this function also plots the cues divided in two columns, one for inhibitors and one for stimuli.

Value

This function just produces plots on your graphics window.

Author(s)

C. Terfve

See Also

plotCNOlist, plotCNOlistPDF, plotCNOlistLargePDF

Examples

```
data(CNOlistDREAM,package="CellNOptR")
plotCNOlistLarge(CNOlistDREAM, nsplit=2)
```

plotCNOlistLargePDF *Plots a CNOlist into a pdf file, for lists with many conditions.*

Description

This function is a wrapper for plotCNOlistLarge, that plots the output directly in a pdf file.

Usage

```
plotCNOlistLargePDF(CNOlist, fileName, nsplit)
```

Arguments

CNOlist	a CNOlist
fileName	a name for your pdf file, eg. "plot.pdf"
nsplit	the number os splits along the condition dimension (see plotCNOlistLarge)

Details

This function makes plots of CNOlists that are more readable when many conditions are present in the data. In addition to plotting the conditions divided into multiple plots, this function also plots the cues divided in two columns, one for inhibitors and one for stimuli.

Value

This function doesn't return anything, it just produces a pdf file with your plots, in your current working directory.

Author(s)

C. Terfve

See Also

plotCNOlistLarge, plotCNOlist, plotCNOlistPDF

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)
data(CNOlistDREAM,package="CellNOptR")
plotCNOlistLargePDF(CNOlistDREAM, fileName="dreamData.pdf",nsplit=2)
```

plotCNOlistPDF *Plots a CNOlist into a pdf file.*

Description

This function is a wrapper for plotCNOlist, that plots the output directly in a pdf file.

Usage

```
plotCNOlistPDF(CNOlist, fileName)
```

Arguments

CNOlist	a CNOlist
fileName	a name for your pdf file, eg. "plot.pdf"

Value

This function doesn't return anything, it just produces a pdf file containing your plot, in your working directory.

Author(s)

C. Terfve

See Also

plotCNolist, plotCNOlistLarge, plotCNOlistLargePDF

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)
data(CNOlistToy,package="CellNOptR")
plotCNOlistPDF(CNOlist=CNOlistToy,fileName="ToyModelGraph.pdf")
```

plotFit *Plot the evolution of an optimisation*

Description

This function takes in the results of an optimisation by gaBinaryT1 and plots the evolution of best fit and average fit against generations.

Usage

```
plotFit(OptRes)
```

Arguments

OptRes	an object created by the optimisation engine (gabinaryT1)

Value

This function doesn't return anything, it just produces a plot in your graphics window.

Author(s)

C. Terfve

See Also

gabinaryT1

Examples

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

#process the model

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyComp<-compressModel(ToyModel,indicesToy)
indicesToyComp<-indexFinder(CNOlistToy,ToyComp)
ToyCompExp<-expandGates(ToyComp)

#optimise

ToyFields4Sim<-prep4Sim(ToyCompExp)
initBstring<-rep(1,length(ToyCompExp$reacID))
ToyT1opt<-gaBinaryT1(
CNOlist=CNOlistToy,
Model=ToyCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyComp,
initBstring=initBstring,
maxGens=2,
PopSize=5,
verbose=FALSE)

plotFit(OptRes=ToyT1opt)
```

---

plotOptimResults *Plot the data and simulated values*

Description

This function is the equivalent of CNOPlotFits, it plots the data and the simulated values, along with an image plot that tells which cues were present. The plots are coloured according to the fit between data and simulated data.

Usage

```
plotOptimResults(SimResults = SimResults, expResults = expResults, times = times
```

Arguments

SimResults	a list with a field for each time point, each containing a matrix of dimensions (number of conditions) * (number of signals), with the first field being t0. Typically produced by simulating a model and then extracting the columns that correspond to signals
expResults	same as above, but contains the experimental results, ie this is CNOlist$valueSignals
times	a vector of times, its length should be the same as the number of fields in SimResults and ExpResults
namesCues	a vector of names, typically CNOlist$namesCues
namesSignals	a vector of names, typically CNOlist$namesSignals
valueCues	a matrix of dimensions (number of conditions) * (number of cues), typically CNOlist$valueCues

Details

The colouring of the background is done as follows: the mean absolute difference between observed and simulated values are computed, and colours are chosen based on this value: red (above 0.9), indianred1 (between O.8 and 0.9), lightpink2 (between 0.7 and 0.8), lightpink (between 0.6 and 0.7), mistyrose (between 0.5 and 0.6), palegoldenrod (between 0.4 and 0.5), palegreen (between 0.3 and 0.4), darkolivegreen3 (between 0.2 and 0.3), chartreuse3 (between 0.1 and 0.2), forestgreen (between 0 and 0.1). This function is used inside cutAndPlotResultsT1.

Value

This function doesn't return anything, it just produces a plot in your graphics window.

Author(s)

C. Terfve

References

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

See Also

cutAndPlotResultsT1

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)

#We will plot the fit of the full initial model compared to the data, without any optimis
#This is normally not done on a stand alone basis, but if you have a model and would like

#load and prepare data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
```

```
ToyFields4Sim<-prep4Sim(ToyModel)

#simulate model

Sim<-simulatorT1(CNOlist=CNOlistToy,Model=ToyModel,SimList=ToyFields4Sim,indexList=indice

#format data and results

SimRes<-Sim[,indicesToy$signals]
SimResults<-list(t0=matrix(data=0,nrow=dim(SimRes)[1],ncol=dim(SimRes)[2]),t1=SimRes)
expResults<-list(t0=CNOlistToy$valueSignals[[1]],t1=CNOlistToy$valueSignals[[2]])

#plot

plotOptimResults(
SimResults=SimResults,
expResults=expResults,
times=CNOlistToy$timeSignals[1:2],
namesCues=CNOlistToy$namesCues,
namesSignals=CNOlistToy$namesSignals,
valueCues=CNOlistToy$valueCues)
```

`plotOptimResultsPDF`

Plot the data and simulated values in a pdf file

Description

This is a wrapper for `plotOptimResults`

Usage

```
plotOptimResultsPDF(SimResults = SimResults, expResults = expResults, times = ti
```

Arguments

`SimResults`	a list with a field for each time point, each containing a matrix of dimensions number of conditions * number of signals, with the first field being t0. Typically produced by simulating a model and then extracting the columns that correspond to signals
`expResults`	same as above, but contains the experimental results, ie this is `CNOlist$valueSignals`
`times`	a vector of times, its length should be the same as the number of fields in SimResults and ExpResults
`namesCues`	a vector of names, typically `CNOlist$namesCues`
`namesSignals`	a vector of names, typically `CNOlist$namesSignals`
`valueCues`	a matrix of dimensions (number of conditions) * (number of cues), typically `CNOlist$valueCues`
`fileName`	a name for your file, eg. "plot.pdf"

Details

The coloring of the background is done as follows: the mean absolute difference between observed and simulated values are computed, and colours are chosen based on this value: red (above 0.9), indianred1 (between O.8 and 0.9), lightpink2 (between 0.7 and 0.8), lightpink (between 0.6 and 0.7), mistyrose (between 0.5 and 0.6), palegoldenrod (between 0.4 and 0.5), palegreen (between 0.3 and 0.4), darkolivegreen3 (between 0.2 and 0.3), chartreuse3 (between 0.1 and 0.2), forestgreen (between 0 and 0.1). This function is used inside `cutAndPlotResultsT1`.

Value

This function doesn't return anything, it just produces a plot in a pdf document in your working directory.

Author(s)

C. Terfve

See Also

plotOptimResults, cutAndPlotResultsT1

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)

#We will plot the fit of the full initial model compared to the data, without any optimis
#This is normally not done on a stand alone basis, but if you have a model and would like
#its output compared to your data, then this is what you should do

#load and prepare data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyFields4Sim<-prep4Sim(ToyModel)

#simulate the model

Sim<-simulatorT1(CNOlist=CNOlistToy,Model=ToyModel,SimList=ToyFields4Sim,indexList=indice

#format the results and data as expected by plotOptimResults

SimRes<-Sim[,indicesToy$signals]
SimResults<-list(t0=matrix(data=0,nrow=dim(SimRes)[1],ncol=dim(SimRes)[2]),t1=SimRes)
expResults<-list(t0=CNOlistToy$valueSignals[[1]],t1=CNOlistToy$valueSignals[[2]])

#plot

plotOptimResultsPDF(
SimResults=SimResults,
expResults=expResults,
times=CNOlistToy$timeSignals[1:2],
namesCues=CNOlistToy$namesCues,
namesSignals=CNOlistToy$namesSignals,
valueCues=CNOlistToy$valueCues,
```

```
fileName="Toyfull.pdf")
```

`prep4Sim` *Prepare a model for simulation*

Description

Adds to the model some fields that are used by the simulation engine

Usage

```
prep4Sim(Model)
```

Arguments

`Model`	a model list, as output by readSif, normally pre-processed but that is not a requirement of this function

Details

This adds fields that are necessary for the simulation engine in a version that is extensible for constrained Fuzzy logic extension of the methods applied here (in development).

Value

this function returns a list with fields:

`finalCube`	stores, for each reac(row) the location of its inputs (col)
`ixNeg`	stores, for each reac(row) and each input (col) whether it is a negative input
`ignoreCube`	logical matrix of the same size as the 2 above, that tells whether the particular cell is filled or not
`maxIx`	row vector that stores, for each reac, the location of its output
`modelname`	stores the name of the model from which these fields were derived

Author(s)

C. Terfve

See Also

simulatorT1

Examples

```
data(ToyModel,package="CellNOptR")
ToyFields4Sim<-prep4Sim(ToyModel)
```

`readMIDAS` *Reads in a csv MIDAS file*

Description

This function takes in a single argument, the name of a csv MIDAS file containing the data, and returns a list that contains all the elements to build a CNOlist. The output of this function should be used as input for `makeCNOlist`.

Usage

```
readMIDAS(MIDASfile)
```

Arguments

`MIDASfile`	a csv MIDAS file

Details

This function does not return a CNOlist, but the output of this function can be used directly into makeCNOlist to create one. The MIDAS file format is described in Saez-Rodriguez et al. (2008).

If you have all of the readouts measured at the same series of time points, you can specify a unique DA: column which must have the format "DA:ALL".

Value

this function returns a list with fields:

`dataMatrix`	matrix containing the data in the MIDAS file
`TRcol`	indexes of the columns that contain the treatments (excluding cell line)
`DAcol`	indexes of the columns that contain the data time points
`DVcol`	indexes of the columns that contain the actual values (measurements)

Author(s)

C.Terfve

References

J. Saez-Rodriguez, A. Goldsipe, J. Muhlich, L. Alexopoulos, B. Millard, D. A. Lauffenburger, P. K. Sorger Flexible Informatics for Linking Experimental Data to Mathematical Models via DataRail. Bioinformatics, 24:6, 840-847 (2008).

See Also

makeCNOlist

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)
cpfile<-dir(system.file("ToyModel",package="CellNOptR"),full=TRUE)
file.copy(from=cpfile,to=getwd(),overwrite=TRUE)
dataToy<-readMIDAS(MIDASfile='ToyDataMMB.csv')
CNOlistToy<-makeCNOlist(dataset=dataToy,subfield=FALSE)
```

---

readSif *Read a sif file and create a model object*

Description

This function reads in a cityscape sif file and creates a model object that can be used in the CellNOptR procedure.

Usage

```
readSif(sifFile)
```

Arguments

sifFile	the name of a sif file

Details

This function takes in a single argument, sifFile, that points to a previous knowledge network in .sif format i.e. sourceNode-tab-sign-tab-targetNode. If there are ANDs they should be introduced as dummy nodes called and# (don't forget the number after "and" otherwise this won't be recognised). Please be aware that "and" nodes are not expected to be negated, i.e. there are not supposed to be !and1=xyz because that amounts to inverting the sign of all inputs of and1, which is more simply done at the inputs level.

Value

a model list with fields:

interMat	contains a matrix with column for each reaction and a row for each species, with a -1 where the species is the source node and a +1 where the species is a target node, and 0 otherwise
notMat	has the same format as interMat but just contains a 1 if the source node enters the reac with a negative effect, and 0 otherwise
namesSpecies	vector that contains the names of the species in the same order as the rows of the interMat and notMat matrices
reacID	vector that holds character strings specifying the reaction in full letters, in the same order as the columns of interMat and notMat

Author(s)

C. Terfve

References

Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Research 2003 Nov; 13(11):2498-504.

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)
cpfile<-dir(system.file("ToyModel",package="CellNOptR"),full=TRUE)
file.copy(from=cpfile,to=getwd(),overwrite=TRUE)
ToyModel<-readSif(sifFile="ToyPKNMMB.sif")
```

---

residualError *Compute the residual error for a dataset*

Description

This function takes in a CNOlist and computes the residual error, which is the minimum error between the scaled continuous data and a binary boolean approximation of this data.

Usage

```
residualError(CNOlist)
```

Arguments

`CNOlist`	a CNOlist

Details

Be aware that it is expected that `$valueSignals[[1]]` holds t0 (all signals=0) and `$valueSignals[[2]]` holds t1, `$valueSignals[[3]]` holds t2. If you give a CNOlist with more than 3 elements in valueSignals you will get a warning message but the function should still work based on the first 2 time points. If you give a CNOlist with only 2 elements in valueSignals (i.e. you don't have a time 2), the function will fill in the residual error t1 and leave t2 and t1andt2 = NA

Value

a vector with named entries t1, t2 and t1andt2 that hold the residual error for when only t1 is considered, only t2 is considered, or both are considered

Author(s)

C. Terfve

See Also

makeCNOlist, normaliseCNOlist, GetFit

Examples

```
data(CNOlistToy,package="CellNOptR")
resECNOlistToy<-residualError(CNOlistToy)
```

simulateT1 *Cut and simulation of a boolean model at t1*

Description

This function cuts a model according to a bitstring optimised at T1, and simulates the model accordingly

Usage

```
simulateT1(CNOlist, Model, bStringT1, SimList, indexList)
```

Arguments

CNOlist	a CNOlist object
Model	a full model
bStringT1	a bitstring to cut the model, as output by gabinaryT1 (i.e. a vector of 1s and 0s, of length equal to the number of reactions in the model)
SimList	a list of additional fields for simulation as created by prep4Sim, corresponding to the full model
indexList	a list of indexes as created by indexFinder

Details

This function is a wrapper for simulatorT1, that cuts the model before simulating it.

Value

a matrix of simulated values, including all species in the model, i.e. to be used as input of gabinaryT2 (not implemented here) but not to be used directly in plotOptimResults

Author(s)

C.Terfve

See Also

cutAndPlotOptimResultsT1, simulatorT1

Examples

```
#This will compute the output of a random model obtained by randomly selecting which gate

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=FALSE)
ToyFields4Sim<-prep4Sim(ToyModel)

SimRes<-simulateT1(
CNOlist=CNOlistToy,
Model=ToyModel,
```

```
bStringT1=round(runif(length(ToyModel$reacID))),
SimList=ToyFields4Sim,
indexList=indicesToy)
```

`simulatorT1` *Simulation of a boolean model*

Description

This is the simulator, inspired from BoolSimEngMKM in the Matlab CellNOpt, to be used on one time point simulations

Usage

```
simulatorT1(CNOlist, Model, SimList, indexList)
```

Arguments

`CNOlist`	a CNOlist
`Model`	a Model that only contains the reactions to be evaluated
`SimList`	a SimList as created by prep4Sim, that has also already been cut to contain only the reactions to be evaluated
`indexList`	an indexList as created by indexFinder

Details

Differences from the BoolSimEngMKM simulator include: the valueInhibitors has not been previously flipped; the function outputs the values across all conditions for all species in the model, instead of only for the signal species. This is because then the output of this function can be used as initial values for the version of the simulator that works on time point 2 (not implemented in this version).

If you would like to compute the output of a model that contains some of the gates in the model but not all, we suggest that you use the function `SimulateT1` and specify in the `bStringT1` argument which gates you want to be included. Indeed, SimulateT1 is a wrapper around simulatorT1 that takes care of cutting the model for you before simulating it.

Value

This function outputs a single matrix of format similar to valueSignals in the CNOlist but that contains an output for each species in the model. This matrix is the simulated equivalent of valueSignals at time 1, if you consider only the columns given by indexSignals.

Author(s)

C. Terfve

References

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

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

See Also

SimulateT1, cutAndPlotResultsT1

Examples

```
#This computes the output of the full model, which is normally not done on a stand alone

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyFields4Sim<-prep4Sim(ToyModel)

Sim<-simulatorT1(
CNOlist=CNOlistToy,
Model=ToyModel,
SimList=ToyFields4Sim,
indexList=indicesToy)
```

`writeDot` *Write a model, and attached features, to a dot file*

Description

This function writes a model to a Graphviz dot file with encoded features such as edge weight and nodes status (see details).

Usage

```
writeDot(dotNodes,dotMatrix,Model,fileName)
```

Arguments

`dotNodes`	internal variables created by writeNetwork or writeScaffold; dotNodes is a matrix with 2 columns: the first has the node names,and the second the attributes (signal, stimulated, inhibited, compressed, nano). A node can appear twice in this matrix if it belongs to more of one of the above categories; a node could also not appear here if it is is none of these categories
`dotMatrix`	internal variables created by writeNetwork or writeScaffold; dotMatrix is a matrix with 4 or 5 columns, and a row for each reaction:the first column holds the name of the input node, the second column holds the sign of the reaction (-1 if negative, 1 if positive), the third column holds the name of the output node, the fourth column holds the time stamp (0,1,2), an optional 5th column holds the weights of the edges

Model	A model to be plotted, if used inside writeNetwork then this should be the previous knowledge network (ModelOriginal), if inside writeScaffold then this should be the scaffold (ModelComprExpanded)
fileName	a name for the file

Details

This function is not to be used on its own, it should be used internally to writeNetwork or writeScaffold. For the colouring of the nodes, nodes that are both stimulated and inhibited or any other combination, only one colour per category is used, and the following order of priority for the colours is used: signals prime over inhibited nodes which primes over stimulated nodes which primes over non-controllable/non-observable nodes, which primes over compressed. Nodes that are neither of those have a black contour, stimulated nodes are green, inhibited are red, measure are blue, compressed and non-controllable/non-observable nodes are black and dashed. Edges are coloured according to time stamp in the optimal model (green=t1, blue=t1 and/or t2, grey=neither); on the scaffold, the strokes of the edges reflects the weights in the models within reltol (i.e. for each edge, the weight is the frequency with which it appeared among the models within the relative tolerance boundaries around the best solution).

Value

This function does not have any output, it just writes a dot file in your working directory.

Author(s)

C. Terfve

References

Emden R. Gansner , Stephen C. North. An Open Graph Visualization System and Its Applications to Software Engineering. Software - Practice and Experience (1999)

See Also

writeNetwork, writeScaffold

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)

#load data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#pre-process model

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=TRUE)
ToyNCNOcut<-cutNONC(ToyModel,ToyNCNOindices)
indicesToyNCNOcut<-indexFinder(CNOlistToy,ToyNCNOcut)
ToyNCNOcutComp<-compressModel(ToyNCNOcut,indicesToyNCNOcut)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)
```

```
#optimise

ToyFields4Sim<-prep4Sim(ToyNCNOcutCompExp)
initBstring<-rep(1,length(ToyNCNOcutCompExp$reacID))
ToyT1opt<-gaBinaryT1(
CNOlist=CNOlistToy,
Model=ToyNCNOcutCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyNCNOcutComp,
initBstring=initBstring,
maxGens=2,
PopSize=5,
verbose=TRUE)

#write network

writeNetwork(
ModelOriginal=ToyModel,
ModelComprExpanded=ToyNCNOcutCompExp,
optimResT1=ToyT1opt,
optimResT2=NA,
CNOlist=CNOlistToy)
```

`writeNetwork` *Write a previous knowledge network model to a sif file (with attribute files), as well as a dot file*

Description

This function writes the original previous knowledge network (the model that you loaded in the beginning of your analysis) in a sif file, with a nodes attribute file that specifies if each node was stimulated/inhibited/signal/compressed/non-controllable-non-observable and an edge attribute file that specifies if the edge was absent in the optimal model (0) present in the optimal model at t1 (1) or present in the optimal model at t2 (2).

This function also writes a Graphviz dot file that contains the same information (see `writeDot` for more information about the dot file conventions).

Usage

```
writeNetwork(ModelOriginal, ModelComprExpanded, optimResT1, optimResT2, CNOlist)
```

Arguments

`ModelOriginal`	The PKN model
`ModelComprExpanded`	The scaffold model (i.e. compressed and expanded)
`optimResT1`	The results of the optimisation process at t1
`optimResT2`	The results of the optimisation process at t2 (set this to NA, the t2 optimisation is not implemented in this version).
`CNOlist`	The CNOlist on which the optimisation is based

Details

The weights of the edges are computed as the mean across models within the relative tolerance limits, as output in the results from the optimisation `$StringsTol`. Strings that are in `$StringsTol` are the ones that are within the relative tolerance limits around the best solution in the population across all generations of the optimisation.

!If there is no time 2, then the argument optimResT2 should be = NA

This function maps back the edges weights from the optimised (expanded and compressed) model to the original model. 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).

Value

This function does not have any output, it just writes a sif file, an edge attribute file, and a node attribute file

Note

The mapback of this function is still an open question, even in the Matlab version. Future developments will include more robust versions of the mapping back algorithm, probably as a separate mapback function.

Author(s)

C. Terfve

See Also

writeScaffold, writeDot

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)

#load data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#pre-process model

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyNCNOindices<-findNONC(ToyModel,indicesToy,verbose=TRUE)
ToyNCNOcut<-cutNONC(ToyModel,ToyNCNOindices)
indicesToyNCNOcut<-indexFinder(CNOlistToy,ToyNCNOcut)
ToyNCNOcutComp<-compressModel(ToyNCNOcut,indicesToyNCNOcut)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)

#optimise

ToyFields4Sim<-prep4Sim(ToyNCNOcutCompExp)
initBstring<-rep(1,length(ToyNCNOcutCompExp$reacID))
ToyT1opt<-gaBinaryT1(
```

```
CNOlist=CNOlistToy,
Model=ToyNCNOcutCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyNCNOcutComp,
initBstring=initBstring,
verbose=TRUE,
maxGens=2,
PopSize=5)

#write network

writeNetwork(
ModelOriginal=ToyModel,
ModelComprExpanded=ToyNCNOcutCompExp,
optimResT1=ToyT1opt,
optimResT2=NA,
CNOlist=CNOlistToy)
```

`writeReport` *Write a report of a CellNOptR analysis*

Description

This function writes a short report of a CellNOptR analysis in an html page, that is linked to the various graphs produced

Usage

```
writeReport(ModelOriginal, ModelOpt, optimResT1, optimResT2, CNOlist, directory,
```

Arguments

`ModelOriginal`	the original previous knowledge network (i.e. model that you loaded) in a model list format
`ModelOpt`	the model that was actually used for optimisation (i.e. the scaffold network, after compression and expansion) in a model list format
`optimResT1`	the results of the optimisation at t1, as output by gabinaryT1
`optimResT2`	the results of the optimisation at t2, as output by gabinaryT2. Always set to NA here since the t2 optimisation is not implemented in this version
`CNOlist`	a CNOlist
`directory`	the name of a new directory that will be created, where your results will be moved
`namesFiles`	a list of the names of the files that should have been created. Depending on whether a t2 optimisation was performed or not, all or some of the following fields are expected: dataPlot,evolFit1,evolFit2,SimResults2,SimResults1,Scaffold,tscaffold,wscaffold
`namesData`	a list with fields `$CNOlist` and `$Model` that contain strings that are meaningful identifiers of your data and previous knowledge network (for your own record)
`resE`	a vector with named entries t1, t2 t1andt2, as produced by the function `ResidualError`, that contains the residual error associated with the discretisation of the data

Details

Future versions of this function might directly write and compile a tex file.

Value

This function produces a directory and moves all the files of namesFiles to it, then it creates an html report that contains infos about the optimisation process.

Author(s)

C.Terfve

See Also

writeNetwork, writeScaffold

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)

#load data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#pre-process model (partial)

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyNCNOcutComp<-compressModel(ToyModel,indicesToy)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)

#optimise

ToyFields4Sim<-prep4Sim(ToyNCNOcutCompExp)
initBstring<-rep(1,length(ToyNCNOcutCompExp$reacID))
ToyT1opt<-gaBinaryT1(
CNOlist=CNOlistToy,
Model=ToyNCNOcutCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyNCNOcutComp,
initBstring=initBstring,
maxGens=2,
PopSize=5,
verbose=TRUE)

#write report

namesFilesToy<-list(
dataPlot=NA,
evolFit1=NA,
evolFit2=NA,
SimResults1=NA,
SimResults2=NA,
Scaffold=NA,
```

```
ScaffoldDot=NA,
tscaffold=NA,
wscaffold=NA,
PKN=NA,
PKNdot=NA,
wPKN=NA,
nPKN=NA)
writeReport(
ModelOriginal=ToyModel,
ModelOpt=ToyNCNOcutCompExp,
optimResT1=ToyT1opt,
optimResT2=NA,
CNOlist=CNOlistToy,
directory="testToy",
namesFiles=namesFilesToy,
namesData=list(CNOlist="Toy",Model="ToyModel"),
resE=NA)
```

`writeScaffold` *Writes the scaffold network to a sif file (with attributes) and to a dot file*

Description

This function writes a cityscape sif file for the scaffold network, with an associated edge attribute file that holds whether the edge is present at t1,t2 or not present at all and another associated edge attribute file that holds the weights of the edges. This function also writes a dot file that contains the same information (see `writeDot` for more information about the dot file conventions).

Usage

```
writeScaffold(ModelComprExpanded, optimResT1, optimResT2, ModelOriginal, CNOlist
```

Arguments

`ModelComprExpanded`	The scaffold model (i.e. compressed and expanded)
`optimResT1`	The results of the optimisation process at t1
`optimResT2`	The results of the optimisation process at t2 (set this to NA, the t2 optimisation is not implemented in this version).
`ModelOriginal`	The PKN model
`CNOlist`	The CNOlist on which the optimisation is based

Details

By scaffold network we mean the network that is used as a basis for optimisation (i.e. a compressed and expanded network), therefore no map back of the weights is necessary here.

The weights of the edges are computed as the mean across models within the relative tolerance limits, as output in the results from the optimisation `$StringsTol`. Strings that are in `$StringsTol` are the ones that are within the relative tolerance limits around the best solution in the population across all generations of the optimisation.

!If there is no time 2, then the argument optimResT2 should be = NA.

Value

This function does not return anything, it writes a sif file and 2 edge attributes files, and a dot file, in your working directory.

Author(s)

C.Terfve

See Also

writeNetwork, writeDot

Examples

```
tmpdir<-tempdir()
setwd(tmpdir)

#load the data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

#pre-process the model (partial)

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyNCNOcutComp<-compressModel(ToyModel,indicesToy)
indicesToyNCNOcutComp<-indexFinder(CNOlistToy,ToyNCNOcutComp)
ToyNCNOcutCompExp<-expandGates(ToyNCNOcutComp)

#optimise

ToyFields4Sim<-prep4Sim(ToyNCNOcutCompExp)
initBstring<-rep(1,length(ToyNCNOcutCompExp$reacID))
ToyT1opt<-gaBinaryT1(
CNOlist=CNOlistToy,
Model=ToyNCNOcutCompExp,
SimList=ToyFields4Sim,
indexList=indicesToyNCNOcutComp,
initBstring=initBstring,
maxGens=3,
PopSize=5,
verbose=TRUE)

#write the network

writeScaffold(
ModelOriginal=ToyModel,
ModelComprExpanded=ToyNCNOcutCompExp,
optimResT1=ToyT1opt,
optimResT2=NA,
CNOlist=CNOlistToy)
```

Index