Package: CellNOptR (via r-universe)

November 1, 2026

Type Package

Title Training of boolean logic models of signalling networks using prior knowledge networks and perturbation data

Version 1.59.0

Date 2022-03-16

Depends R (>= 4.0.0), RBGL, graph, methods, RCurl, Rgraphviz, XML, ggplot2, rmarkdown

Imports igraph, stringi, stringr

Suggests data.table, dplyr, tidyr, readr, knitr, RUnit, BiocGenerics,

Enhances doParallel, foreach

VignetteBuilder knitr

biocViews CellBasedAssays, CellBiology, Proteomics, Pathways, Network, TimeCourse, ImmunoOncology

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

License GPL-3

LazyLoad yes

SystemRequirements Graphviz version >= 2.2

RoxygenNote 7.1.2

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-04-28 12:36:01 UTC

RemoteUrl https://github.com/bioc/CellNOptR

RemoteRef HEAD

RemoteSha bd7ff69ddb46550a43762891c38a6d68ace2a86a

Contents

CellNOptR-package . . . 4
build_sif_table_from_rule . . . 5
checkSignals . . . 6
CNOdata . . . 7
CNOlist-class . . . 7
CNOlist-methods . . . 8
CNOlistDREAM . . . 10
CNOlistToy . . . 11
CNOlistToy2 . . . 11
CNORbool . . . 12
CNORwrap . . . 13
compressModel . . . 15
computeScoreT1 . . . 16
computeScoreTN . . . 17
create_binaries . . . 18
createAndRunILP . . . 19
createILPBitstringAll . . . 20
crossInhibitedData . . . 21
crossvalidateBoolean . . . 22
cutAndPlot . . . 24
cutAndPlotResultsT1 . . . 25
cutAndPlotResultsTN . . . 26
cutCNOlist . . . 28
cutModel . . . 29
cutNONC . . . 30
cutSimList . . . 31
defaultParameters . . . 31
exhaustive . . . 32
expandGates . . . 33
findNONC . . . 35
gaBinaryT1 . . . 36
gaBinaryTN . . . 38
getFit . . . 40
graph2sif . . . 42
ilpBinaryT1 . . . 43
ilpBinaryT2 . . . 45
ilpBinaryTN . . . 47
indexFinder . . . 49
internals . . . 50
invokeCPLEX . . . 50
LiverDREAM . . . 51
makeCNOlist . . . 52
mapBack . . . 54
model2igraph . . . 54
model2sif . . . 55
normaliseCNOlist . . . 56

pknmodel . . . 58
plot-method . . . 59
plotCNOlist . . . 60
plotCNOlist2 . . . 61
plotCNOlistLarge . . . 62
plotCNOlistLargePDF . . . 63
plotCNOlistPDF . . . 64
plotFit . . . 64
plotModel . . . 65
plotOptimResults . . . 67
plotOptimResultsPan . . . 69
plotOptimResultsPDF . . . 71
prep4sim . . . 73
preprocessing . . . 74
randomizeCNOlist . . . 75
readBND . . . 76
readBNET . . . 77
readMIDAS . . . 77
readSBMLQual . . . 78
readSIF . . . 79
residualError . . . 81
sif2graph . . . 82
simulateTN . . . 83
simulatorT0 . . . 84
simulatorT1 . . . 85
simulatorTN . . . 87
toSBML . . . 88
ToyModel . . . 89
ToyModel2 . . . 90
write_bounds . . . 90
write_constraints . . . 91
writeDot . . . 92
writeFile . . . 94
writeMIDAS . . . 95
writeNetwork . . . 96
writeObjectiveFunction . . . 98
writeReport . . . 99
writeScaffold . . . 101
writeSIF . . . 103

Index **104**

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:	1.25.1
Date:	2018-01-10
License:	GPLv3
LazyLoad:	yes

Author(s)

T. Cokelaer, A. MacNamara, F. Eduati, S. Schrier, C. Terfve

Maintainer: A. Gabor<gabor.attila87@gmail.com>, until 2018-01-18: T. Cokelaer <cokelaer@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

```
# quick 1 time point optimisation of a Prior Knowledge Network to MIDAS data.
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")

pknmodel = ToyModel
cnolist = CNOlist(CNOlistToy)
model = preprocessing(cnolist, pknmodel)
results = gaBinaryT1(cnolist, model, verbose=FALSE)
plotFit(results)
cutAndPlot(cnolist, model, list(results$bString))

# Same as above and HTML report
CNORwrap(name="Toy",
```

```
    namesData=list(CNOlist="ToyData",model="ToyModel"),
    data=CNOlistToy, model=pknmodel)
```

`build_sif_table_from_rule` *Build a SIF table from a logic rule written in a string*

Description

Build a SIF table from a logic rule written in a string

Usage

```
build_sif_table_from_rule(rule_str, target, last_and_num = 0)
```

Arguments

`rule_str`	String containing the rule to be parsed
`target`	Name of the node affected by the rule
`last_and_num`	If the rule contains 'and' gates, their numeration will start after the number provided here (default is 0)

Value

data.frame with the network structure derived from the rule. The column 'sif_str' contains the string that can be written to a file and then read with 'readSIF()' in order to load a CellNOpt compatible network.

Examples

```
CellNOptR:::build_sif_table_from_rule("B & (C | D)", "A", last_and_num=2)

test_rule <- list()
test_rule[[1]] <- "AMP_ATP | (ATM & ATR) | HIF1 | !(EGFR | FGFR3)"
test_rule[[2]] <- "A & ((B | C) & !(D & E))"
test_rule[[3]] <- "A & B | C"
test_rule[[4]] <- "A & B & C"
test_rule[[5]] <- "A & (B | C)"
test_rule[[6]] <- "(A | B) & (C | D)"
test_rule[[7]] <- "!(C & D) | (E & F)"
test_rule[[8]] <- "(A | B) & (C | !D) & (E | F)"
parsed_rule <- list()
for (i in c(1:length(test_rule))){
  parsed_rule[[i]] <- CellNOptR:::build_sif_table_from_rule(test_rule[[i]], "T")
}
```

checkSignals *Check the CNOlist and model matching*

Description

This function checks that all the signals in a 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. It is called by the preprocessing function so there is no need to call it directly anymore if you use the preprocessing function.

In version 1.3.20, check of inhibitors and stimuli is also performed.

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. If the signals/inhibitors/stimuli do not match the species, this function produces a warning that explains which signals does no match any species.

Author(s)

C. Terfve, T. Cokelaer

See Also

makeCNOlist, readSIF, preprocessing

Examples

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

CNOdata *Get data from a CellNOpt data repository*

Description

This function fetch a file from the URL provided (default is http://www.ebi.ac.uk/~cokelaer/cellnopt/data/_downloads) and save it into a temporary file.

You will need Rcurl package to use this function.

Usage

```
CNOdata(filename, verbose=FALSE, url=NULL)
```

Arguments

filename	a valid filename that can be found in the url
verbose	FALSE by default, it prints the path of the temporary file where data has been copied.
url	You can overwrite the default URL (http://www.ebi.ac.uk/~cokelaer/cellnopt/data/_downloads) with this argument.

Value

the path of the temporary file where data has been copied.

Author(s)

T. Cokelaer

Examples

```
## Not run: readSIF(CNOdata("PKN-ToyMMB.sif"))
```

CNOlist-class *Class "CNOlist"*

Description

This function takes as input the filename of a MIDAS file (or the list returned by makeCNOlist) and returns an instance of CNOlist class. It provides an object oriented approach to manipulate CNOlist. This function calls readMIDAS and makeCNOlist.

Objects from the Class

Objects can be created by calls of the form `new("CNOlist", ...)`.

Slots

`cues`: Object of class `"matrix"`

`inhibitors`: Object of class `"matrix"`

`stimuli`: Object of class `"matrix"`

`signals`: Object of class `"list"`

`variances`: Object of class `"list"`

`timepoints`: Object of class `"vector"` timepoints contained in the signals matrix.
See CNOlist-methods for details

Methods

Available methods are plot, compatCNOlist, randomize, length. See CNOlist-methods for details.

Author(s)

T. Cokelaer

See Also

`randomizeCNOlist`, `plotCNOlist`, `plotCNOlist2`

Examples

```
showClass("CNOlist")

files<-dir(system.file("ToyModel",package="CellNOptR"),full=TRUE)
cnolist  = CNOlist(files[[1]])
# getters:
getCues(cnolist)
getInhibitors(cnolist)
getSignals(cnolist)
getVariances(cnolist)
getTimepoints(cnolist)
getStimuli(cnolist)
# In version 1.3.30 and above, use the plot method instead of former plotCNOlist function.
plot(cnolist)
new_cnolist = randomize(cnolist)
length(cnolist)
```

CNOlist-methods *List of CNOlist-class methods*

Description

CNOlist is a class with a set of methods described here below.

Usage

```
signature(x="CNOlist")
```

Getters

getCues Returns the cues (matrix) found in the `CNOlist`

getSignals Returns the signals (list of matrices) found in the `CNOlist`

getStimuli Returns the cues found in the `CNOlist`

getInhibitors Returns the inhibitors found in the `CNOlist`

getTimepoints Returns the timepoints found in the `CNOlist`

getVariances Returns the Variances (list of matrices) found in the `CNOlist`. Will be different from zero only if replicates were found in the MIDAS data. See `makeCNOlist`

Setters

setSignals Set the signals. No sanity check!

Other methods

compatCNOlist convert the instance CNOlist into the old-style returned by `makeCNOlist` that is a list. Used in the ODE package.

length returns length of CNOlist (number of time points)

randomize randomizes the signals matrice in a CNOlist. See `randomizeCNOlist` for details

show prints summary information

plot plot the CNOlist instance using the plotCNOlist function.

plot `signature(x="CNOlist", y="CNOlist")`: Please see the page of `plotCNOlist2` for more details.

readErrors `signature(object="CNOlist", filename="MIDAS-file")`: reads measurement error corresponding to the data from the MIDAS file and updates the CNOlist object

writeErrors `signature(object="CNOlist", filename="string",overwrite=F)`: writes measurement error corresponding to the data from the CNOlist to a MIDAS file

Author(s)

T.Cokelaer

See Also

`CNOlist-class`, `randomizeCNOlist` `makeCNOlist`

Examples

```
showClass("CNOlist")

data(CNOlistToyMMB, package="CellNOptR")
cnolist = CNOlistToyMMB

# In version 1.3.30 and above, use the plot method instead of former plotCNOlist function.
plot(cnolist)
# In version 1.5.14 and above, use getters instead of the @ operator
getCues(cnolist)

# others:
new_cnolist = randomize(cnolist)
```

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

```
data(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

1. 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.

2. 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

```
data(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.

`CNOlistToy2` *Toy data with 2 time points*

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, and modified to contain 2 time points. The second time point is such a way that all of the signals stay as in time 1, except for cJun and Jnk which go to zero.

Usage

```
data(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.

CNORbool *Simple Boolean analysis standalone*

Description

This function performs the optimisation of a PKN model to a CNOlist data set. It optimises each time point found in the data and returns the processed model as well as a list of optimised bitstring corresponding to each time points that has been optimised.

This function does not create any plots or reports unlike CNORwrap.

Usage

```
CNORbool(CNOlist, model, paramsList=defaultParameters(),
    compression=TRUE, expansion=TRUE, cutNONC=TRUE, verbose=FALSE,
    timeIndices = NULL)
```

Arguments

CNOlist	a CNOlist structure, as created by makeCNOlist or a MIDAS filename
model	a model structure, as created by readSIF or a SIF filename.
paramsList	Parameters of the genetic algorithm. If not provided, it is populated with the defaultParameters function.
compression	compress the model (default TRUE)
expansion	expand the gates (default TRUE)
cutNONC	cut the NONC nodes off the model by (default TRUE)
verbose	FALSE
timeIndices	by default, optimise T1 and T2 assuming there are the 2 first time points. However, with this argument you can change that behaviour to arbitrary time points.

Value

This function returns 2 components. The first one is the processed model used in the optimisation. The second is a list of optimised bitstrings found for each time points available in the MIDAS data set.

Author(s)

T.Cokelaer, S.Schrier

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
res = CNORbool(CNOlist=CNOlistToy, model=ToyModel)
```

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. Call the preprocessing function (Cut the nonc off the model, compress the model and expand the gates);
4. Compute the residual error;
5. Prepare for simulation;
6. Optimisation T1 and T2 (optional);
7. Plot simulated and experimental results;
8. Plot the evolution of fit;
9. Write the scaffold and PKN;
10. Write the report

Usage

```
CNORwrap(paramsList=NA, data=NA, model=NA, name, namesData=NA, time=1,
compression=TRUE, expansion=TRUE, cutNONC=TRUE)
```

Arguments

`paramsList` A list of parameters related to the Genetic Algorithm parameters:

1. sizeFac: default to 1e-04;
2. NAFac: default to 1;popSize: default to 50;
3. pMutation: default to 0.5;
4. maxTime: default to 60;
5. maxGens: default to 500;
6. stallGenMax: default to 100;
7. selPress: default to 1.2;
8. elitism: default to 5;
9. relTol: default to 0.1;
10. verbose: default to FALSE (default to true in the functions used by CNORwrap but CNORwrap sets them to false by default).

and the Data and Model structure.

If paramsList is not provided (NA), it is filled internally with the defaultParameters function.

If Data and Model are not provided in `paramList`, the function looks for Data and Model arguments.

If Data and Model are provided, the function overwrites the field `data` and `model` in `paramsList`.

`data`	a CNOlist structure, as created by makeCNOlist
`model`	a model structure, as created by readSIF.
`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). If not provided, the list is built automatically using the name arguments.
`time`	either 1 or 2: Do you want to perform a one time point steady state optimisation or a 2 time points pseudo steady state optimisation. By default this is set to 1.
`compression`	compress the model (default TRUE)
`expansion`	expand the gates (default TRUE)
`cutNONC`	cut the NONC nodes off the model by (default TRUE)

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

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

pList = defaultParameters(CNOlistToy, ToyModel)
pList$maxGens = 5
pList$popSize = 5

CNORwrap(paramsList=pList, name="Toy",
    namesData=list(CNOlist="ToyData", model="ToyModel"))

## Not run:
#version with default parameters

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

CNORwrap(name="Toy",
    namesData=list(CNOlist="ToyData",model="ToyModel"),
 data=CNOlistToy, model=ToyModel)

## End(Not run)
```

---

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.

You can also use preprocessing function instead that calls compressModel and other preprocessing functions.

Usage

```
compressModel(model, indexes)
```

Arguments

model	a model structure as produced by readSIF.
indexes	list of indexes of the 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

Note

No need to call this function directly since version 0.99.24. Use preprocessing instead.

Author(s)

C. Terfve

See Also

indexFinder, readSIF, preprocessing

Examples

```
#load data

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

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

---

computeScoreT1 *Compute the score of a model/data set using a bitString to cut the model.*

Description

The bitString made of 0 and 1 allows to select a submodel from the model provided. Then, the simulator function are called to compute the objective function. The sizeFac and NAFac are penalties added to the final score as described in `gaBinaryT1`. The indexList and simList arguments can be provided to speed up the code otherwise, they are recomputed from the CNOlist and model.

Usage

```
computeScoreT1(CNOlist, model, bString, simList=NULL, indexList=NULL,
    sizeFac=0.0001, NAFac=1, timeIndex=2)
```

Arguments

CNOlist	a CNOlist structure, as created by makeCNOlist.
model	a model structure, as created by codereadSIF, normally pre-processed but that is not a requirement of this function.
bString	a bitstring of the same size as the number of reactions in the model above
simList	If provided, simList should be created by prep4sim, that has also already been cut to contain only the reactions to be evaluated.
indexList	If provided, indexList should contain a list of indexes of the species stimulated/inhibited/measured in the model, as created by indexFinder.
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
timeIndex	the index of the time point to optimize. Must be greater or equal to 2 (1 corresponds to time=0). Must be less than the number of time points. Default is 2.

Value

score	See gaBinaryT1 for details

Author(s)

T. Cokelaer

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
model <- preprocessing(CNOlistToy,ToyModel)
score = computeScoreT1(CNOlist(CNOlistToy), model, bString=rep(1,16))
```

---

computeScoreTN *Compute the score at TN of a model/data set using a bitString to cut the model.*

Description

The bitString made of 0 and 1 allows to select a submodel from the model provided. Then, the simulator function are called to compute the objective function. The sizeFac and NAFac are penalties added to the final score as described in gaBinaryTN.

Usage

```
computeScoreTN(CNOlist, model, simList=NULL, indexList=NULL, simResPrev=NULL,
  bStringPrev=NULL, bStringNext=NULL, timeIndex=NULL, sizeFac=0.0001, NAFac=1, bStrings=NULL)
```

Arguments

CNOlist	a CNOlist structure, as created by makeCNOlist.
model	a model structure, as created by codereadSIF, normally pre-processed but that is not a requirement of this function.
simList	a simList as created by prep4sim, that has also already been cut to contain only the reactions to be evaluated. If not provided, it is recomputed automatically.
indexList	a list of indexes of the species stimulated/inhibited/measured in the model, as created by indexFinder. If not provided ,it is recomputed automatically.
simResPrev	Results of Previous simulation at time TN-1 step.
bStringPrev	the best bitString at time TN-1
bStringNext	the bitString to use to compute the score at time TN
timeIndex	Future feature will allows timeIndex to provide the exact list of indices to cut and plot. For now, it is based on the bitStrings provided.
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
bStrings	list of optimised bitstrings found at the previous time points

Value

score	See gaBinaryTN for details

Author(s)

T. Cokelaer, S.Schrier

Examples

```
data(CNOlistToy2,package="CellNOptR")
data(ToyModel2,package="CellNOptR")
model <- preprocessing(CNOlistToy2, ToyModel2)
bStringT1 = c(0,0,1,1,1,1,1,1,0,0,1,1,1,1,1,1)
simT1<-simulateTN(CNOlist=CNOlistToy2, model=model, bStrings=list(bStringT1))

score1 = computeScoreTN(CNOlistToy2, model, bStrings=list(bStringT1,c(1,0,1,0)))
```

---

create_binaries *Defining the set of binary variables for the ILP implementation of CellNOptR.*

Description

This function takes as input the model and data in a CNOlist object in order to define the set of binary variablese.

Usage

```
create_binaries(model,
                            midas,
                            numberOfExperiments,
                            y_vector)
```

Arguments

model	the model
midas	the midas table
numberOfExperiments	the number of experimental conditions
y_vector	the variables for each interaction in the PKN

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

---

createAndRunILP *Creating and running the ILP problem.*

Description

This function takes as an input the cno inputs (model + data) together with CPLEX parameters and then solves the ILP problem.

Usage

```
createAndRunILP(model = model,
                            midas = midas,
                            cnolist = cnolist,
                            accountForModelSize = accountForModelSize,
                            sizeFac = sizeFac,
                            source_path = source_path,
                            mipGap = mipGap,
                            relGap = relGap,
                            timelimit = timelimit,
                            cplexPath = cplexPath,
                            method = method,
```

```
                                numSolutions = numSolutions,
                                limitPop = limitPop,
                                poolIntensity = poolIntensity,
                                poolReplace = poolReplace)
```

Arguments

model	the model
midas	the midas table
cnolist	the cnolist object
accountForModelSize	the verbose parameter whether to account for model size
sizeFac	the size penalty factor
source_path	the source path
mipGap	the mipgap
relGap	the relGap
timelimit	the timelimit
cplexPath	the cplex solver path
method	the optimization method (quadratic/linear)
numSolutions	the number of desired solutions
limitPop	the limitPop
poolIntensity	the poolIntensity
poolReplace	the poolReplace

Author(s)

E Gjerga, H Koch

---

createILPBitstringAll *Reading the optimal solutions as bitstrings.*

Description

This function takes as the cplex optimization results and the variables assigned to each interaction in the ILP formulation in order to read the optimal bitstring.

Usage

```
createILPBitstringAll(cplexSolutionFileName,
                                  y_vector,
                                  binary_variables)
```

Arguments

cplexSolutionFileName	the file name where the cplex results are stored
y_vector	the variables for each interaction in the PKN
binary_variables	the binary variables of the ILP formulation

Author(s)

E Gjerga, H Koch

---

crossInhibitedData *If an inhibitor is also a measured species, replace the data with NA (when inhibited)*

Description

If an inhibitor is also a measured species, replace the data with NA (when inhibited)

Usage

```
crossInhibitedData(object)
```

Arguments

object	the CNOlist that contains the data

Value

the new cnolist

Author(s)

T. Cokelaer

Examples

```
data(ToyModel,package="CellNOptR")
data(CNOlistToy,package="CellNOptR")
cnolist = crossInhibitedData(CNOlist(CNOlistToy))
```

`crossvalidateBoolean` *k-fold crossvalidation for Boolean model.*

Description

Cross-validation analysis for the boolean case.

Usage

```
crossvalidateBoolean(CNOlist,model,nfolds=10,foldid=NULL,
                     type=c('datapoint','experiment','observable'),timeIndex = 2,parallel=FALSE,...)
```

Arguments

`CNOlist`	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at time 1).
`model`	a model structure, as created by `readSIF`, normally pre-processed but that is not a requirement of this function.
`nfolds`	number of folds - default is 10. Although nfolds can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets.
`foldid`	an optional vector of values between '1' and 'nfold' identifying what fold each observation is in. If supplied, 'nfold' can be missing.
`type`	define the way to do the crossvalidation. The default is type="datapoint"', which assigns the data randomly into folds. The option 'type="experiment"' uses whole experiments for crossvalidation (all data corresponding to a cue combination). The 'type=observable' uses the subset of nodes across all experiments for crossvalidation.
`timeIndex`	the index of the time point to optimize. Must be greater or equal to 2 (1 corresponds to time=0). Must be less than the number of time points. Default is 2.
`parallel`	verbose parameter, indicating wheter to parallelize the cross-validation procedure or not (default set to FALSE).
`...`	further arguments are passed to gaBinaryT1

Details

Does a k-fold cross-validation for Boolean CellNOpt models. In k-iterations a fraction of the data is eliminated from the CNOlist. The model is trained on the remaining data and then the model predicts the held-out data. Then the prediction accuracy is reported for each iteration.

Value

This function returns a list with elements:

`cvScores`	cross-validation scores

`fitScores`	fitting scores
`bStrings`	the optimal bit-string list for each run
`crossvalidate.call`	echo of the function which was called
`foldid`	the fold id's

Author(s)

A. Gabor, E. Gjerga

Examples

```
data("ToyModel", package="CellNOptR")
data("CNOlistToy", package="CellNOptR")
pknmodel = ToyModel
cnodata = CNOlist(CNOlistToy)

# original and preprocessed network
plotModel(pknmodel,cnodata)
model = preprocessing(data = cnodata,
     model = pknmodel,
     compression = TRUE,
     expansion = TRUE)
plotModel(model,cnodata)

# original CNOlist contains many timepoints, we use only a subset
plot(cnodata)
selectedTime = c(0,10)
cnodata_prep = cutCNOlist(cnodata,
                          model = model,
                       cutTimeIndices = which(!getTimepoints(cnodata) %in% selectedTime))

plot(cnodata_prep)

# optimise and show results
opt = gaBinaryT1(CNOlist = cnodata_prep,model = model,verbose = FALSE)

# 10-fold crossvalidation using T1 data
# We use only T1 data for crossvalidation, because data in the T0 matrix is not independent.
# All rows of data in T0 describes the basal condition.

# Crossvalidation produce some text in the command window:
## Not run:
library(doParallel)
registerDoParallel(cores=3)
R=crossvalidateBoolean(CNOlist = cnodata_prep,
      model = model,
      type = "datapoint",
      nfolds = 10,
      parallel = TRUE)
```

```
## End(Not run)
```

`cutAndPlot` *Interface to cutAndPlotResults functions.*

Description

This function takes a model and cnolist as well as a list of optimised bitstring at different time points. It calls the appropriate cutAndPlotResultsTX function.

Usage

```
cutAndPlot(CNOlist, model,  bStrings, plotPDF=FALSE, tag=NULL,
plotParams = list(maxrow = 10))
```

Arguments

`CNOlist`	a CNOlist, corresponding to the optimisation one
`model`	a model (the full one that was used for optimisation)
`bStrings`	a bitstring for T1 as output by gaBinaryT1 (i.e. a vector of 1s and 0s)
`plotPDF`	TRUE or FALSE; tells whether you want a pdf to be produced or not
`tag`	NULL or string; tells whether you want to prefix filenames with a tag (replaces the default behaviour).
`plotParams`	a list of option related to the PDF and plotting outputs. (1) maxrow is the maximum number of row used to plot the results. See plotOptimResultsPan for other fields.

Value

This function returns nothing. It plots a graph in your graphic window and sqve it in a file if asked

Author(s)

T. Cokelaer

See Also

cutAndPlotResultsT1

Examples

```
#load data

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

#pre-process model
model = preprocessing(CNOlistToy, ToyModel)

cutAndPlot(CNOlistToy, model,
    bStrings=list(rep(1,length(model$reacID))), plotPDF=FALSE)
```

`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=NULL, CNOlist, indexList=NULL, plotPDF =
FALSE, tag = NULL, tPt=CNOlist@timepoints[2],  plotParams = list(maxrow=10))
```

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`	deprecated argument kept for back compatibility. a simlist corresponding to the model, as output by prep4sim
`CNOlist`	a CNOlist, corresponding to the optimisation one
`indexList`	deprecated argument kept for back compatibility. 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
`tag`	NULL or string; tells whether you want to prefix filenames with a tag (replaces the default behaviour).
`tPt`	The number of time points in the data.
`plotParams`	a list of option related to the PDF and plotting outputs. (1) maxrow is the maximum number of row used to plot the results. See plotOptimResultsPan for other fields.

Value

This function returns plotted MSEs and list of filenames generated (if any)

Author(s)

C.Terfve, T. Cokelaer,A. MacNamara

See Also

gaBinaryT1

Examples

```
#load data

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

#pre-process model
model = preprocessing(CNOlistToy, ToyModel)

#optimise
ToyT1opt<-gaBinaryT1(
 CNOlist=CNOlistToy,
 model=model,
 maxGens=20,
 popSize = 10,
 verbose=FALSE)

#plotting
cutAndPlotResultsT1(
 model=model,
 CNOlist=CNOlistToy,
 bString=ToyT1opt$bString,
 plotPDF=FALSE)
```

---

cutAndPlotResultsTN *Plot the results of an optimisation at tN*

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. This function is designed to work on results of a 2 step optimisation.

Usage

```
cutAndPlotResultsTN(CNOlist, model, bStrings, plotPDF = FALSE, tag=NULL,
    plotParams = list(maxrow = 10))
```

Arguments

CNOlist	a CNOlist, corresponding to the optimisation one
model	a model (the full one that was used for optimisation)
bStrings	a list of bitstring at different time points
plotPDF	TRUE or FALSE, tells whether you want a pdf to be produced or not
tag	NULL or string; tells whether you want to prefix filenames with a tag (replaces the default behaviour).
maxrow	maximum number of row in the plot.
plotParams	a list of option related to the PDF and plotting outputs. (1) maxrow is the maximum number of row used to plot the results. See plotOptimResultsPan for other fields.

Value

This function returns plotted MSEs

Note

New in version 1.3.28

Author(s)

T. Cokelaer, A. MacNamara, Sarah Schrier, C. Terfve based on cutAndPlotResultsT1

See Also

gaBinaryT1, prep4sim, cutAndPlotResultsT1

Examples

```
#load data

data(CNOlistToy2,package="CellNOptR")
data(ToyModel2,package="CellNOptR")

#pre-process model
model = preprocessing(CNOlistToy2, ToyModel2)

#optimise t1
ToyT1<-gaBinaryT1(
 CNOlist=CNOlistToy2,
 model=model,
 maxGens=20,
 popSize = 10,
 verbose=FALSE)

#Optimise T2
ToyT2<-gaBinaryTN(
 CNOlist=CNOlistToy2,
```

```
 model=model,
 bStrings=list(ToyT1$bString),
 maxGens=20,
 popSize = 10,
 verbose=FALSE)

cutAndPlotResultsTN(
 CNOlist=CNOlistToy2,
 model=model,
 bStrings=list(ToyT1$bString, bStringT2=ToyT2$bString),
 plotPDF=FALSE)
```

`cutCNOlist` *Cut a CNOlist structure according to a model*

Description

The MIDAS file may contain species that are not contained in the model. If you want to remove cues and signals from your CNOlist that are not contained in the model, you can use this function by using the parameter model. It can also be used to remove some time points by using the parameter cutTimeIndices. Both parameters can be used at the same time and at least one of them must be provided.

Usage

```
cutCNOlist(cnolist, model, cutTimeIndices, verbose=FALSE)
```

Arguments

`cnolist`	the CNOlist structure
`model`	the model
`cutTimeIndices`	the time indices to remove
`verbose`	Set it to True to get the signals and cues not found in the model

Value

`cutCNOlist`	the new CNOlist object

Note

added in version 1.5.10

Author(s)

T. Cokelaer

See Also

CNOlist-class

Examples

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

cno_prep = cutCNOlist(cnolist = CNOlistToy,model = ToyModel,verbose = FALSE)
```

---

cutModel *Cut a model structure according to a bitstring*

Description

This function is for developers only.

Usage

```
cutModel(model, bString)
```

Arguments

model	the model object to cut.
bString	the bitString to be used to cut the model object.

Value

cutModel	the new model object

Note

added in version 1.3.16

Author(s)

T. Cokelaer

Examples

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

bString = rep(1,length(ToyModel$reacID))
# remove some reactions by setting them to 0.
bString[c(2,5,6)] <- 0

prepModel = cutModel(model = ToyModel, bString = bString )
```

cutNONC *Cuts the non-observable/non-controllable species from the model*

Description

This function cuts the non-observable and/or non-controllable species from 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

Note

No need to call this function directly since version 0.99.24. Use preprocessing instead.

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)
```

cutSimList *Cut a simList structure according to a bitstring*

Description

This function is for developers only.

Usage

```
cutSimList(simList, bString)
```

Arguments

simList	the simList object to cut.
bString	the bitString to be used to cut the simList object.

Value

cutSimList	the new simList object

Author(s)

T. Cokelaer

defaultParameters *Create a list of default parameters*

Description

This function provides a list of default parameters including the Genetic Algorithm parameters.

Usage

```
defaultParameters(data=NA, model=NA)
```

Arguments

data	a CNOlist structure, as created by makeCNOlist
model	a model structure, as created by readSIF, normally pre-processed but that is not a requirement of this function

Details

The list contains the Genetic Algorithm parameter, a verbose option and can be used to store the Data and Model.

Value

params	a list with the fields: data, model, verbose and all default parameters of gaBinaryT1

Author(s)

T. Cokelaer

Examples

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

---

exhaustive *Exhaustive search over the optimisation of a PKN model on MIDAS data.*

Description

This function performs an exhaustive search of the parameter space tring all the solutions. It is used internally by the genetic algorithm when a small model has to be optimised and the number of solutions to try is smaller than the number of iterations that the Genetic Algorithm will perform.

Usage

```
exhaustive(CNOlist, model, shuffle=FALSE, Nmax=NULL, verbose=TRUE, sizeFac =
0.0001, NAFac = 1, relTol=0.1, timeIndex=2)
```

Arguments

CNOlist	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at time 1)
model	a model structure, as created by readSIF, normally pre-processed but that is not a requirement of this function
shuffle	The list of bitstrings is set up arbitrarely. You may want to shuffle it.
Nmax	The total number of computation will be 2 to the power N, where N is the size of the model (ReacID field). The total number of computation can be large. You may want to set a maximumn number of computation using Nmax.
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
relTol	the relative tolerance for the best bitstring reported by the genetic algorithm, i.e., how different from the best solution, default set to 0.1 Not yet implemented.
verbose	logical (default to TRUE) do you want the statistics of each generation to be printed on the screen?
timeIndex	the index of the time point to optimize. Must be greater or equal to 2 (1 corresponds to time=0). Must be less than the number of time points. Default is 2.

Value

This function returns a list with elements:

bString	the best bitstring
bScore	the best score
all_scores	all scores that have been computed
results	a matrix with columns "Generation","Best_score","Best_bitString","Stall_Generation","Avg_Score_Gen'
stringsTol	the bitstrings whose scores are within the tolerance
stringsTolScores	the scores of the above-mentioned strings

Note that the field results, is not yet populated but maybe in the future.

Author(s)

T. Cokelaer

See Also

gaBinaryT1

Examples

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

#pre-process model

model = preprocessing(CNOlistToy, ToyModel)

#optimise

results <-exhaustive(
 CNOlist=CNOlistToy,
 model=model,
    shuffle=TRUE,
    Nmax=1000,
 verbose=FALSE)
```

---

expandGates *Expand the gates of a model*

Description

This function takes in a model and (1) splits all AND gates into ORs. In addition, (2) wherever there are more than one, it creates all possible ANDs combinations of them, but considering only ANDs with 2, 3 or 4 inputs according to the user argument (default is 2)

Usage

```
expandGates(model, ignoreList=NA, maxInputsPerGate=2)
```

Arguments

model	a model structure
ignoreList	an index vector of states to ignore incoming edges during step (1), i.e. at the time of splitting up AND gates
maxInputsPerGate	maximum number of input per gates (Default is 2; up to 4)

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

No need to call this function directly since version 0.99.24. Use preprocessing instead.

Author(s)

C.Terfve. T. Cokelaer, A.MacNamara, Martin-Franz-Xaver Pirkl

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, maxInputsPerGate=4)
```

`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=FALSE)
```

Arguments

`model`	a model structure, as created by readSIF
`indexes`	a list of indexes of the species stimulated/inhibited/measured, as created by indexFinder from a model.
`verbose`	verbose option (default to FALSE)

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

Note

No need to call this function directly since version 0.99.24. Use preprocessing instead.

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)
ToyNCNOindices <- findNONC(ToyModel, indicesToy)
```

---

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, initBstring=NULL, sizeFac = 1e-04,
    NAFac = 1, popSize = 50, pMutation = 0.5, maxTime = 60, maxGens = 500,
    stallGenMax = 100, selPress = 1.2, elitism = 5, relTol = 0.1, verbose=TRUE,
    priorBitString=NULL, timeIndex=2)
```

Arguments

CNOlist	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at time 1)
model	a model structure, as created by readSIF, normally pre-processed but that is not a requirement of this function
initBstring	an initial bitstring to be tested, should be of the same size as the number of reactions in the model above (model$reacID). Default is all ones.
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
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, default set to 0.1

verbose	logical (default to TRUE) do you want the statistics of each generation to be printed on the screen?
priorBitString	At each generation, the GA algorithm creates a population of bitstrings that will be used to perform the optimisation. If the user knows the values of some bits, they can be used to overwrite bit values proposed by the GA algorithm. If provided, the priorBitString must have the same length as the initial bitstring and be made of 0, 1 or NA (by default, this bitstring is set to NULL, which is equivalent to setting all bits to NA). Bits that are set to 0 or 1 are used to replace the bits created by the GA itself (see example).
timeIndex	the index of the time point to optimize. Must be greater or equal to 2 (1 corresponds to time=0). Must be less than the number of time points. Default is 2.

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
bScore	the best score
results	a matrix with columns "Generation","Best_score","Best_bitString","Stall_Generation","Avg_Score_Gen'
stringsTol	the bitstrings whose scores are within the tolerance
stringsTolScores	the scores of the above-mentioned strings

Author(s)

C. Terfve. T. Cokelaer

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

gaBinaryTN, simulatorT1

Examples

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

#pre-process model

model = preprocessing(CNOlistToy, ToyModel)

#optimise

initBstring<-rep(1,length(model$reacID))
ToyT1opt<-gaBinaryT1(
 CNOlist=CNOlistToy,
 model=model,
 initBstring=initBstring,
 maxGens=100, popSize=10, verbose=FALSE)

# During the optimisation, some bits can be overwritten by your prior knowledge
# First, you need to create a priorBitString made of NA where known bit values
# are replaced by 0 or 1
priorBitString = rep(NA, length(model$reacID))
priorBitString[1] = 0
priorBitString[2] = 1

# Second, you call the gaBinaryT1 function by providing the priorBitString
# argument:
ToyT1opt<-gaBinaryT1(CNOlist=CNOlistToy, model=model,
     initBstring=initBstring, maxGens=10, popSize=10, verbose=FALSE,
    priorBitString=priorBitString)
```

---

gaBinaryTN *Genetic algorithm for time point N*

Description

This is the genetic algorithm for time point N, that should follow optimisation based on time point 1.

Replaced gaBinaryT2.

Usage

```
gaBinaryTN(CNOlist, model,  bStrings, sizeFac = 1e-04, NAFac = 1,
popSize = 50, pMutation = 0.5, maxTime = 60, maxGens = 500,
stallGenMax = 100, selPress = 1.2, elitism = 5, relTol = 0.1, verbose=TRUE,
priorBitString=NULL, timeIndex = NULL)
```

Arguments

CNOlist	a CNOlist on which the score is based (based on valueSignals[[3]], i.e. data at t2)
model	a model structure, as created by readSIF, normally pre-processed but that is not a requirement of this function
bStrings	the optimal bitstring from optimisation at time 1 (i.e. a vector of 0s and 1s)
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
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 gen. al. 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
priorBitString	A bitString of same length at the initial bitstring made of 0, 1 or NA. By default, this bitstring is set to NULL (equivalent to setting all bits to NA). If provided, all bitstring in a population will be changed to be in agreement with the prior-BitString list.
timeIndex	todo

Details

This function takes in the same input as the T1 ga, but in addition it takes in the bitstring optimised for T1, and does not take an initial bitstring. Be aware that the bitString that this function returns is one that only includes the bits that it actually looks at, i.e. the bits that were 0 in the bStringT1

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_Gen'
stringsTol	the bitstrings whose scores are within the tolerance
stringsTolScores	the scores of the above-mentionned strings

Author(s)

C.Terfve, T. Cokelaer

See Also

getFit, simulatorT1, simulatorT2

Examples

```
#load data

data(CNOlistToy2,package="CellNOptR")
data(ToyModel2,package="CellNOptR")

#pre-process model

checkSignals(CNOlistToy2,ToyModel2)
model = preprocessing(CNOlistToy2, ToyModel2)

#optimise t1
ToyT1<-gaBinaryT1(
 CNOlist=CNOlistToy2,
 model=model,
 maxGens=10,
 popSize = 10,
 verbose=FALSE)

#Optimise T2

ToyT2<-gaBinaryTN(
 CNOlist=CNOlistToy2,
 model=model,
 bStrings=list(ToyT1$bString),
 maxGens=10,
 popSize = 10,
 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=NULL, timePoint=c("t1", "t2"),
sizeFac=1e-04, NAFac=1, nInTot, simResultsT0=NULL)
```

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. User should not use this parameter that is kept for back compatibility with previous version of the simulator written in R. The new simulator (C code) does not neccesitate the usage of the indexList parameter in this function anymore.
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
nInTot	the number of inputs in the model prior to cutting, used to normalised the size penalty
simResultsT0	Results of the time 0 simulator (internal usage of gaBinaryT1)

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, 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)
simResults = simResults[, indicesToy$signals]
Score<-getFit(
 simResults=simResults,
 CNOlist=CNOlistToy,
 model=ToyModel,
 timePoint="t1",
    nInTot=length(which(ToyModel$interMat == -1))
    )
```

`graph2sif` *Convert graph to SIF*

Description

This function converts a network form graph format to SIF format. The resulting table can also be saved in a SIF file.

Usage

```
graph2sif(graph, writeSif=FALSE, filename="Graph")
```

Arguments

`graph`	a graph, as generated using sif2graph
`writeSif`	if writeSif=FALSE (default) the SIF file is not saved. If writeSif=TRUE it is saved.
`filename`	the name of the SIF file saved if writeSif=TRUE. Default is Model.sif.

Details

The sign of link is supposed to be encoded in the graph as the weigth of the edge (-1 negative regulation, +1 positive regulation).

Value

`sifFile` a table with all the links in the model in the format sourceNode-tab-sign-tab-targetNode

Author(s)

F. Eduati

See Also

model2sif, sif2graph, readSIF,

Examples

```
data(ToyModel,package="CellNOptR")
sif_file = tempfile(fileext = ".sif")
writeSIF(model = ToyModel,filename = sif_file)
g = sif2graph(sif_file)

graph2sif(graph = g,writeSif = FALSE)
```

---

ilpBinaryT1 *ILP method used to optimise a model*

Description

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

Usage

```
ilpBinaryT1(cnolist,
                        model,
                        cplexPath,
                        sizeFac = 0.0001,
                        mipGap = 0,
                        relGap = 0,
                        timelimit = 3600,
                        method = "quadratic",
                        numSolutions = 100,
                        limitPop = 500,
                        poolIntensity = 0,
                        poolReplace = 2)
```

Arguments

cnolist	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at time 1)
model	a model structure, as created by readSIF, normally pre-processed but that is not a requirement of this function
sizeFac	the scaling factor for the size term in the objective function, default to 0.0001
mipGap	the absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When this difference falls below the value of this parameter, the linear integer optimization is stopped. Default set to 0
relGap	the relative tolerance on the objective value for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool. Default set to 0
timelimit	the maximum optimisation time in seconds, default set to 3600
cplexPath	the path where the cplex solver is stored (mandatory).
method	the method of writing the objective function (quadratic/linear). Default set to "quadratic"
numSolutions	the number of solutions to save
limitPop	the number of solutions to be generated. Default set to 500
poolIntensity	the Intensity of solution searching. Default set to 4
poolReplace	Pool replacement strategy

Value

This function returns a list with elements:

bitstringILPAll	the list of all optimal bitstrings identified
bScore	the best score for each set of bitstrings
time_cplex_only	the time it took for cplex to solve the problem
total_time	the total time for the pipeline to run (writing problem + solving problem + retrieving solutions)
stringsTolScores	the scores of the above-mentioned strings

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

Examples

```
# Toy Exampple
data("ToyModel", package="CellNOptR")
data("CNOlistToy", package="CellNOptR")
pknmodel = ToyModel
cnolist = CNOlist(CNOlistToy)
model = preprocessing(data = cnolist, model = pknmodel, compression = TRUE, expansion = TRUE)
plotModel(model = model, CNOlist = cnolist)

# Training to data - ILP
## Not run:
resILP = ilpBinaryT1(cnolist = cnolist, model = model)

## End(Not run)
```

`ilpBinaryT2` *ILP method used to optimise a model*

Description

This function is the ilp method to be used to optimise a model by fitting to data for time point 2, that should follow optimisation based on time point 1.

Usage

```
ilpBinaryT2(cnolist,
                        model,
                        sizeFac = 0.0001,
                        mipGap = 0,
                        relGap = 0,
                        timelimit = 3600,
                        cplexPath,
                        method = "quadratic",
                        numSolutions = 100,
                        limitPop = 500,
                        poolIntensity = 0,
                        poolReplace = 2)
```

Arguments

`cnolist`	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at time 1)
`model`	a model structure, as created by `readSIF`, normally pre-processed but that is not a requirement of this function
`sizeFac`	the scaling factor for the size term in the objective function, default to 0.0001

mipGap	the absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When this difference falls below the value of this parameter, the linear integer optimization is stopped. Default set to 0
relGap	the relative tolerance on the objective value for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool. Default set to 0
timelimit	the maximum optimisation time in seconds, default set to 3600
cplexPath	the path where the cplex solver is stored. Default set to "~/Documents/cplex"
method	the method of writing the objective function (quadratic/linear). Default set to "quadratic"
numSolutions	the number of solutions to save
limitPop	the number of solutions to be generated. Default set to 500
poolIntensity	the Intensity of solution searching. Default set to 4
poolReplace	pool replacement strategy, consult CPLEX manual for details.

Value

This function returns a list with elements:

bitstringILPAll	the list of all optimal bitstrings identified
bScore	the best score for each set of bitstrings
time_cplex_only	the time it took for cplex to solve the problem
total_time	the total time for the pipeline to run (writing problem + solving problem + retrieving solutions)
stringsTolScores	the scores of the above-mentioned strings

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

Examples

```
# Toy Exampple
data("ToyModel", package="CellNOptR")
data("CNOlistToy", package="CellNOptR")
pknmodel = ToyModel
```

```
cnolist = CNOlist(CNOlistToy)
model = preprocessing(data = cnolist, model = pknmodel, compression = TRUE, expansion = TRUE)
plotModel(model = model, CNOlist = cnolist)

# Training to data - ILP
## Not run:
resILP = ilpBinaryT2(cnolist = cnolist, model = model)

## End(Not run)
```

`ilpBinaryTN` *ILP method used to optimise a model*

Description

This function is the ilp method to be used to optimise a model by fitting to data for time point 2, that should follow optimisation based on time point 1.

Usage

```
ilpBinaryTN(cnolist,
                        model,
                        sizeFac = 0.0001,
                        mipGap = 0,
                        relGap = 0,
                        timelimit = 3600,
                        cplexPath,
                        method = "quadratic",
                        numSolutions = 100,
                        limitPop = 500,
                        poolIntensity = 0,
                        poolReplace = 2,
                        timeIndices = c(1, 2))
```

Arguments

`cnolist`	a CNOlist on which the score is based (based on valueSignals[[2]], i.e. data at time 1)
`model`	a model structure, as created by `readSIF`, normally pre-processed but that is not a requirement of this function
`sizeFac`	the scaling factor for the size term in the objective function, default to 0.0001
`mipGap`	the absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When this difference falls below the value of this parameter, the linear integer optimization is stopped. Default set to 0
`relGap`	the relative tolerance on the objective value for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool. Default set to 0

timelimit	the maximum optimisation time in seconds, default set to 3600
cplexPath	the path where the cplex solver is stored. Default set to "~/Documents/cplex"
method	the method of writing the objective function (quadratic/linear). Default set to "quadratic"
numSolutions	the number of solutions to save
limitPop	the number of solutions to be generated. Default set to 500
poolIntensity	the Intensity of solution searching. Default set to 4
timeIndices	the time indeces to optimize. Default set to timeIndices=c(1, 2)
poolReplace	pool replacement strategy, consult CPLEX manual for details.

Value

This function returns a list with elements:

bitstringILPAll	the list of all optimal bitstrings identified
bScore	the best score for each set of bitstrings
time_cplex_only	the time it took for cplex to solve the problem
total_time	the total time for the pipeline to run (writing problem + solving problem + retrieving solutions)
stringsTolScores	the scores of the above-mentioned strings

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

Examples

```
# Toy Exampple
data("ToyModel", package="CellNOptR")
data("CNOlistToy", package="CellNOptR")
pknmodel = ToyModel
cnolist = CNOlist(CNOlistToy)
model = preprocessing(data = cnolist, model = pknmodel, compression = TRUE, expansion = TRUE)
plotModel(model = model, CNOlist = cnolist)

# Training to data - ILP
## Not run:
```

```
resILP = ilpBinaryTN(cnolist = cnolist, model = model)

## End(Not run)
```

`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

Note

For internal usage since version 1.3.28

Author(s)

C. Terfve

See Also

makeCNOlist, readSIF

Examples

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

`internals` *List of CellNOptR internal functions.*

Description

This is a list of functions that are part of CelNOptR but are not exposed to the end-user. It may be of interest for developers. They may have a manual associated to it. If so, you can get the documentation in a R console as usual by tying preceding the name of the function with question tag.

Details

Function:	deprecated since	manual
cSimulator:	internal usage	yes
buildBitString:	internal usage	yes
simulateT1:	1.3.28 (use simulateTN)	yes

Author(s)

T.Cokelaer

`invokeCPLEX` *Solving the ILP problem with CPLEX.*

Description

This function takes as an input the file name where we write the ILP formulation together with CPLEX parameters and then solves the ILP problem.

Usage

```
invokeCPLEX(inputFileName,
                        outputFileName,
                        mipGap=mipGap,
                        relGap = relGap,
                        timelimit=timelimit,
                        cplexPath = cplexPath,
                        numSolutions = numSolutions,
                        limitPop = limitPop,
                        poolIntensity = poolIntensity,
                        poolReplace = poolReplace)
```

Arguments

inputFileName	the file name where the cplex ilp problem is stored
outputFileName	the file name where to store the cplex result
mipGap	the mipgap
relGap	the relGap
timelimit	the timelimit
cplexPath	the cplex solver path
numSolutions	the number of desired solutions
limitPop	the limitPop
poolIntensity	the poolIntensity
poolReplace	the poolReplace

Author(s)

E Gjerga, H Koch

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

```
data(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

1. 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.
2. 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.

`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. Instead, please use the `CNOlist` class to read a MIDAS file that will be converted to a CNOlist.

Usage

```
makeCNOlist(dataset, subfield, verbose=TRUE)
```

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.
`verbose`	logical (default to TRUE) print information on the screen.

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 one time point, the files valueSignals contains two matrices, one for t0 and one for t1.

If 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.

Columns with the following tags are ignored: NOINHIB, NO-INHIB, NO-LIG, NOCYTO.

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.
`valueVariances`	a list of the same length as timeSignals, each element containing a matrix of dimensions nConditions x nsignals, with the standard deviation of the replicates.

Author(s)

C. Terfve, T. Cokelaer

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, CNOlist-class

Examples

```
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)
```

mapBack — *Map an optimised model back onto the PKN model.*

Description

Map an optimised model back onto the PKN model.

Usage

```
mapBack(model, PKN, bString)
```

Arguments

model	the optimised model
PKN	the Prior Knowledge network
bString	the optimised bitString

Value

bStringPKN	the corresponding bitstring corresponding to the original PKN.

Author(s)

F.Eduati

See Also

graph2sif, sif2graph, readSIF,

model2igraph — *Convert a model object to a igraph object*

Description

This function receives as input a model object and converts it to a graph object made by igraph. igraph provides lots of utilities especially to write the file in different format such as GML.

Usage

```
model2igraph(model)
```

Arguments

model	the model as generated using readSIF

Value

g	a igraph object

Author(s)

T. Cokelaer

See Also

graph2sif, sif2graph

Examples

```
data(ToyModel,package="CellNOptR")
model2igraph(model=ToyModel)
```

---

model2sif *Convert a model object in sif format*

Description

This function receives as input a model object and converts it to the cytoscape sif format. It can be used to convert either the whole model (before or after pre-processing) or the optimized one (if the corresponding bitString in provided). The resulting table can also be saved in a sif file.

Usage

```
model2sif(model,optimRes=NA,writeSif=FALSE, filename="Model")
```

Arguments

model	the model as generated using readSIF
optimRes	the output of the optimisation (as obtained using gaBinaryT1), default set to NA the whole model in converted
writeSif	it writeSif=FALSE (default) the sif file is not saved. If writeSif=TRUE it is saved.
filename	the name of the sif file saved if writeSif=TRUE. Default is Model.sif.

Details

All links in the model are converted in sif format that is sourceNode-tab-sign-tab-targetNode. If there are ANDs, they are converted using dummy nodes called and# (e.g. A+B=C will be A-tab-1-tab-and1; B-tab-1-tab-and1; and1-tab-1-tab-C).

Value

sifFile	a table with all the links in the model in the format sourceNode-tab-sign-tab-targetNode

Author(s)

F.Eduati

See Also

graph2sif, sif2graph, readSIF,

Examples

```
data(ToyModel,package="CellNOptR")
model2sif(model=ToyModel,writeSif = 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., detection=0,
    saturation=Inf, changeTh=0, norm2TorCtrl=NULL, mode="time",
    options=list(rescale_negative = TRUE), verbose=FALSE)
```

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, default to 2
EC50Noise	parameter for the computation of a penalty for data comparatively smaller than other time points or conditions. No effect if set to zero (default).
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	deprecated since 1.5.0. Use the mode argument instead.

mode	"time" or "ctrl" or "raw" (experimental): choice of a normalisation method: "ctrl" computes the relative change compared to the control at the same time. "time" computes the relative change compared to the same condition and measurement at time 0. "raw" does not take into account time zero data; data are relative to time 0 so can be positive or negative values.
options	rescale column with negative values to be in 0-1 range (experimental)
verbose	prints some information if True (default is False)

Details

The normalisation procedure works as follows:

1. every value that is out of the dynamic range of the equipment (as specified by the parameters `detection` and `saturation` are set to NA,
2. values are transformed to fold changes relative to the same condition at t0 (if `mode="time"`) or the control condition (i.e. same inhibitors, no stimuli) at the same time (if `mode="ctrl"`),
3. the fold changes are transformed with a Hill function $\frac{x^{HillCoef}}{((EC50Data^{HillCoef})+(x^{HillCoef}))}$
4. 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)
5. the noise penalty is transformed by a saturation function (for each measurement $\frac{x}{(EC50Noise+x}$ where $x = \frac{x}{\max x}$),
6. the noise penalty and Hilled fold changes are multiplied,
7. 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 non-normalised CNOlist)

CNOlistToy$valueSignals$t0<-matrix(
 data=runif(n=(dim(CNOlistToy$valueSignals$t0)[1]*dim(CNOlistToy$valueSignals$t0)[2]),min=0,max=400),
 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=0,max=100),
 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,
 mode = "time")
```

---

pknmodel *pknmodel*

Description

Small in-silico case study used for demonstration. Use with CNOlist_ToyPB data.

Usage

```
data(PKN_ToyPB)
```

Format

An object of class `list` of length 4.

plot-method *plot a "CNOlist" object - methods*

Description

A plot method for CNOlist.

Usage

signature(x="CNOlist"): Please see the help page for the plot.CNOlist method in the CellNOptR package

arguments

x The CNOlist object to plot

Author(s)

T.Cokelaer

See Also

readMIDAS, makeCNOlist

Examples

```
showClass("CNOlist")

files<-dir(system.file("ToyModel",package="CellNOptR"),full=TRUE)
cnolist  = CNOlist(files[[1]])
# accessors:
getCues(cnolist)
getInhibitors(cnolist)
getSignals(cnolist)
getTimepoints(cnolist)
getStimuli(cnolist)
# In version 1.3.30 and above, use the plot method instead of former plotCNOlist function.
plot(cnolist)
```

`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)
```

plotCNOlist2 *Another version of plotCNOlist that allows to plot 2 cnolist in the same layout to compare them. This function uses ggplot2 library. It is recommended for small data sets (about 15 species).*

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, Cues are simply represented by a number.

Usage

```
plotCNOlist2(cnolist, simulated_cnolist=NULL, ymin=0,ymax=1)
```

Arguments

cnolist	a CNOlist
simulated_cnolist	another cnolist
ymin	Change the lower y-limit (default is 0)
ymax	Change the lower y-limit (default is 1)

Details

This function can plot either a single CNOlist, or 2 on top of each other.

Author(s)

T. Cokelaer

See Also

plotCNOlistPDF, plotCNOlistLarge, plotCNOlistLargePDF, plotCNOlist

Examples

```
# this data set is not an object so we need to convert it
data(CNOlistToy,package="CellNOptR")
cnolist = CNOlist(CNOlistToy)
plotCNOlist2(cnolist)
```

`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, newDevice=FALSE)
```

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).
`newDevice`	nsplit plots are created within the same device. In principle, most of the R Graphical USer Interface will allow the user to navigate between the different plots. However, if scripting only the last plot will be seen. If you want to create new device for each different plot, then set this option to TRUE.

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, width=14, height=7)
```

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)
width	set the width of the PDF document.
height	set the height of the PDF document.

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

```
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

```
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, filename = NULL)
```

Arguments

optRes	an object created by the optimisation engine (gabinaryT1)
filename	NULL or string: if provided, the plot is save in PDF format in the filename.

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
model = preprocessing(CNOlistToy,ToyModel)

#optimise

ToyT1opt<-gaBinaryT1(
 CNOlist=CNOlistToy,
 model=model,
 maxGens=10,
 popSize=10,
 verbose=FALSE)

plotFit(optRes=ToyT1opt)
```

---

plotModel *Plot a model*

Description

This function can be used to plot a prior model network before any pre-processing step. However, additional information can be provided such as a CNOlist (see makeCNOlist and readMIDAS) or information related to the pre-processing steps (compression, NONC nodes, expansion gates). It can also be used to plot optimised model given the optimisation bitstring.

Usage

```
plotModel(model, CNOlist=NULL, bString=NULL, indexIntegr=NULL, signals=NULL,
stimuli=NULL, inhibitors=NULL, NCNO=NULL, compressed=NULL, output="STDOUT",
filename=NULL, graphvizParams=list(), show=TRUE, remove_dot=TRUE, removeEmptyAnds=TRUE)
```

Arguments

model	a model as returned by readSIF. Alternatively, the filename can also be provided.
CNOlist	output of makeCNOlist
bString	a sequence made of numbers between 0 and 1 of same length as the one returned by the Genetic Algorithm (GA). This is a generalisation of the bitString returned by the GA function: several bit strings can be averaged and used.
indexIntegr	additional indices to highlight some edge (optional).
signals	a list of nodes belonging to the signals class
stimuli	a list of nodes belonging to the stimuli class
inhibitors	a list of nodes belonging to the inhibitors class
NCNO	a list of NCNO nodes.
compressed	a list of compressed nodes
filename	the filename (without extension) used to write the dot file
output	the type of output (PNG, PDF, SVG accepted)
graphvizParams	a list of optional arguments dedicated to Rgraphviz to tune the layout: • arrowsize default is 2 • sizea string for the size of the dot output; default is "15,15" • fontsize default is 22 • edgecolor default is "black" • nodeLabels overwrit node label with a list of proper length. • nodeWidth default is 2 • nodeHeight default is 1 • viewEmptyEdges default is TRUE • mode can be 'classic' or 'sbgn' (default). The difference appears in the and gate. • andWidth = 0.2 in 'classic' mode and 0.5 in 'sbgn' mode • andHeight = 0.2 in 'classic' mode and 0.5 in 'sbgn' mode
show	show the plot (default is True)
remove_dot	remove the dot file that has been created. Default is False
removeEmptyAnds	removes AND gates from plotted graph if the corresponding bit is 0 to give a compact view. Default is TRUE.

Details

This function plots the model and also saves it in a dot file that can be processed later on. However, you can also save the plot in PNG or PDF or SVG format (one at a time).

The CNOlist argument contains the signals/stimuli/inhibitors so if you provide a CNOlist there is no need to use these arguments. If you decide to use them they will overwrite the contents of the CNOlist argument.

optimRes is the output of gaBinary. One of its field is called bString and contains a list of 0 and 1 (the optimisation is perfomed with a binary procedure). This list of 0 and 1 is then used to plot or not the edges of the model. However, you can provide a bitString made of floats (e.g., average of several bitStrings). In such case, edges will appear in gray light or dark according to the bistring value (between 0 and 1).

Value

a graph representation of the model

`graph$g`	A graph representation of the model
`graph$attrs`	graph attributes
`graph$nodeAttrs`	nodes attributes
`graph$edgeAttrs`	edges attributes
`graph$clusters`	clusters of nodes

Note

This function depends on the Rgraphviz package.

Author(s)

T. Cokelaer

See Also

readMIDAS, readSIF, makeCNOlist, writeNetwork, writeDot, gaBinaryT1

Examples

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
res<-plotModel(ToyModel, CNOlist=CNOlistToy, compressed=c("TRAF6", "p38"),
    graphvizParams=list(mode="classic", fontsize=30))
```

---

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, expResults, times, namesCues, namesSignals,
valueCues, formalism="new")
```

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
formalism	New convention is to take the time=0 data set into account to compute the MSE. you can use the previous convetion by setting this argument to something different from the default value.

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

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

#load and prepare data

data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyFields4Sim<-prep4sim(ToyModel)

#simulate model

simRes<-simulatorT1(CNOlist=CNOlistToy,model=ToyModel, simList=ToyFields4Sim, indexList=indicesToy)

#format data and results

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)
```

---

plotOptimResultsPan *Plots the data and simulated values from any CellNOptR formalism*

Description

This function plots the data and simulated values according to each experiment in CNOlist. The data is shown as black triangles and the simulation by a blue dashed line. The combination of cues is given by a panel where black denotes the presence and white the absence of the cue. The goodness-of-fit between model and data is color-coded on a continuous scale from white to red.

Usage

```
plotOptimResultsPan(simResults, yInterpol=NULL, xCoords=NULL,
CNOlist=CNOlist, formalism=c("ss1","ss2","ssN","dt","ode"), pdf=FALSE,
pdfFileName="", tPt=NULL,
plotParams = list(margin = 0.1, width=15, height=12, cmap_scale=1, cex=1.6,
ymin=NULL, Fac=1, rotation=0))
```

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.
yInterpol	If using CNORdt, these are the interpolated experimental results from getFitTimeScale() that are needed to compare against the Boolean simulation.
xCoords	These are the x-coordinates obtained from the optimized scaling factor in CNORdt that allow for comparison between time course experimental data and a Boolean model.
CNOlist	A CNOlist.
formalism	An abbreviation of the CellNOptR formalism being used.
pdf	A Boolean argument denoting whether to print the figure produced by this function to file.
pdfFileName	If printing to file, the filename to be used.
tPt	The number of time points in the data.
plotParams	a list of option related to the PDF and plotting outputs. Currently, the following attributes are used: (1) margin of the boxes, (2) width and heigth used while creating the PDF, (3) cmap_scale a value that scales the colors towards small errors (<1) or large errors (>1); default is 1 (linear colormap) (4) cex is the fontsize used in the header (5) ymin sets the minimum y axis limit; by default it is the minimum value found over all data points and therefore can be negative.

Details

Depending on the logic formalism, this function is generally called from cutAndPlotResults*(). As shown in the example below however, it can plot the fit of any data and corresponding compatible model. The color denotes the goodness-of-fit, where white shows no difference between simulation and data and red is the maximum error from all conditions and readouts.

Value

This function does not return a value.

Author(s)

A. MacNamara

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

```
data(CNOlistToy,package="CellNOptR")
data(ToyModel,package="CellNOptR")
indicesToy <- indexFinder(CNOlistToy, ToyModel, verbose=TRUE)
ToyFields4Sim <- prep4sim(ToyModel)

# simulate model
simRes <- simulatorT1(CNOlist=CNOlistToy, model=ToyModel, simList=ToyFields4Sim, indexList=indicesToy)
simRes = simRes[, indicesToy$signals]

# format data and results

simResults <- list(t0=matrix(data=0, nrow=dim(simRes)[1], ncol=dim(simRes)[2]), t1=simRes)
# plot
plotOptimResultsPan(simResults,
  CNOlist=CNOlistToy,
  formalism="ss1",
  pdf=FALSE,
  tPt=10
)
```

---

plotOptimResultsPDF *Plot the data and simulated values in a pdf file*

Description

This is a wrapper for `plotOptimResults`

Usage

```
plotOptimResultsPDF(simResults, expResults, times, namesCues, namesSignals,
valueCues, filename, formalism="new")
```

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"
formalism	New convention is to take the time=0 data set into account to compute the MSE. you can use the previous convetion by setting this argument to something different from the default value.

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

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

simRes<-simulatorT1(CNOlist=CNOlistToy,model=ToyModel,simList=ToyFields4Sim,indexList=indicesToy)

#format the results and data as expected by plotOptimResults

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
`maxInput`	stores the max number of inputs observed in the model for a single reaction

Note

For internal usage since version 1.3.28

Author(s)

C. Terfve, T. Cokelaer

See Also

simulatorT1

Examples

```
data(ToyModel,package="CellNOptR")
ToyFields4Sim<-prep4sim(ToyModel)
```

---

preprocessing *Performs the pre-processing steps*

Description

This function performs any of the following preprocessing steps:

1. removes Non-Controllable and Non-Observables nodes
2. compress the model
3. and-gates expansion

Usage

```
preprocessing(data, model, cutNONC=TRUE, compression=TRUE,
    expansion=TRUE, ignoreList=NA, maxInputsPerGate=2, verbose=TRUE)
```

Arguments

data	the CNOlist that contains the data that you will use
model	the model object as returned by readSIF
cutNONC	Removes the NONC nodes using cutNONC and findNONC (Default is TRUE).
compression	Compress the model using compressModel (Default is TRUE).
expansion	Add and gates using expandGates (Default is TRUE).
ignoreList	an index vector of states to ignore incoming edges in expandGates.
maxInputsPerGate	used by the expandGates function to set maximum inputs per and gates.
verbose	verbose option (Default is TRUE).

Details

The function can apply any or none of the pre-processing steps. It returns the new model and the indices returned by indexFinder.

Value

the new model

Author(s)

T. Cokelaer

See Also

readSIF, readMIDAS, cutNONC, findNONC, compressModel, expandGates.

Examples

```
data(ToyModel,package="CellNOptR")
data(CNOlistToy,package="CellNOptR")
model = preprocessing(CNOlistToy, ToyModel, cutNONC=FALSE)
```

---

randomizeCNOlist *add noise to the data contained in a CNOlist.*

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

```
randomizeCNOlist(cnolist, sd=0.1, minValue=0, maxValue=1, mode="gaussian")
```

Arguments

cnolist	a CNOlist
sd	standard deviation to be used when adding gaussian noise. Not used if mode is uniform.
minValue	When adding Gaussian noise, the result may be below the minValue(default 0). If so, the value is set to minValue.
maxValue	When adding Gaussian noise, the result may be above the maxValue(default 1). If so, the value is set to maxValue.
mode	The mode can be either 'gaussian', 'shuffle' or uniform'. In gaussian mode, a gaussian noise is added to the data. The mean parameter is the data and the standard deviation is defined by the sd parameter. In uniform mode, the data is simply replaced by values taken from a uniform distribution between 0 and 1. In 'shuffle' mode all rows and columns are shuffled.

Value

a noisy CNOlist

Author(s)

T. Cokelaer

Examples

```
data(CNOlistToyMMB, package="CellNOptR")
cnolist = CNOlistToyMMB
cnolist2 = randomizeCNOlist(cnolist, mode="uniform")

# a method called randomize is available in the CNOlist class so you could type:
cnolist2 = randomize(cnolist, mode="uniform")
```

---

readBND *Read network from BND file*

Description

BND is a file format used by MaBoSS to store the boolean network definition. The reader works if the logic for the activation of the node is stated withing the parameter 'logic' of the node. An example file can be found in 'https://maboss.curie.fr/pub/example.bnd'.

Usage

```
readBND(filename)
```

Arguments

filename	BND file.

Value

CellNOpt network

Author(s)

Luis Tobalina

Examples

```
## Not run:
model = readBND("https://maboss.curie.fr/pub/example.bnd")

## End(Not run)
```

readBNET *Read network from BNET file*

Description

Read network from BNET file

Usage

```
readBNET(filename)
```

Arguments

filename	BNET file. The file is a tab delimited file with two columns, 'targets' and 'factors'. 'factors' contains the logic rule and 'targets' the node activated by the logic rule.

Value

CellNOpt network

Author(s)

Luis Tobalina

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, verbose=TRUE)
```

Arguments

MIDASfile	a CSV MIDAS file (see details)
verbose	logical (default to TRUE).

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

```
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)
```

---

readSBMLQual *Read a SBMLQual document and returns a SIF object (as returned by readSIG*

Description

This function reads a SBMLQual XML file where a model is stored. The XML is scanned and saved in SIF format in a temporary file. This file is read by readSIF. The returned object is therefore the output of readSIF.

Usage

```
readSBMLQual(filename)
```

Arguments

filename	The name of a SBMLQual file.

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)

T. Cokelaer

References

SBMLQual: qualitative models. See SBML.org for details.

Examples

```
## Not run:
sif = readSBMLQual("test.xml")

## End(Not run)
```

---

readSIF *Read a SIF file and create a model object*

Description

This function reads in a cytoscape 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 (Cytoscape format). See details for the accepted format.

Details

The input argument is the name of a SIF file that contains a prior knowledge network. The SIF format (See http://wiki.cytoscape.org/Cytoscape_User_Manual/Network_Formats/) can be of the form:

```
nodeA typeA node2
```

or

```
node2 typeB node3 node4
```

with space or tabulations in between types and nodes. Spaces and tabulations have different meaning in the original cytoscape format. However, we do not differentiate them. Therefore names of the nodes cannot have white space inside them.

The accepted values for the types are -1 (inhibitor) or 1 (normal relation).

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.

The SIF format can also include unconnected node that is a row with a single name:

```
nodeA
```

Although there would be no error, these type of rows are ignored.

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

```
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 and so on. The output is a list of residual errors at each time greater than 2. In addition, the total error is stored. For back compatibility, an additional field called t1andt2 is stored (NA is only t1 provided).

Value

a vector with named entries t1, t2, ...tn, t1andt2 and total. If only t1 is provided, t1andt2 is set to NA. The field t1andt2 may be removed in the future. Use the field called total instead.

Author(s)

C. Terfve, T. Cokelaer

See Also

makeCNOlist, normaliseCNOlist, getFit

Examples

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

`sif2graph` *Convert sif to graph*

Description

This function receives as input a network in form sif format and converts it to graph format.

Usage

```
sif2graph(sif)
```

Arguments

`sif`	the name of a sif file or the equivalent table

Details

This function takes a network in sif format (tabel or file), 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.

In the resulting graph, the sign of each link is encodes as the weigth of the edge (-1 negative regulation, +1 positive regulation).

Value

g	the corresponding graph

Author(s)

F.Eduati

See Also

graph2sif, model2sif, readSIF,

Examples

```
data(ToyModel,package="CellNOptR")
sif_file = tempfile(fileext = ".sif")
writeSIF(model = ToyModel,filename = sif_file)
g = sif2graph(sif_file)
```

simulateTN *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

```
simulateTN(CNOlist, model, bStrings)
```

Arguments

CNOlist	a CNOlist object
model	a full model
bStrings	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)

Details

This function is a wrapper around the family of functions called simulatorT1 , T2 and TN.

Value

a matrix of simulated values, including all species in the model, i.e. to be used as input of gaBinaryTN.

Author(s)

T.Cokelaer, S.Schrier based on simulatorT1 (C.Terfve)

See Also

cutAndPlotResultsT1, simulatorT1

Examples

```
# This will compute the output of a random model obtained by randomly selecting
# which gates of the initial models are included.

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

simRes<-simulateTN(
 CNOlist=CNOlistToy,
 model=ToyModel,
 bStrings=list(round(runif(length(ToyModel$reacID)))))
```

`simulatorT0` *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

```
simulatorT0(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)

T. Cokelaer

References

1. 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.
2. 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

simulateTN, cutAndPlotResultsT1

Examples

```
#This computes the output of the full model, which is normally not done on a stand alone basis, but if you have a mode

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

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyFields4Sim<-prep4sim(ToyModel)

Sim<-simulatorT0(
 CNOlist=CNOlistToy,
 model=ToyModel,
 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, mode=1)
```

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
mode	switch to use the cSimualor for time 0 or 1

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

1. 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.
2. 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

simulateTN, cutAndPlotResultsT1

Examples

```
#This computes the output of the full model, which is normally not done on a stand alone basis, but if you have a mode

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)
```

`simulatorTN` *Simulation of a boolean model at any time points dependent on a previous one.*

Description

This is a the simulator at TN using a C implementation. The computation relies on the time TN-1. T1 is a special case that is solved by using simulatorT2.

Usage

```
simulatorTN(simResultsPrev, CNOlist, model, simList, indexList, timeIndex=3)
```

Arguments

`simResultsPrev`	a matrix that is the output of simulatorTN (i.e. one row per condition and one column per species IN THE MODEL)
`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
`timeIndex`	3 by default to behave like deprecated function simulatorT2. This is the timeIndex at which the simulation is requested.

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)

T. Cokelaer, based on cSimulator (A. MacNamara)

References

1. 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.
2. 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

simulateTN, cutAndPlotResultsT1

Examples

```
#This computes the output of the full model, which is normally not done on a stand alone basis, but if you have a mode

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

indicesToy<-indexFinder(CNOlistToy,ToyModel,verbose=TRUE)
ToyFields4Sim<-prep4sim(ToyModel)

#Sim<-simulatorTN(
# CNOlist=CNOlistToy,
# model=ToyModel,
# simList=ToyFields4Sim,
# indexList=indicesToy)
```

---

toSBML *Export the network to SBML-qual format*

Description

Export a Boolean network <network> to an sbml-qual file <fileName>. This file can then be read in using other software that supports SBMLqual standards.

The function also takes a bit string as input. It cuts the model according to the values in bitstrings and write the new model object to SBMLqual.

Usage

```
toSBML(network, file, bitString = c(rep(1,length(network$reacID))),version=c("standard","cellnopt"))
```

Arguments

network	a valid CellNOptR network model created by e.g. readSIF()
file	a valid filename to save the SBMLqual model
bitString	optional vector of binary values (for example the resulted bitString from optimisation) to cut the unnecessary interactions from the network before exporting.
version	defines the format of SBMLqual file, read details.

Details

version = "standard" exports one transition block for each node of the network. This format is the SBMLqual standard, that can be imported then with other softwares

version = "cellnopt" the exported file follows a simplified syntax, where each edge of the network is transformed to a transition block in the SBMLqual file. Can be later imported to CellNOptR again.

Author(s)

Francesco Ceccarelli

Examples

```
data(ToyModel,package="CellNOptR")
toSBML(ToyModel,file = tempfile())
```

ToyModel *Toy model*

Description

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

Usage

```
data(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.

`ToyModel2` *Toy model*

Description

This data object contains the Toy model from the package vignette, already loaded and formatted as a Model object, and modified for the 2 time points version (a negative fedback between cJun and Jnk (!cJun=Jnk) is added).

Usage

```
data(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.

`write_bounds` *Writing the set of boundaries for each integer variable for the ILP implementation of CellNOptR.*

Description

This function takes as input the integer variables assigned to each of the elements in the network and data in a CNOlist object in order to produce the set boundaries for each integer variable.

Usage

```
write_bounds(model,
             midasTreatmentPart,
             y_vector,
             binary_variables)
```

Arguments

model	the model
midasTreatmentPart	the treatment part of CNOlist where data is stored
y_vector	the variables for each interaction in the PKN
binary_variables	the set of binary variables

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

---

write_constraints *Writing the set of constraints for the ILP implementation of CellNOptR.*

Description

This function takes as input the integer variables assigned to each of the elements in the network and data in a CNOlist object in order to produce the set of constraints needed to be optimized by the ILP solver.

Usage

```
write_constraints(model,
                              midasExperimentPart,
                              midasTreatmentPart,
                              reaction_sets,
                              y_vector,
                              midas,
                              binary_variables)
```

Arguments

model	the model
midasExperimentPart	the experimental part of CNOlist where data is stored
midasTreatmentPart	the treatment part of CNOlist where data is stored
reaction_sets	the set of reactions
y_vector	the variables for each interaction in the PKN
midas	the midas table
binary_variables	the set of binary variables

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

---

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

```
#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,
 initBstring=initBstring,
 maxGens=2,
 popSize=5,
 verbose=TRUE)

#write network

writeNetwork(
 modelOriginal=ToyModel,
 modelComprExpanded=ToyNCNOcutCompExp,
 optimResT1=ToyT1opt,
 optimResT2=NA,
 CNOlist=CNOlistToy)
```

`writeFile` *Writing the ILP problem.*

Description

This function takes as input the objective function. constraints, bounds and the solver path in order to generate a file containing the ILP problem.

Usage

```
writeFile(objectiveFunction,
                      constraints,
                      bounds,
                      binaries)
```

Arguments

`objectiveFunction`
: the objective function of the ILP problem

constraints	the set of constraints of the ILP problem
bounds	the set of bounds for each integer variable
binaries	the set of binary variables

Author(s)

E Gjerga, H Koch

---

writeMIDAS *Write a CNOlist structure into a MIDAS file*

Description

This function takes a CNOlist structure (output of makeCNOlist and readMIDAS) and save it a MIDAS format.

Usage

```
writeMIDAS(CNOlist, filename,  timeIndices=NULL, overwrite=FALSE)
```

Arguments

CNOlist	a CNOlist structure
filename	a filename. Not overwritten if it exists already
timeIndices	select subset of the times to be saved. Works with indices (not time values)
overwrite	overwrite the file if it exists already (Default is FALSE)

Author(s)

T. Cokelaer

See Also

makeCNOlist, readMIDAS, CNOlist-class

Examples

```
data(CNOlistToy)
writeMIDAS(CNOlistToy, 'test.csv')
readMIDAS('test.csv')
writeMIDAS(CNOlistToy, 'test.csv', timeIndices=c(1,2), overwrite=TRUE)
```

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,
tag = NULL,verbose=FALSE)
```

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 if you have performed a one time point optimisation).
`CNOlist`	The CNOlist on which the optimisation is based
`tag`	NULL or string; tells whether you want to prefix filenames with a tag (replaces the default behaviour).
`verbose`	If verbose=TRUE, the function prints a message every time an edge in the scaffold network couldn't be mapped back to the PKN

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

```
#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,
 initBstring=initBstring,
 verbose=TRUE,
 maxGens=2,
 popSize=5)

#write network

writeNetwork(
 modelOriginal=ToyModel,
```

```
modelComprExpanded=ToyNCNOcutCompExp,
optimResT1=ToyT1opt,
optimResT2=NA,
CNOlist=CNOlistToy)
```

`writeObjectiveFunction` *Writing the objective function for the ILP implementation of CellNOptR.*

Description

This function takes as input the integer variables assigned to each of the elements in the network and data in a CNOlist object in order to produce the objective function needed to be optimized by the ILP solver.

Usage

```
writeObjectiveFunction(model,
                                    midasExperimentPart,
                                    y_vector=y_vector,
                                    accountForModelSize = TRUE,
                                    sizeFac = .000001,
                                    meansOfMeasurements_at_t0,
                                    method = "quadratic" )
```

Arguments

`model`	the model
`midasExperimentPart`	the experimental part of CNOlist where data is stored
`y_vector`	the variables for each interaction in the PKN
`accountForModelSize`	the verbose variable if we wish to apply the size penalty
`sizeFac`	the size penalty factor
`meansOfMeasurements_at_t0`	the means of measurements at time-point 0
`method`	the method of defining the objective function ("quadratic/linear")

Author(s)

E Gjerga, H Koch

References

Alexander Mitsos, Ioannis N. Melas, Paraskeuas Siminelakis, Aikaterini D. Chairakaki, Julio Saez-Rodriguez, and Leonidas G. Alexopoulos. Identifying Drug Effects via Pathway Alterations using an Integer Linear Programming Optimization Formulation on Phosphoproteomic Data. PLoS Comput Biol. 2009 Dec; 5(12): e1000591.

`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,
   namesFiles = list(dataPlot = NA, evolFitT1=NA, evolFitT2=NA, simResultsT1=NA,
    simResultsT2=NA, scaffold=NA, scaffoldDot=NA, tscaffold=NA, wscaffold=NA,
    PKN=NA, PKNdot=NA, wPKN=NA,nPKN=NA), namesData = list(CNOlist=NA, model=NA),
    resE=NULL)
```

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 gabinaryTN. Set this to NA if you have performed a one time point optimisation.
`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, evolFitT1, evolFitT2, simResultsT1, simResultsT2, scaffoldDot, scaffold, tscaffold, wscaffold, PKN, PKNdot, wPKN, nPKN.
`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. Since version 1.3.29, there is no need to provide this argument. Kept for back compatibility.

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

```
#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,
 initBstring=initBstring,
 maxGens=2,
 popSize=5,
 verbose=TRUE)

#write report

namesFilesToy<-list(
 dataPlot=NA,
 evolFitT1=NA,
 evolFitT2=NA,
 simResultsT1=NA,
 simResultsT2=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 cytoscape 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, tag=NULL)
```

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 if you have performed a one time point optimisation).
`modelOriginal`	The PKN model
`CNOlist`	The CNOlist on which the optimisation is based
`tag`	NULL or string; tells whether you want to prefix filenames with a tag (replaces the default behaviour).

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

```
#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,
 initBstring=initBstring,
 maxGens=3,
 popSize=5,
 verbose=TRUE)

#write the network
```

```
writeScaffold(
 modelOriginal=ToyModel,
 modelComprExpanded=ToyNCNOcutCompExp,
 optimResT1=ToyT1opt,
 optimResT2=NA,
 CNOlist=CNOlistToy)
```

---

writeSIF *Convert a model into a SIF format and save the result in a file.*

Description

This function takes as input a model (as created by e.g., read from a SIF data set with readSIF function) and save it into a file.

Usage

```
writeSIF(model, filename,  overwrite = FALSE)
```

Arguments

model	the model
filename	the filename
overwrite	by default, do not overwrite a file.

Author(s)

T Cokelaer

Examples

```
cpfile<-dir(system.file("ToyModel",package="CellNOptR"),full=TRUE)
file.copy(from=cpfile,to=getwd(),overwrite=TRUE)
ToyModel<-readSIF(sifFile="ToyPKNMMB.sif")
writeSIF(ToyModel, "ToyPKNMMB_copy.sif")
```

Index

∗ **CNOlist**
  CNOlist-methods, 8
  plot-method, 59
∗ **classes**
  CNOlist-class, 7
∗ **datasets**
  CNOlistDREAM, 10
  CNOlistToy, 11
  CNOlistToy2, 11
  LiverDREAM, 51
  pknmodel, 58
  ToyModel, 89
  ToyModel2, 90
∗ **methods**
  CNOlist-methods, 8
  plot-method, 59
∗ **package**
  CellNOptR-package, 4

build_sif_table_from_rule, 5

CellNOptR *(*CellNOptR-package*)*, 4
CellNOptR-package, 4
checkSignals, 6
CNOdata, 7
cnodata *(*CNOdata*)*, 7
CNOlist *(*CNOlist-class*)*, 7
CNOlist-class, 7, *8*, *28*, *53*, *95*
CNOlist-methods, 8, *8*
CNOlistDREAM, 10
CNOlistToy, 11
CNOlistToy2, 11
CNORbool, 12
CNORwrap, *12*, 13
compatCNOlist *(*CNOlist-methods*)*, 8
compatCNOlist,CNOlist
        *(*CNOlist-methods*)*, 8
compatCNOlist,CNOlist-method
        *(*CNOlist-class*)*, 7
compressModel, 15, *15*, *74*, *75*
computeScoreT1, 16
computeScoreTN, 17
create_binaries, 18
createAndRunILP, 19
createILPBitstringAll, 20
crossInhibitedData, 21
crossvalidateBoolean, 22
cSimulator, *88*
cutAndPlot, 24
cutAndPlotResultsT1, *24*, 25, *27*, *83*, *85*, *86*, *88*
cutAndPlotResultsTN, 26
cutCNOlist, 28
cutModel, 29
cutNONC, 30, *35*, *74*, *75*
cutSimList, 31

defaultParameters, *12*, *14*, 31
DreamModel *(*LiverDREAM*)*, 51

exhaustive, 32
expandGates, 33, *74*, *75*

findNONC, *30*, 35, *74*, *75*

gaBinaryT1, *26*, *27*, *32*, *33*, 36, *41*, *55*, *65*, *67*
gaBinaryTN, *37*, 38
getCues *(*CNOlist-methods*)*, 8
getCues,CNOlist *(*CNOlist-methods*)*, 8
getCues,CNOlist-method *(*CNOlist-class*)*, 7
getFit, 40, *40*, *81*
getInhibitors *(*CNOlist-methods*)*, 8
getInhibitors,CNOlist
        *(*CNOlist-methods*)*, 8
getInhibitors,CNOlist-method
        *(*CNOlist-class*)*, 7
getSignals *(*CNOlist-methods*)*, 8
getSignals,CNOlist *(*CNOlist-methods*)*, 8
getSignals,CNOlist-method
        *(*CNOlist-class*)*, 7

getStimuli (CNOlist-methods), 8
getStimuli,CNOlist (CNOlist-methods), 8
getStimuli,CNOlist-method (CNOlist-class), 7
getTimepoints (CNOlist-methods), 8
getTimepoints,CNOlist (CNOlist-methods), 8
getTimepoints,CNOlist-method (CNOlist-class), 7
getVariances (CNOlist-methods), 8
getVariances,CNOlist (CNOlist-methods), 8
getVariances,CNOlist,CNOlist-method (CNOlist-methods), 8
getVariances,CNOlist-method (CNOlist-class), 7
graph2sif, 42, *54–56*, *82*

ilpBinaryT1, 43
ilpBinaryT2, 45
ilpBinaryTN, 47
indexFinder, *15, 16*, *35*, 49
internals, 50
invokeCPLEX, 50

length (CNOlist-methods), 8
length,CNOlist (CNOlist-methods), 8
length,CNOlist,ANY-method (CNOlist-methods), 8
length,CNOlist-method (CNOlist-class), 7
LiverDREAM, 51

makeCNOlist, *6, 7*, *9*, *12*, *14*, *49*, 52, *58, 59*, *67*, *77, 78*, *81*, *95*
mapBack, 54
model2igraph, 54
model2sif, *43*, 55, *82*

normaliseCNOlist, 56, *81*

pknmodel, 58
plot,CNOlist,ANY-method (plot-method), 59
plot,CNOlist,CNOlist,CNOlist-method (CNOlist-methods), 8
plot,CNOlist,CNOlist-method (CNOlist-class), 7
plot-method, 59
plot.CNOlist (plot-method), 59
plotCNOlist, *8, 9*, 60, *62–64*
plotCNOlist2, *8, 9*, 61
plotCNOlistLarge, 62, *63, 64*
plotCNOlistLargePDF, *62*, 63, *64*
plotCNOlistPDF, *62, 63*, 64
plotFit, 64
plotModel, 65
plotOptimResults, 67
plotOptimResultsPan, *24, 25*, *27*, 69
plotOptimResultsPDF, 71
prep4Sim (prep4sim), 73
prep4sim, *27*, 73
preprocessing, *6*, *15, 16*, *30*, *34, 35*, 74

randomize (CNOlist-methods), 8
randomize,CNOlist (CNOlist-methods), 8
randomize,CNOlist,CNOlist-method (CNOlist-methods), 8
randomize,CNOlist-method (CNOlist-class), 7
randomizeCNOlist, *8, 9*, 75
readBND, 76
readBNET, 77
readErrors (CNOlist-methods), 8
readErrors,CNOlist (CNOlist-methods), 8
readErrors,CNOlist-method (CNOlist-class), 7
readMIDAS, *7*, *53*, *59*, *67*, *75*, 77, *95*
readMidas (readMIDAS), 77
readSBMLQual, 78
readSIF, *6*, *12*, *14–16*, *30*, *35*, *43*, *49*, *54–56*, *67*, *74, 75*, 79, *82*
readSif (readSIF), 79
residualError, 81

setSignals<- (CNOlist-methods), 8
setSignals<-,CNOlist (CNOlist-methods), 8
setSignals<-,CNOlist,CNOlist-method (CNOlist-methods), 8
setSignals<-,CNOlist-method (CNOlist-class), 7
sif2graph, *42, 43*, *54–56*, 82
simulateTN, 83, *85, 86*, *88*
simulatorT0, 84
simulatorT1, *37*, *40, 41*, *74*, *83*, 85
simulatorT2, *40*, *87*
simulatorTN, 87

toSBML, 88
ToyModel, 89
ToyModel2, 90

write_bounds, 90
write_constraints, 91
writeDot, *67*, 92
writeErrors *(*CNOlist-methods*)*, 8
writeErrors,CNOlist *(*CNOlist-methods*)*, 8
writeErrors,CNOlist-method
*(*CNOlist-class*)*, 7
writeFile, 94
writeMIDAS, 95
writeNetwork, *67*, *93*, 96, *100*
writeObjectiveFunction, 98
writeReport, 99
writeScaffold, *93*, *100*, 101
writeSIF, 103
writeSif *(*writeSIF*)*, 103