Package ‘CNORfeeder’

March 28, 2017

Type Package
Title Integration of CellNOptR to add missing links
Version 1.14.0
Date 2015-14-08
Author F.Eduati
Maintainer F.Eduati <eduati@ebi.ac.uk>
Depends R (>= 2.15.0), CellNOptR (>= 1.4.0), graph
Suggests minet, catnet, Rgraphviz, RUnit, BiocGenerics, igraph
biocViews CellBasedAssays, CellBiology, Proteomics, Bioinformatics, NetworkInference
Description This package integrates literature-constrained and data-driven methods to infer signalling networks from perturbation experiments. It permits to extends a given network with links derived from the data via various inference methods and uses information on physical interactions of proteins to guide and validate the integration of links.
License GPL-3
LazyLoad yes
NeedsCompilation no

R topics documented:

   CNORfeeder-package ........................................ 2
   Binference ................................................. 3
   gaBinaryT1W .............................................. 4
   linksRanking .............................................. 6
   makeBTables ............................................. 7
   mapBTables2model ....................................... 9
   mapDDN2model ........................................... 10
   MIinference ............................................. 11
   PPIInigraph ............................................. 12
   UniprotIDdream ......................................... 13
   weighting .............................................. 14

Index 16
CNORfeeder-package

R package to integrate literature-constrained and data-driven methods to infer signalling networks from perturbation experiments

Description

CNORfeeder permits to extend a network derived from literature with links derived strictly from the data via various inference methods using information on physical interactions of proteins to guide and validate the integration of links. The package is designed to be integrated with CellNOptR.

Details

Package: CNORfeeder
Type: Package
Version: 1.0.0.
Date: 2012-11-22
License: GPLv2
LazyLoad: yes

Author(s)

F. Eduati Maintainer: F. Eduati <eduati@ebi.ac.uk>

References


Examples

library(CNORfeeder)

# this is an example of the main steps of the integrated CellNOptR - CNORfeeder pipeline

# load the data already formatted as CNOlist
data(CNOlistDREAM,package="CellNOptR")
# load the model (PKN) already in the CNO format
data(DreamModel,package="CellNOptR")
# see CellNOptR documentation to import other data/PKNs)

# A. INFERENCE - CNORfeeder
# FEED inference: codified in Boolean Tables
BTable <- makeBTables(CNOlist=CNOlistDREAM, k=2, measErr=c(0.1, 0))

# B. COMPRESSION - CellNOptR
# preprocessing step
model<-preprocessing(data=CNOlistDREAM, model=DreamModel)
Binference

# C. INTEGRATION - CNORfeeder
# integration with the compressed model
modelIntegr <- mapBTables2model(BTable=BTable, model=model, allInter=TRUE)
# see example in ?MapDDN2Model to use other reverse-engineering methods

# D. WEIGHTING - CNORfeeder
# integrated links are weighted more according to the integration factor integrFac
modelIntegrWeight <- weighting(modelIntegr=modelIntegr, PKNmodel=DreamModel,
CNOlist=CNOlistDREAM, integrFac=10)

# E. TRAINING - CellNOptR
initBstring<-rep(1,length(modelIntegr$reacID))
# training to data using genetic algorithm (run longer to obtain better results)
DreamT1opt<-gaBinaryT1W(
CNOlist=CNOlistDREAM,
model=modelIntegrWeight,
initBstring=initBstring,
maxGens=2,
popSize=5,
verbose=FALSE)

Binference

Bayesian network inference

Description

This function uses data (CNOlist) to infer a Bayesian network using the catnet package.

Usage

Binference(CNOlist, mode="AIC", tempCheckOrders=10,
maxIter=100, filename="BAYESIAN")

Arguments

CNOlist  a CNOlist structure, as produced by makeCNOlist
mode  a character, optimization network selection criterion such as "AIC" and "BIC",
to be used in cnSearchSA
tempCheckOrders  an integer, the number of iteration, orders to be searched, with constant temper-
ature, to be used in cnSearchSA
maxIter  an integer, the total number of iterations, thus orders, to be processed, to be used
in cnSearchSA
filename  name of the sif file saved, default BAYESIAN

Details

This function transforms the data in a format compatible with catnet package, infers the network
using the Stochastic Network Search as implemented in catnet (see cnSearchSA), computes the con-
sensus model of the models returned by cnSearchSA considering only links that have a frequency
of appearance greater than 0.1 and returns the model in the sif format.
**Value**

sif  the inferred data-driven network in sif format

**Author(s)**

F.Eduati

**See Also**

mapDDN2model

**Examples**

data(CNOlistDREAM, package="CellNOptR")

DDN<-Binference(CNOlistDREAM, tempCheckOrders=10, maxIter=100, filename="BAYESIAN")

---

**Description**

This function is the genetic algorithm to be used to optimise a model by fitting to data containing one time point. It is the function `gaBinaryT1` of CellNOptR modified in order to differently weights for the integrated links.

**Usage**

`gaBinaryT1W(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, maxSizeHashTable=5000)`

**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. If the linksWeight field is provided in model structure, all links are weighted according to that.
- **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
gaBinaryT1W

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 set 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).

maxSizeHashTable a hash table is use to store bitstring and related score. This allows the GA to be very efficient in the case of small models. The size of the hash table is 5000 by default, which may be too large for large models.

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


stringsTol the bitstrings whose scores are within the tolerance

stringsTolScores the scores of the above-mentioned strings

Author(s)

C. Terfve, T. Cokelaer, F.Eduati
References


See Also

gBinaryT1

Examples

data(CNOlistDREAM, package="CellNOptR")
data(DreamModel, package="CellNOptR")
model <- preprocessing(data=CNOlistDREAM, model=DreamModel)

BTable <- makeBTables(CNOlist=CNOlistDREAM, k=2, measErr=c(0.1, 0))
modelIntegr <- mapBTables2model(BTable=BTable, model=model, allInter=TRUE)

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

initBstring<-rep(1, length(modelIntegr$reacID))
# training to data using genetic algorithm (run longer to obtain better results)
DreamT1opt<-gBinaryT1W(CNOlist=CNOlistDREAM, model=modelIntegrWeight, initBstring=initBstring, maxGens=2, popSize=5, verbose=FALSE)

linksRanking

Ranking of links inferred from data

Description

This function uses data (CNOlist) to rank links based on measurement error model as used by FEED method to reverse-engineer the network.

Usage

linksRanking(CNOlist, measErr=c(0.1, 0), savefile=FALSE)

Arguments

CNOlist a CNOlist structure, as produced by makeCNOlist
measErr a 2 value vector (err1, err2) defining the error model of the data as sd^2 = err1^2 + (err2*data)^2, default to c(0.1, 0)
savefile TRUE to save the file in txt format, FALSE not. Default is FALSE.
**makeBTables**

**Details**

This function is similar to the fist step of FEED to reverse engineer the network strictly from data, i.e. the inference of Boolean tables, as described in (Eduati et al., PLoS ONE, 2010) and implemented in makeBTables. Links are ranked according to the upper limit value of parameter k allowing the presence of the link, where k is the parameter which is multiplied by the measurement error in order to assess the relevance of a link. The function returns link in decreasing order of importance and associate to each link a value (maximum value of k allowing the presence of the link) quantifying its relevance.

**Value**

this function returns a list with fields:

- **Lrank**  
  a matrix in which each link is associated with a numerical value, links are ordered in decreasing order of reliability

**Author(s)**

F. Eduati

**References**


**See Also**

makeCNOlist, makeBTables

**Examples**

```r
data(CNOlistDREAM, package="CellNOptR")
Lrank <- linksRanking(CNOlist=CNOlistDREAM, measErr=c(0.1, 0))
```

---

**makeBTables**  
*Make Boolean tables*

**Description**

This function uses data (CNOlist) to infer a Boolean table for each measured protein, codifying if a particular stimulus inhibitor combination affects the protein. A stimulus or an inhibitor significantly affects an output protein if it is able to modify its activity level of a quantity that exceeds the uncertainty associated with its measurement.

**Usage**

```r
makeBTables(CNOlist, k=2, measErr=c(0.1, 0), timePoint=NA)
```
Arguments

- **CNOlist**: a CNOlist structure, as produced by `makeCNOlist`
- **k**: a parameter that determine the threshold of significancy of the effect of stimuli and inhibitors, default to 2
- **measErr**: a 2 value vector (err1, err2) defining the error model of the data as \( sd^2 = err1^2 + (err2*data)^2 \), default to (0.1, 0)
- **timePoint**: the time point to be considered for the inference of the Boolean tables (i.e. "t1" or "t2"), if not specified all time points are considered

Details

This function computes the fist step of FEED to reverse engineer the network strictly from data, i.e. the inference of Boolean tables, as described in (Eduati et al., PLoS ONE, 2010). For each protein, a Boolean table is inferred having one columns for each stimulus and one row for each inhibitor. If a stimulus produces a significant effect on the activity level of the protein this is codified with a 1 in the corresponding column, if also the inhibitor affects the protein there is a 2 in the corresponding cell. The sign of the regulation is coded in separate tables.

Value

this function returns a list with fields:

- **namesSignals**: a vector of names of signals
- **tables**: a list with one Boolean table for each protein codifying the effect of stimuli (columns) and inhibitors (rows), 1 if the stimulus affect the protein, 2 if also the inhibitor does
- **NotMatStim**: has the same format as tables but just contains a 1 if the regulation has a negative effect, and 0 otherwise
- **NotMatInhib**: has the same format as tables but just contains a 1 if the regulation has a negative effect, and 0 otherwise

Author(s)

F.Eduati

References


See Also

`makeCNOlist`, `mapBTables2model`

Examples

data(CNOlistDREAM,package="CellNOptR")
BTable <- makeBTables(CNOlist=CNOlistDREAM, k=2, measErr=c(0.1, 0))
mapBTables2model

Integrate Boolean tables with the model

Description
This function infers the network from the Boolean tables and integrates it with the network encoded in the model (generally derived from prior knowledge), adding links that are missing.

Usage
mapBTables2model(BTable, model, optimRes=NA, allInter=TRUE, compressed=TRUE)

Arguments
BTable a BTable list, as created by makeBTables
model a model list, as created by readSif
optimRes a bit string with the reaction of the model to be considered, default considers all reactions
allInter one new link in the network can correspond to more links in the model, set it to TRUE if you want to add all possible links, FALSE to add only one link, default is TRUE
compressed this argument is used to decide how to deal with unmeasured and unperturbed nodes (white nodes). As general guideline, it should be set to TRUE if the PKN has been compressed in the preprocessing step, FALSE otherwise. Default is TRUE.

Details
The function receive as input the Boolean Tables, infers the data-driven network form them (as descibed in (Eduati et al., PLoS ONE, 2010)) and integrates it with the model, returning a new model with the integrated links. If the Model is not given as input (Model=NULL), the data-driven network is returned as model.

Value
a new model with the integrated links and an additional field:
indexIntegr a vector with the indexes of the integrated links

Author(s)
F.Eduati

References

See Also
readSif, readMIDAS, makeBTables
Examples

```r
data(CNOlistDREAM, package="CellNOptR")
data(DreamModel, package="CellNOptR")
model <- preprocessing(data=CNOlistDREAM, model=DreamModel)
BTable <- makeBTables(CNOlist=CNOlistDREAM, k=2, measErr=c(0.1, 0))
modelIntegr <- mapBTables2model(BTable=BTable, model=model, allInter=TRUE)
# modelIntegr$reacID[modelIntegr$indexIntegr] to see the integrated links
```

mapDDN2model

Integrate data-drive network with the model

Description

This function integrates the data-driven network (in sif format) with the network encoded in the model (generally derived from prior knowledge), adding links that are missing.

Usage

```r
mapDDN2model(DDN, model, CNOlist, allInter=TRUE)
```

Arguments

- **DDN**: a sif file encoding a data-driven network, as created by `Binference` or `MIinference`
- **model**: a model list, as created by `readSif`
- **CNOlist**: a CNOlist, as created by `makeCNOlist`
- **allInter**: one new link in the network can correspond to more links in the model, set it to TRUE if you want to add all possible links, FALSE to add only one link, default is TRUE

Details

The function receives as input a sif file with the data-driven network, as created by `Binference` or `MIinference`, and integrates it with the model, returning a new model with the integrated links.

Value

- a new Model with the integrated links and an additional field:
  - `indexIntegr`: a vector with the indexes of the integrated links

Author(s)

F.Eduati

See Also

- `readSif`, `readMIDAS`, `Binference`, `MIinference`
**MIinference**

Mutual information based network inference

**Description**

This function uses data (CNOlist) to infer a data-driven network using the mutual information based approaches ARACNe and CLR as implemented in the minet package.

**Usage**

```r
MIinference(CNOlist, method="ARACNE", PKNgraph=NULL, filename="ARACNE")
```

**Arguments**

- **CNOlist**: a CNOlist structure, as produced by `makeCNOlist`
- **method**: a character, the name of the method to be used: ARACNE or CLR. Default, ARACNE
- **PKNgraph**: a network to be used for comparison to assess the directionality of some links. Default is NULL.
- **filename**: name of the sif file saved, default ARACNE

**Details**

This function transforms the data in a format compatible with minet package, infers the network using aracne or clr as implemented in the minet package and returns the network in the sif format. It is important to notice that mutual information approaches do not allow for determining the directionality of the links thus both directions are considered. The function allows to give as input a network in graph format (graph package, see `sif2graph` to convert from sif to graph format) to be used as comparison to assess the directionality of some links, e.g. PKN.

**Value**

- **sif**: the inferred data-driven network in sif format

**Examples**

```r
data(CNOlistDREAM, package="CellNOptR")
data(DreamModel, package="CellNOptR")
model<-preprocessing(data=CNOlistDREAM, model=DreamModel)

## Not run:
DDN<-Binference(CNOlistDREAM, tempCheckOrders=10, maxIter=100,
                 filename="BAYESIAN")

modelIntegr<-mapDDN2model(DDN=DDN, model=model, CNOlist=CNOlistDREAM)

## End(Not run)
```
Author(s)

F. Eduati

References


See Also

mapDDN2model, sif2graph, model2sif

Examples

data(CNOlistDREAM, package="CellNOptR")
data(DreamModel, package="CellNOptR")
PKNgraph<-sif2graph(model2sif(DreamModel))

method="ARACNE"
#method="CLR"
DDN<-MIinference(CNOlist=CNOlistDREAM, method=method,
PKNgraph=PKNgraph, filename=method)

Description

The human protein-protein interaction network was built using a unified PPI dataset obtained as APID (Prieto, C. and De Las Rivas, J. 2006), by the combination of interactions coming from six source databases. The starting whole dataset was composed by 68488 human physical protein-protein interactions validated at least by one experimental method and reported in one article published in PubMed. From this dataset we obtained two PPI subsets with increasing confidence: a set of 28971 interactions validated by at least one binary experimental method (binary as defined in (De Las Rivas, J. and Fontanillo, C. 2010)); a set 6033 interactions validated by at least two experimental methods, one of them binary.

Usage

PPINigraph

Format

PPINigraph is an igraph with proteins as nodes and undirected links as physical protein interactions.

Source

This network was built for the analysis performed in (Eduati, F. et al. 2012)
UniprotIDdream

References


---

UniprotIDdream

Uniprot identifiers for proteins in DreamModel

Description

This data object contains the Uniprot identifiers corresponding to DreamModel of CellNOptR package, in order to associate them with the corresponding nodes in the protein-protein interaction network (PPINigraph).

Usage

UniprotIDdream

Format

UniprotIDdream is a list where each element is a protein of the DreamModel and is associated with the respective Uniprot identifiers.

Source

This data object is manually derived from the Uniprot database.

References


weighting

Weight integrated links.

Description

This function weights links integrated in the model using additional penalty and/or information from protein-protein interactions networks (PINs).

Usage

weighting(modelIntegr,PKNmodel,CNOlist,integrFac,UniprotID,PPI)

Arguments

- modelIntegr: the integrated model as created by mapDDN2model or mapBTables2model
- PKNmodel: the model of the original prior-knowledge network
- CNOlist: a CNOlist, as created by makeCNOlist
- integrFac: a number indicating the penalty for integrated links
- UniprotID: a list with the Uniprot identifiers of proteins in the PKN
- PPI: an igraph of the PIN to be used, if no network is provided (=NULL) this information is not used. Default is NULL.

Details

Integrated links are less reliable than links from the PKN, thus should be penalized in the optimization process. This function allows to include a penalty for integrated links (integrFact). Furthermore links can be differently prioritized based on information derived from protein interaction networks (PIN): the basic idea is that if, for a directed link A -> B integrated in the PKN, there is a corresponding path in the PIN, it is more plausible that there is a molecular pathway A -> B. Because shorter paths are more feasible, as a first approximation the shortest path length between A and B in the PIN can be used as a reliability score for the integrated link. Since the optimization is performed on a compressed version of the PKN, one link integrated in the compressed network generally corresponds to multiple possible links integrated in the PKN and the shortest path of all. The weight for each integrated link in the compressed network is thus computed as (1 + the inverse of the sum of the inverse of the corresponding PKN of the shortest paths in the PIN). A high quality network of known human physical protein-protein interaction assembled from multiple databases is provided with the package: interactions were included only if validated by at least one binary experimental method in a published paper and the number of experimental evidences was reported for each interaction.

Value

- modelIntegr: the input modelIntegr with an additional field: a vector with the weights of the integrated links

Author(s)

F.Eduati
weighting

See Also

mapDDN2model, mapBTables2model, gaBinaryT1W

Examples

data(CNOlistDREAM, package="CellNOptR")
data(DreamModel, package="CellNOptR")
data(UniprotIDdream, package="CNORfeeder")

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

BTable <- makeBTables(CNOlist=CNOlistDREAM, k=2, measErr=c(0.1, 0))
modelIntegr <- mapBTables2model(BTable=BTable, model=model, allInter=TRUE)

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

# weighting using PPI might take some minutes
## Not run:
data(UniprotIDdream, package="CNORfeeder")
data(PPINigraph, package="CNORfeeder")
modelIntegrWeight2 <- weighting(modelIntegr=modelIntegr, PKNmodel=DreamModel, CNOlist=CNOlistDREAM, integrFac=10, UniprotID=UniprotIDdream, PPI=PPINigraph)

## End(Not run)
Index

* Topic **datasets**
  - PPINigraph, 12
  - UniprotIDdream, 13

* Topic **package**
  - CNORfeeder-package, 2

Binference, 3, 10

CNORfeeder (CNORfeeder-package), 2
CNORfeeder-package, 2

gaBinaryT1, 4, 6

gaBinaryT1W, 4, 15

linksRanking, 6

makeBTables, 7, 7, 9
makeCNOnlist, 3, 6–8, 10, 11, 14
mapBTables2model, 8, 9, 14, 15
mapDDN2model, 4, 10, 12, 14, 15
MIinference, 10, 11
model2sif, 12

PPINigraph, 12

readMIDAS, 9, 10
readSif, 9, 10

sif2graph, 11, 12

UniprotIDdream, 13

weighting, 14