Package ‘sybilEFBA’

February 20, 2015

Type Package
Title Using Gene Expression Data to Improve Flux Balance Analysis Predictions
Version 1.0.2
Date 2015-02-15
Depends R (>= 2.10.0), sybil, Matrix
Suggests cplexAPI (>= 1.2.6), glpkAPI (>= 1.2.1)
Description Three different approaches to use gene expression data (or protein measurements) for improving FBA predictions.
LazyLoad yes
License GPL-3
Author Abdelmoneim Amer Desouki [aut, cre]
Maintainer Abdelmoneim Amer Desouki
<abdelmoneim.amer@uni-duesseldorf.de>
NeedsCompilation no
Repository CRAN
Date/Publication 2015-02-16 07:02:45

R topics documented:
sybilEFBA-package .............................................. 2
eFBA_gene ......................................................... 2
eFBA_rxn .......................................................... 5
FEcorr ............................................................. 7
findFluxGeneExpr .............................................. 9
findMDCFlux ...................................................... 11
gene2Rule ......................................................... 15
iAF1260 .......................................................... 17
iMM904 ........................................................... 17
iND750 ............................................................ 18

Index 19
Description

The package sybilEFBA implements some ideas to get flux predictions that are better correlated to gene expression data.

Details

<table>
<thead>
<tr>
<th>Package</th>
<th>sybilEFBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Version</td>
<td>0.0.1</td>
</tr>
<tr>
<td>Date</td>
<td>2013-06-03</td>
</tr>
<tr>
<td>License</td>
<td>GPL Version 3</td>
</tr>
<tr>
<td>LazyLoad</td>
<td>yes</td>
</tr>
<tr>
<td>Depends</td>
<td>sybil, methods</td>
</tr>
</tbody>
</table>

Author(s)

Abdelmoneim Amer Desouki

See Also

sybil eFBA_gene eFBA_rxn FECorr

eFBA_gene

Function: eFBA

Description

This function performs an expression based flux balance analysis. It takes into account the expression status of genes. First step calculates FB as the max biomass using standard FBA then uses FB=cTx as a new constraint in a MILP with a new objective function Minimize: Sum(expr(g) != flux(g) where expr(g)=1 if g is expressed , else 0 flux(g)=0 if (all reactions catalyzed by g) have no flux(Threshold Tf and sum of all), 0 otherwise.
Usage

eFBA_gene(model, expressionData, 
    Tf =0.01,#SYBIL_SETTINGS("TOLERANCE"),
    pct_objective=100, 
    lpdir = SYBIL_SETTINGS("OPT_DIRECTION"),
    solver = SYBIL_SETTINGS("SOLVER"),
    method = SYBIL_SETTINGS("METHOD"),
    solverParm=data.frame(CPX_PARAM_EPRHS=1e-6),
    testgpr=NULL,
    verboseMode = 2, ...)

Arguments

model An object of class modelorg.
expressionData a dataframe (gene expression data): gene ID, and expression status (0 :OFF or 1: ON)
pct_objective Biomass will be garnteed to be at least this value multiplied by max biomass calculated using standard FBA. Values are 0 to 100.
Tf Threshold in flux default value 1e-5. Fluxes with magnitude less than Tf are considered zeros.
lpdir Character value, direction of optimisation. Can be set to "min" or "max". Default: SYBIL_SETTINGS("OPT_DIRECTION")
solver Single character value. The solver to use. See SYBIL_SETTINGS for possible values. Default: SYBIL_SETTINGS("SOLVER").
method Single character value. The optimization algorithm to use. Possible values depend on the setting in solver. See SYBIL_SETTINGS for possible values. Default: LP_METHOD(SYBIL_SETTINGS).
solverParm A data frame containing parameters for the specified solver. Default: SOLVER_CTRL_PARM(SYBIL_SETTINGS).
testgpr Boolean flag containing flags of whether a reaction is considered or not
verboseMode An integer value indicating the amount of output to stdout: 0: nothing, 1: status messages, 2: like 1 plus with more details, 3: generates files of the LP problem. Default: 2.

Further arguments passed to optimizeProb. Argument solverParm is a good candidate.

Details

formulates the following LP: the problem: minimize:

\[
\begin{align*}
\| S \|_{1} & = b \| \quad \Rightarrow \quad \| c^T \|_{1} = FB \|_{1} \\
\| -1 \|_{1} - u_b \|_{1} \cdot y_i & \leq 0 \| \mid > u_b \cdot y_i \leq v_i \\
\leq u_b \cdot y_i + 1 \cdot u_b \|_{1} \leq 0 & | | \quad \Rightarrow \quad \| b \|_{1} \mid w_t l_b | 0 \| u_b \| w_t u_b | 1 \| 1 \| 1 \| b | 0 \| 1 \| 2 \| e_i = 1 \text{ where } e_i = 0 \\
\text{gene } i \text{ NOT expressed/1 otherwise}
\end{align*}
\]
Value

returns a list containing slots: origfluxes: original fluxes calculated by standard FBA. fluxes: new fluxes that better matches gene expression. gene_stat: state of genes (0 or 1) according to calculated fluxes stat: solution status returned from solver

Author(s)

Abdelmoneim Amer Desouki

See Also

modelorg, optimizeProb, eFBA_rxn

Examples

## Not run:
## The examples here require the package glpkAPI to be
## installed. If that package is not available, you have to set  
## the argument 'solver' (the default is: solver = "glpkAPI").

## load the example data set
data(iAF1260)
    model=iAF1260
allgenes=allGenes(model)
##set all genes to OFF
ex=cbind(LOCUS=allgenes,State=rep(0,length(allgenes)));
##set state of genes as required

## Optionally optimize only for a set of reactions
testgpr=rep(FALSE,react_num(model))
testgpr[react_id(model) %in% c('R_PYK','R_FLDL','R_FEROpp')]=TRUE

# read or set the expression status of genes : 0 not expressed,
#  1 Expressed, NA: unknown or ignored
ex_on=ex
ex_on[ex_on[,1] %in% c('b0123','b1354','b0684','b3924'),2]=1
ex_off=ex
# all genes are not expressed

slvr="glpkAPI"
# run two times, one with genes ON, the other with genes OFF
ser_on = eFBA_gene(model, ex_on,solver=slvr,verbose=3,testgpr=testgpr,Tf=0.001);
ser_off = eFBA_gene(model, ex_off,solver=slvr,verbose=3,testgpr=testgpr,Tf=0.001)

# print results
cbind(rxn=c('R_PYK','R_FLDL','R_FEROpp'),geneONfluxes=ser_on$rxn[testgpr, "newFlux"],
geneOFFfluxes=ser_off$rxn[testgpr, "newFlux"],
ruleStateON=ser_on$rxn[testgpr,"expr"],ruleStateOFF=ser_off$rxn[testgpr,"expr"],
FBA_flux=ser_off$rxn[testgpr,origFlux"])

# Count difference between these two solutions
sum(abs( as.numeric(ser_on$rxn[, "newFlux"])-as.numeric(ser_off$rxn[, "newFlux"]))>0.001)
eFBA_rxn

## Function: eFBA

### Description

This function performs an expression based flux balance analysis. It takes into account the expression status of genes. First step calculate FB as the max biomass using standard FBA use FB=cTx as a new constraint in a MILP with a new objective function Minimize: \( \text{Sum}(\text{expr}(\text{gpr}(r)) \neq \text{flux}(r)) = \text{flux}(r) \) where expr(r)=1 if gpr rule is ON (use boolean logic to evaluate rules depending on given gene state) flux(r)=0 if reaction r has no flux (Threshold Tf), 0 otherwise.

### Usage

```r
efba_rxn(model, expressionData,
    Tf =0.01,
    pct_objective=100,
    lpdir = SYBIL_SETTINGS("OPT_DIRECTION"),
    solver = SYBIL_SETTINGS("SOLVER"),
    method = SYBIL_SETTINGS("METHOD"),
    solverParm=data.frame(CPX_PARAM_EPRHS=1e-6),testgpr=NULL,
    verboseMode = 2)
```

### Arguments

- **model**: An object of class `modelorg`.
- **expressionData**: a dataframe (gene expression data): gene ID, and expression status (0 :OFF or 1: ON)
- **pct_objective**: Biomass will be guaranteed to be at least this value multiplied by max biomass calculated using standard FBA. Values are 0 to 100.
- **Tf**: Threshold in flux default value 1e-5. Fluxes with magnitude less than Tf are considered zeros.
- **lpdir**: Character value, direction of optimisation. Can be set to "min" or "max". Default: SYBIL_SETTINGS("OPT_DIRECTION").
- **solver**: Single character value. The solver to use. See SYBIL_SETTINGS for possible values. Default: SYBIL_SETTINGS("SOLVER").
- **method**: Single character value. The optimization algorithm to use. Possible values depend on the setting in solver. See SYBIL_SETTINGS for possible values. Default: LP_METHOD(SYBIL_SETTINGS).
- **solverParm**: A data frame containing parameters for the specified solver. Default: SOLVER_CTRL_PARM(SYBIL_SETTINGS).
### Details

formulates the following LP: the problem: minimize:

\[
\begin{align*}
\min & \quad c^T \cdot 0 - \frac{1}{2} x^T \cdot \frac{1}{2} \cdot w^T \cdot w \\
\text{subject to} & \quad \begin{cases}
S^T \cdot 0 = b \\
-1 \cdot y \cdot u \leq 0 \\
ub \cdot y \cdot i \leq vi
\end{cases}
\end{align*}
\]

where \( w = [0, 1] \) and \( y \) is the expression status of genes.

### Value

returns a list containing slots: origfluxes: original fluxes calculated by standard FBA. fluxes: new fluxes that better matches gene expression. gene_stat: state of genes (0 or 1) according to calculated fluxes stat: solution status returned from solver

### Author(s)

Abdelmoneim Amer Desouki

### See Also

`modelorg`, `optimizeProb`, `eFBA_gene`

### Examples

```r
## Not run:
## The examples here require the package glpkAPI to be installed. If that package is not available, you have to set the argument 'solver' (the default is: solver = "glpkAPI").

## load the example data set
data(iaf1260)
model = iaf1260
allgenes = allGenes(model)
## set all genes to OFF
ex = cbind(LOCUS = allgenes, State = rep(0, length(allgenes)));
## set state of genes as required

## Optionally optimize only for a set of reactions
testgpr = rep(FALSE, react_num(model))
testgpr[react_id(model) %in% c("R_PYK", "R_FLDR", "R_FEROp")]=TRUE

# read or set the expression status of genes : 0 not expressed,
# 1 Expressed, NA: unknown or ignored
ex_on = ex
ex_on[ex_on[,1] %in% c("b0123", "b1854", "b0684", "b3924")=1

ex_off = ex
```

# all genes are not expressed
slvr="glpkAPI"
#
# run two times, one with genes ON, the other with genes OFF
ser_on = eFBA_rxn(model, ex_on, solver=slvr, verbose=3, testgpr=testgpr, Tf=0.001);
ser_off = eFBA_rxn(model, ex_off, solver=slvr, verbose=3, testgpr=testgpr, Tf=0.001)

# print results
cbind(rxn=c('R_PYK','R_FLDR','R_FEROpp'),geneONfluxes=ser_on$rxn[testgpr, "newFlux"],
geneOFFfluxes=ser_off$rxn[testgpr, "newFlux"],
ruleStateON=ser_on$rxn[testgpr,"expr"],ruleStateOFF=ser_off$rxn[testgpr,"expr"],
FBA_flux=ser_off$rxn[testgpr,"origFlux"])

# Count difference between these two solutions
sum(abs(as.numeric(ser_on$rxn[, "newFlux"])-as.numeric(ser_off$rxn[, "newFlux"])))>0.001)

## End(Not run)

---

**FECorr**

**Function: FECorr: Flux Expression Correlation**

**Description**

This function uses FVA under different conditions to find fluxes that linearly correlates to corresponding gene expression.

**Usage**

```r
FECorr(model, nCond, initCond, geneExpressionData=NULL, RuleExpressionData=NULL, pct_objective=100, selected_rxns=NULL, only_identified_rules=FALSE, minExprFoldChange=0, lpdir = SYBIL_SETTINGS("OPT_DIRECTION"), solver = SYBIL_SETTINGS("SOLVER"), method = SYBIL_SETTINGS("METHOD"), solverParm=data.frame(CPX_PARAM_EPRHS=1e-6), verboseMode = 2)
```

**Arguments**

- `model` An object of class `modelorg`.
- `nCond` Number of conditions (FBA problems to be solved)
- `geneExpressionData` a data frame: geneID,Cond_id, ExpressionVal column rows are genes and column j+1 is representing gene expression under condition j
RuleExpressionData

rxn_id, cond_id, ExpressionVal

initCond

rxn_id, cond_id, lb, ub, obj_coef

pct_objective

Biomass will be guaranteed to be at least this value multiplied by max biomass calculated using standard FBA. Values are 0 to 100.

selected_rxns

optional parameter used to select a set of reactions not all, Boolean with the same length react_id(model)

only_identified_rules

give rxns containing genes with unidentified expression

minExprFoldChange

can be used to consider only genes with a significant change in expression level (i.e. min(expression(gene))*minExprFoldChange*2 must be less than or equal to max(expression(gene)))

lpdir

Character value, direction of optimisation. Can be set to "min" or "max". Default: SYBILL_SETTINGS("OPT_DIRECTION").

solver

Single character string giving the solver package to use. See SYBILL_SETTINGS for possible values. Default: SYBILL_SETTINGS("SOLVER").

method

Single character string giving the method the desired solver has to use. SYBILL_SETTINGS for possible values. Default: SYBILL_SETTINGS("METHOD").

solverParm

A named data frame or list containing parameters for the specified solver. Parameters can be set as data frame or list: solverParm = list(parm1 = val1, parm2 = val2) with parm1 and parm2 being the names of two different parameters and val1 and val2 the corresponding values. For possible parameters and values see the documentation of the used solver package (e.g. glpkAPI). Default: SYBILL_SETTINGS("SOLVER_CTRL_PARM").

verboseMode

An integer value indicating the amount of output to stdout: 0: nothing, 1: status messages, 2: like 1 plus with more details, 3: generates files of the LP problem. Default: 2.

Details

Main steps 1- Run FVA for all conditions, exclude rxns fixed in all conditions 2- Identify ruleExpression for set of rxns remaining from 1, 3- Fit Expr to FVA range. 4- Run findMDCFlux to find closest genome-scale flux 5- Recalculate correlation: Posterior, iFlux, ruleExpr

Value

returns a list containing slots: geneID, slope, intercept, base_level: OGOR: one gene one rxn iFlux: new fluxes calculated at all the given conditions which considers gene expression data to get a linear fit between gene expression and fluxes. It is a data frame containing the following columns rxn_id: reactionId in model, cond_id: lb, ub, objCoef, xpc_flux, fva_min, fva_max, RuleExprVal, iflux
findFluxGeneExpr

Author(s)
Abdelmoneim Amer Desouki

See Also
modelorg, optimizeProb, gene2Rule

Examples

```
# Not run:
library(sybil)
data(IAF1260)
model= iAF1260
# trivial test 1, linear levels for an unbounded rxn
ncnd=3

rxn="R_ADK1"
nc=length(react_id(model))
initCond=cbind(rxn_id=react_id(model),cond_id=rep(1,nc),
  lb=lowbnd(model),ub=uppbnd(model),obj=obj_coef(model))
initCond=rbind(initCond,cbind(rxn_id=react_id(model),
  cond_id=rep(2,nc),lb=lowbnd(model),ub=uppbnd(model),
  obj=obj_coef(model)))
initCond=rbind(initCond,cbind(rxn_id=react_id(model),
  cond_id=rep(3,nc),lb=lowbnd(model),ub=uppbnd(model),
  obj=obj_coef(model)))

cnds=(1:3)
gprExp=cbind(rxn_id=rxn,cond_id=1,expr_val=2)
gprExp=rbind(gprExp,cbind(rxn_id=rxn,cond_id=2,expr_val=4))
gprExp=rbind(gprExp,cbind(rxn_id=rxn,cond_id=3,expr_val=6))
fcflx=FCCorr(model,nCond=ncnd,initCond=initCond,
  RuleExpressionData=gprExp,selected_rxns=(react_id(model)==rxn),
  verboseMode=4);
fcflx[[2]][fcflx[[2]][2]==rxn,"ifl"]

    cor(as.numeric(fcflx[[2]][fcflx[[2]][2]==rxn,"expr_val"]),
     as.numeric(fcflx[[2]][fcflx[[2]][2]==rxn,"ifl"]))

# End(Not run)
```

findFluxGeneExpr  function to find minimum set of required genes to get a given flux distribution
Description

given a flux distribution, use gpr to formulate a MILP to find minimal set of required genes such that any GPR rule of a reaction carrying a nonzero flux must be evaluated to TRUE.

Usage

```r
findFluxGeneExpr(model, fluxes, threshold = 1e-06,
lpdir = SYBIL_SETTINGS("OPT_DIRECTION"),
solver = SYBIL_SETTINGS("SOLVER"), method = SYBIL_SETTINGS("METHOD"),
solverParm = SYBIL_SETTINGS("SOLVER_CTRL_PARM"), verboseMode = 2)
```

Arguments

- `model` An object of class `modelorg`.
- `fluxes` The flux distribution that we want to find the minimal set of genes able to produce it.
- `threshold` Threshold in flux default value 1e-6. Fluxes with magnitude less than threshold are considered zeros.
- `lpdir` Character value, direction of optimisation. Can be set to "min" or "max". Default: SYBIL_SETTINGS("OPT_DIRECTION").
- `solver` Single character value. The solver to use. See `SYBIL_SETTINGS` for possible values. Default: SYBIL_SETTINGS("SOLVER").
- `solverParm` A data frame containing parameters for the specified solver. Default: SOLVER_CTRL_PARM(SYBIL_SETTINGS).
- `verboseMode` An integer value indicating the amount of output to stdout: 0: nothing, 1: status messages, 2: like 1 plus with more details, 3: generates files of the LP problem. Default: 2.

Value

return list of genes with State: "ON" if it is required, "OFF": when it is not required.

Author(s)

Abdelmoneim Amer Desouki

See Also

`modelorg, optimizeProb, eFBA_gene`
Examples

```r
## Not run:

data(iAF1260)
model=iAF1260
allgenes=allGenes(model)
exoff=cbind(LOCUS=allgenes,State=rep(0,length(allgenes)));

testgpr=(gpr(model)!="")
table(testgpr)

slvr="glpkAPI"
seroff = eFBA_gene(model, exoff,solver=slvr,verbose=3,
testgpr=testgpr,Tf=0.0001)
mnfux=optimizeProb(model, algorithm = "mtf",solver=slvr);

# The minimum number of genes required to get each of the three fluxes
print(cbind(FBA_reqgenes=sum(geEFBA$State=="ON"),
          minTotFlx_reqGene=sum(gemnFBA$State=="ON"),
          FBA_reqGenes=sum(geFBA$State=="ON")) )

## End(Not run)
```

Description

Given a wildtype flux (some fluxes can be absent i.e NA), find FBA solution of network given by model, such that Sum(|v_i-v_wt|) is minimal (like IMOMA but with option to exclude some reactions)

Usage

```r
findMDCFlux(model, wtflux, objVal = NA, pct_objective=100,
             lpdir = SYBIL_SETTINGS("OPT_DIRECTION"),
             solver = SYBIL_SETTINGS("SOLVER"), method = SYBIL_SETTINGS("METHOD"),
             solverParm = data.frame(CPX_PARAM_EPRHS = 1e-06), verboseMode = 2)
```
findMDCFlux

Arguments

model An object of class modelorg.
wtfux desired flux distribution, when wtflux is NA its constraint won’t be included in LP.
pct_objective Biomass will be guaranteed to be at least this value multiplied by max biomass calculated using standard FBA. Values are 0 to 100.
objVal lower bound of objective function, if not set it will be set to maximum biomass.
lpdir Character value, direction of optimisation. Can be set to "min" or "max". Default: SYBIL_SETTINGS("OPT_DIRECTION").
solver Single character string giving the solver package to use. See SYBIL_SETTINGS for possible values. Default: SYBIL_SETTINGS("SOLVER").
method Single character string giving the method the desired solver has to use. SYBIL_SETTINGS for possible values. Default: SYBIL_SETTINGS("METHOD").
solverParm A named data frame or list containing parameters for the specified solver. Parameters can be set as data frame or list: solverParm = list(parm1 = val1, parm2 = val2) with parm1 and parm2 being the names of two different parameters and val1 and val2 the corresponding values. For possible parameters and values see the documentation of the used solver package (e.g. glpkAPI). Default: SYBIL_SETTINGS("SOLVER_CTRL_PARM").
verboseMode An integer value indicating the amount of output to stdout: 0: nothing, 1: status messages, 2: like 1 plus with more details, 3: generates files of the LP problem. Default: 2.

Value

return list with slot mdcflx containing the new calculated fluxes and status returned from solver.

Author(s)

Abdelmoneim Amer Desouki

See Also

modelorg, lmoma

Examples

## Not run:
## The function is currently defined as
function (model, wtflux, objVal = NA, lpdit = SYBIL_SETTINGS("OPT_DIRECTION"),
solver = SYBIL_SETTINGS("SOLVER"), method = SYBIL_SETTINGS("METHOD"),
solverParm = data.frame(CPX_PARAM_EPRHS = 1e-06), verboseMode = 2) {
  if (!is(model, "modelorg")) {
    stop("needs an object of class modelorg!")
  }
if (is.na(objVal)) {
    sol = optimizeProb(model, solver = solver, method = method, 
                      solverParm = solverParm)
    objVal = lp_obj(sol)
}

out <- FALSE
nc <- react_num(model)
nr <- met_num(model)
nd <- sum(!is.na(wtflux))
absMAX <- SYBIL_SETTINGS("MAXIMUM")
nRows = nr + 2 * nd + 1
nCols = nc + 2 * nd
LHS <- Matrix::Matrix(0, nrow = nRows, ncol = nCols, sparse = TRUE)
LHS[1:(nr, 1:nc)] <- S(model)
ii = matrix(c((nr + 1):(nr + nd), which(!is.na(wtflux))),
             ncol = 2)
LHS[ii] <- 1
if (nd == 1) {
    LHS[(nr + 1):(nr + nd), (nc + 1):(nc + nd)] <- 1
} else {
    diag(LHS[(nr + 1):(nr + nd), (nc + 1):(nc + nd)]) <- 1
}
ii = matrix(c((nr + nd + 1):(nr + 2 * nd), which(!is.na(wtflux))),
             ncol = 2)
LHS[ii] <- -1
if (nd == 1) {
    LHS[(nr + nd + 1):(nr + 2 * nd), (nc + nd + 1):(nc +
               2 * nd)] <- 1
} else {
    diag(LHS[(nr + nd + 1):(nr + 2 * nd), (nc + nd + 1):(nc +
               2 * nd)]) <- 1
}
LHS[(nr + 2 * nd + 1), 1:nc] <- obj_coef(model)
if (verboseMode > 2)
    print(sprintf("nrows=%d, ncols=%d, nd=%d, nc=%d, nr=%d",
                nRows, nCols, nd, nc, nr))
lower <- c(lowbnd(model), rep(0, 2 * nd))
upper <- c(uppbnd(model), rep(ABS_MAX, 2 * nd))
rlower <- c(rep(0, nr), wtflux[!is.na(wtflux)], -wtflux[!is.na(wtflux)],
            objVal)
rupper <- c(rep(0, nr), rep(ABS_MAX, 2 * nd + 1))
cobj <- c(rep(0, nc), rep(1, 2 * nd))
cenames = paste(c(rep("x", nc), rep("dp", nd), rep("dn", nd)),
                c(1:nc, which(!is.na(wtflux)), which(!is.na(wtflux))),
                sep = ".")
switch(solver, glpkAPI = {
    out <- vector(mode = "list", length = 5)
    prob <- glpkAPI::initProbGLPK()
    rtype <- c(rep(glpkAPI::GLP FX, nr), rep(glpkAPI::GLP LO,
               2 * nd))
})
```r
if (lpdir == "max") {
    rtype <- c(rtype, glpkAPI::GLP_LO)
} else {
    rtype <- c(rtype, glpkAPI::GLP_UP)
}
TMPmat <- as(LHS, "TsparseMatrix")
out[[1]] <- glpkAPI::addRowsGLPK(prob, nrows = nRows)
outj <- glpkAPI::addColsGLPK(prob, ncols = nCols)
mapply(setColNameGLPK, j = c(1:nCols), cname = cNames,
      MoreArgs = list(lp = prob))
glpkAPI::setObjDirGLPK(prob, glpkAPI::GLP_MIN)
out[[2]] <- glpkAPI::loadMatrixGLPK(prob, length(TMPmat@x),
    TMPmat@i + 1, TMPmat@j + 1, TMPmat@x)
out[[3]] <- glpkAPI::setColsBndsObjCoefsGLPK(prob, c(1:nCols),
    lower, upper, cobj)
out[[4]] <- glpkAPI::setRowsBndsGLPK(prob, c(1:nRows),
    lower, rupper, rtype)
parm <- sapply(dimnames(solverParm)[[2]], function(x) eval(parse(text = x)))
val <- solverParm[1,]
if (method == "interior") {
    glpkAPI::setInteriorParmGLPK(parm, val)
    out[[5]] <- TRUE
} else {
    glpkAPI::setSimplexParmGLPK(parm, val)
    out[[5]] <- TRUE
}
if (verboseMode > 2) {
    fname = format(Sys.time(), "glpk_ManhatDist_XY%m%d_%H%M.lp")
    print(sprintf("write problem: %s/\%s", getwd(), fname))
    writeLPGLPK(prob, fname)
    print("Solving...")
}
lp_ok <- glpkAPI::solveSimplexGLPK(prob)
lp_obj <- glpkAPI::getObjValGLPK(prob)
lp_stat <- glpkAPI::getSolStatGLPK(prob)
if (!is.na(lp_stat)) {
    lp_stat <- lp_ok
}
lp_fluxes <- glpkAPI::getColsPrimGLPK(prob)
}, cplexAPI = {
out <- vector(mode = "list", length = 4)
prob <- openProbCPLEX()
out <- setIntParmCPLEX(prob$env, CPX_PARAM_SCRIND, CPX_OFF)
chgProbNameCPLEX(prob$env, prob$lp, "ManhatenDist cplex")
setObjDirCPLEX(prob$env, prob$lp, CPX_MIN)
rtype <- c(rep("E", nr), rep("G", 2 * nd), "G")
TMPmat <- as(LHS, "TsparseMatrix")
out[[1]] <- newRowsCPLEX(prob$env, prob$lp, nRows, rlower,
    rtype)
out[[2]] <- newColsCPLEX(prob$env, prob$lp, nCols, cobj,
    lower, upper)
out[[3]] <- chgCoefListCPLEX(prob$env, prob$lp, length(TMPmat@x),
    TMPmat@i, TMPmat@j, TMPmat@x)
```

# Description

function to get state of expression of GPR rules from gene expression data

# rules in brief: #1-Complexes: average, 2-isoenzymes: sum #3-multifunctioning: divide by count

N.B.: GPR rules should be in form Sum-of-products (AND to OR)

# Usage

gene2Rule(model, geneExpr, selected_rxns = NULL)

# Arguments

model An object of class modelorg.
geneExpr a data frame: GeneID, expr_val for each gene.
selected_rxns optional parameter to select only a set of reactions.

# Value

return list with main slot:
ruleExpr: rxn_id,expr_val
Author(s)
Abdelmoneim Amer Desouki

See Also
modelorg, FECorr

Examples

```r
# Not run:
# Should be DIRECTLY executable !! ----
# Define data, use random,
# or do help(data-index) for the standard data sets.

# The function is currently defined as
function (model, geneExpr, selected_rxns = NULL)
{
  if (length(selected_rxns) == 0) {
    ugpr = as.vector(unique(gpr(model)))
  } else {
    ugpr = as.vector(unique(gpr(model)[selected_rxns]))
  }
  ugpr = ugpr[ugpr != ""]
  gprExpr = NULL
  for (v_rule in ugpr) {
    r1 = gsub("\\", " "", v_rule)
    r1 = gsub("\", " ( ", r1)
    pr = lapply(strsplit(unlist(strsplit(r1, " or "))), " and ",
      function(x) gsub("([^])", "\", x))
    expr_val = 0
    for (p in 1:length(pr)) {
      gene_ind = match(pr[[p]], geneExpr$geneID)
      if (length(gene_ind) < length(pr[[p]])) {
        warning(sprintf("Rule %d containing gene names not in geneID\n
term no: %d term: %s ",
          v_rule, p, pr[[p]][1]))
      } else {
        expr_val = expr_val + mean(geneExpr[gene_ind,
          "expr_val"])
      }
    }
    cnt = sum(gpr(model) == v_rule)
    gprExpr = rbind(gprExpr,
      cbind(rxn_id = react_id(model)[gpr(model) == v_rule],
        expr_val = expr_val/cnt, gpr = v_rule, cnt = rep(cnt, cnt))
  }
  return(gprExpr)
}
```

# End(Not run)
**iAF1260**  
*Escherichia coli Metabolic Model iAF1260*

**Description**

The dataset is a genome scale metabolic network of the *E. coli*. It consists of 2077 internal reactions, 304 exchange reactions and a biomass objective function.

**Usage**

```r
data(iAF1260)
```

**Format**

An object of class modelorg

**References**

Feist AM, Henry CS, Reed JL, Krummenacker M, Joyce AR, Karp PD, Broadbelt LJ, Hatzi-  
manikatis V, Palsson BØ (2007) A genome-scale metabolic reconstruction for Escherichia coli K-12  
MG1655 that accounts for 1260 ORFs and thermodynamic information. Mol Syst Biol 3: 121

---

**iMM904**  
*Saccharomyces cerevisiae Metabolic Model*

**Description**

The dataset is a genome scale metabolic network of the *Saccharomyces cerevisiae*. It consists of 1412 internal reactions, 164 exchange reactions and a biomass objective function.

**Usage**

```r
data(iMM904)
```

**Format**

An object of class modelorg

**References**

Mo ML, Palsson BO, Herrgard MJ: Connecting extracellular metabolomic measurements to intra-  
iND750  

Saccharomyces cerevisiae Metabolic Model iND750

Description

The dataset is a genome scale metabolic network of the *Saccharomyces cerevisiae*. It consists of 1149 internal reactions, 116 exchange reactions and a biomass objective function.

Usage

data(iND750)

Format

An object of class modelorg

References

Index

*Topic **EFBA gene**
  efBA_gene, 2
*Topic **EFBA reaction**
  efBA_rxn, 5
  FECorr, 7
*Topic **FBA**
  efBA_gene, 2
  efBA_rxn, 5
  FECorr, 7
*Topic **Manhaten distance closest flux**
  findMDCFlux, 11
*Topic **\textasciitilde kwd1**
  gene2Rule, 15
*Topic **\textasciitilde kwd2**
  findMDCFlux, 11
  gene2Rule, 15
*Topic **datasets**
  iAF1260, 17
  iMM904, 17
  iND750, 18
*Topic **flux distribution**
  findFluxGeneExpr, 9
*Topic **gene expression**
  FECorr, 7
  findFluxGeneExpr, 9
*Topic **iAF1260**
  iAF1260, 17
*Topic **package**
  sybilEFBA-package, 2
  sybilefbaMpackage, 2
  sybil, 2
  SYBIL_SETTINGS, 3, 5, 8, 10, 12
  sybileFBA (sybileFBA-package), 2
  sybileFBA-package, 2

efBA_gene, 2, 2, 6, 10
efBA_rxn, 2, 4, 5
FECorr, 2, 7, 16
findFluxGeneExpr, 9
findMDCFlux, 11
gene2Rule, 9, 15
iAF1260, 17