Package 'sybilccFBA'

January 27, 2015

Type Package

Title Cost Constrained FLux Balance Analysis: MetabOlic Modeling with ENzyme kineTics (MOMENT)

Version 1.0.0

Date 2013-10-23

Depends R (>= 2.10.0), sybil, Matrix

Suggests cplexAPI (>= 1.2.6), glpkAPI (>= 1.2.1)

Description implement cost constrained flux balance analysis as described in Adadi, R., Volkmer, B., Milo, R., Heinemann, M., & Shlomi, T. (2012). Prediction of Microbial Growth Rate versus Biomass Yield by a Metabolic Network with Kinetic Parameters, 8(7). doi:10.1371/journal.pcbi.1002575. Flux balance analysis is a technique to find fluxes in metabolic models at steady state. It is described in Orth, J.D., Thiele, I. and Palsson, B.O. What is flux balance analysis? Nat. Biotech. 28, 245-248 (2010)

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 2014-04-02 11:55:51

R topics documented:

`sybilccFBA-package` *Cost Constrained Flux Balance Analysis(ccFBA)*

Description

The package `sybilccFBA` implements some methods to get cost constrained fluxes. It is required to supply the molecular weights. It can be calculated from genome data using function `calc_MW`. Also requires kinetic data along with the model.

Details

Package:	sybilccFBA
Type:	Package
Version:	0.0.1
Date:	2013-06-03
License:	GPL Version 3
LazyLoad:	yes
Depends:	`sybil`, methods

Author(s)

Abdelmoneim Amer Desouki

See Also

`sybil` `cfba_moment`

`addGlcTrns` *add Glucose Transport constraint*

Description

add glucose transport constraint to the problem. Put an upperbound on glucose consumption.

Usage

```
addGlcTrns(prob, mod2)
```

Arguments

prob	lp problem
mod2	An object of class modelorg with only irreversible reactions. It can be sent to save time of recalculating it with each call.

Author(s)

Abdelmoneim Amer Desouki

See Also

modelorg

Examples

```
##---- 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 (prob, mod2)
{
    Si = 0.2
    Hxt1 = 41 * Si/(Si + 107)
    Hxt2 = 16.1 * Si/(Si + 2.9)
    Hxt3 = 18.5 * Si/(Si + 29)
    Hxt4 = 12 * Si/(Si + 6.2)
    Hxt5 = 14 * Si/(Si + 10)
    Hxt6 = 11.4 * Si/(Si + 1.5)
    Hxt7 = 11.7 * Si/(Si + 1.3)
    Gal2 = 17.5 * Si/(Si + 1.5)
    colid = getNumCols(lp = problem(prob)) + 1
    trnsCol = NULL
    rowind = getNumRows(lp = problem(prob)) + 1
    glcRxn = which(react_id(mod2) == "R_GLCt1")
    addRowsToProb(lp = problem(prob), i = rowind, type = "U",
        lb = 0, ub = 0, cind = list(c(trnsCol[1, "Col"], trnsCol[2,
            "Col"], trnsCol[3, "Col"], trnsCol[4, "Col"], trnsCol[5,
            "Col"], trnsCol[6, "Col"], trnsCol[7, "Col"], glcRxn)),
        nzval = list(c(-Hxt1, -Hxt2, -Hxt3, -Hxt4, -Hxt5, -Hxt6,
            -Hxt7, 1)), rnames = "glcTrns")
    return(prob)
  }
```

calc_MW *Calculate molecular weights*

Description

Calculate Molecular weights of different proteins using the genome .faa file.

Usage

```
calc_MW(aa_fname = "aa.txt", ptt_fname = "test2.ptt", faa_fname = "NC_000913.faa",
 nchrm = 1)
```

Arguments

aa_fname	file name of file containing lisT of amino acid names
ptt_fname	file name of file containing gene names with gene code
faa_fname	file name of file containing gene code and sequence of amino acids
nchrm	the number of chromosomes in the genome

Value

generate a file containing gene name , length, and molecular weight

Author(s)

Abdelmoneim Amer Desouki

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
## Not run:
aa_fname <- system.file("extdata", "aa.txt", package="sybilccFBA")
ptt_fname <- system.file("extdata", "test2.ptt", package="sybilccFBA")
faa_fname <- system.file("extdata", "NC_000913.faa", package="sybilccFBA")

geneCnt <- calc_MW(aa_fname,ptt_fname,faa_fname)
write.csv(file="geneCnt.csv",geneCnt)

## The function is currently defined as
"calc_MW"

## End(Not run)
```

cfba_moment *Function: cfba_moment: implement MOMENT method*

Description

This function uses GPR, kcat, and molecular weights to calculate fluxes according to MOMENT method.

Usage

```
cfba_moment(model,mod2=NULL, Kcat,MW=NULL,
selected_rxns=NULL,verboseMode=2,objVal=NULL,
RHS=NULL,solver=SYBIL_SETTINGS("SOLVER"),medval=NULL)
```

Arguments

model	An object of class modelorg.
mod2	An object of class modelorg with only irreversible reactions. It can be sent to save time of recalculating it with each call.
Kcat	kcat values in unit 1/S. Contains three slots: reaction id,direction(dirxn),value(val)
MW	list of molecular weights of all genes, using function calc_MW, in units g/mol
selected_rxns	optional parameter used to select a set of reactions not all, list of react_id
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.
RHS	the budget C, for EColi 0.27
objVal	when not null the problem will be to find the minimum budget that give the specified obective value(biomass)
solver	Single character string giving the solver package to use. See SYBIL_SETTINGS for possible values. Default: SYBIL_SETTINGS("SOLVER").
medval	median of Kcat values , used for missing values

Details

Main steps 1- Add variables for all genes 2- for each selected reaction: parse gpr, 3- Add variables accordingly and constraints 4-Add solvant constraint

Value

returns a list containing slots: prob:problem object that contains data and model sol: solution of the problem. geneCol: mapping of genes to variables in the problem.

Author(s)

Abdelmoneim Amer Desouki

References

Adadi, R., Volkmer, B., Milo, R., Heinemann, M., & Shlomi, T. (2012). Prediction of Microbial Growth Rate versus Biomass Yield by a Metabolic Network with Kinetic Parameters, 8(7). doi:10.1371/journal.pcbi.1002575

See Also

modelorg, optimizeProb

Examples

```
## Not run:
library(sybilccFBA)
data(iAF1260)
model= iAF1260
  data(mw)
  data(kcat)
   mod2=mod2irrev(model)

uppbnd(mod2)[react_id(mod2)=="R_EX_glc_e__b"]=1000
uppbnd(mod2)[react_id(mod2)=="R_EX_glyc_e__b"]=0
uppbnd(mod2)[react_id(mod2)=="R_EX_ac_e__b"]=0
uppbnd(mod2)[react_id(mod2)=="R_EX_o2_e__b"]=1000
lowbnd(mod2)[react_id(mod2)=="R_ATPM"]=0

  sol=cfba_moment(model,mod2,kcat,MW=mw,verbose=3,RHS=0.27,solver="glpkAPI",medval=3600*22.6)


## End(Not run)
```

---

cfba_moment_pw *Function: cfba_moment_pw: implement MOMENT method*

Description

This function uses GPR, kcat, and molecular weights to calculate fluxes according to MOMENT method. MOMENT pairwise OR like MATLAB implementation

Usage

```
cfba_moment_pw(model,mod2=NULL, Kcat,MW=NULL,
selected_rxns=NULL,verboseMode=2,objVal=NULL,
RHS=NULL,solver=SYBIL_SETTINGS("SOLVER"),medval=NULL)
```

Arguments

model	An object of class modelorg.
mod2	An object of class modelorg with only irreversible reactions. It can be sent to save time of recalculating it with each call.
Kcat	kcat values in unit 1/S. Contains three slots: reaction id,direction(dirxn),value(val)
MW	list of molecular weights of all genes, using function calc_MW, in units g/mol
selected_rxns	optional parameter used to select a set of reactions not all, list of react_id
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.
RHS	the budget C, for EColi 0.27
objVal	when not null the problem will be to find the minimum budget that give the specified obective value(biomass)
solver	Single character string giving the solver package to use. See SYBIL_SETTINGS for possible values. Default: SYBIL_SETTINGS("SOLVER").
medval	median of Kcat values , used for missing values

Details

Main steps 1- Add variables for all genes 2- for each selected reaction: parse gpr, 3- Add variables accordingly and constraints 4-Add solvant constraint

Value

returns a list containing slots: prob:problem object that contains data and model sol: solution of the problem. geneCol: mapping of genes to variables in the problem.

Author(s)

Abdelmoneim Amer Desouki

References

Adadi, R., Volkmer, B., Milo, R., Heinemann, M., & Shlomi, T. (2012). Prediction of Microbial Growth Rate versus Biomass Yield by a Metabolic Network with Kinetic Parameters, 8(7). doi:10.1371/journal.pcbi.1002575

See Also

modelorg, optimizeProb

Examples

```
## Not run:
library(sybilccFBA)
data(iAF1260)
model= iAF1260
  data(mw)
  data(kcat)
   mod2=mod2irrev(model)

uppbnd(mod2)[react_id(mod2)=="R_EX_glc_e__b"]=1000
uppbnd(mod2)[react_id(mod2)=="R_EX_glyc_e__b"]=0
uppbnd(mod2)[react_id(mod2)=="R_EX_ac_e__b"]=0
uppbnd(mod2)[react_id(mod2)=="R_EX_o2_e__b"]=1000
lowbnd(mod2)[react_id(mod2)=="R_ATPM"]=0

  sol=cfba_moment(model,mod2,kcat,MW=mw,verbose=3,RHS=0.27,solver="glpkAPI",medval=3600*22.6)


## End(Not run)
```

---

getRevFlux *getRevFlux*

Description

Given flux of irreversible model the function finds foreward minus backward flux

Usage

```
getRevFlux(model, modirrev, fdirrev)
```

Arguments

model	An object of class modelorg.
modirrev	An object of class modelorg with only irreversible reactions.
fdirrev	fluxes of irreversible model

Value

return fluxes according to the reversible model

Author(s)

Abdelmoneim Amer Desouki

See Also

mod2irrev

Examples

```
##---- 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, modirrev, fdirrev)
{
    fluxes = NULL
    for (r in (react_id(model))) {
        if (!react_rev(model)[react_id(model) == r]) {
          fluxes = rbind(fluxes, cbind(rxn = r, fwd = fdirrev[which(react_id(modirrev) ==
                r)], bwd = 0))
        }
        else {
          fluxes = rbind(fluxes, cbind(rxn = r, fwd = fdirrev[which(react_id(modirrev) ==
                paste(r, "_f", sep = ""))], bwd = fdirrev[which(react_id(modirrev) ==
                paste(r, "_b", sep = ""))]))
        }
    }
    return(fluxes)
  }
```

---

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

```
data(iAF1260)
```

Format

An object of class `modelorg`

References

Feist AM, Henry CS, Reed JL, Krummenacker M, Joyce AR, Karp PD, Broadbelt LJ, Hatzimanikatis 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

```
data(iMM904)
```

Format

An object of class `modelorg`

References

Mo ML, Palsson BO, Herrgard MJ: Connecting extracellular metabolomic measurements to intracellular flux states in yeast. BMC Syst Biol 2009,3:37.

kcat *Escherichia coli KCAT values used in MOMENT method*

Description

The dataset is a list of kcat values used in method MOMENT. Values are in unit 1/S.

Usage

```
data(kcat)
```

Format

A data frame with three columns

References

Adadi, R., Volkmer, B., Milo, R., Heinemann, M., & Shlomi, T. (2012). Prediction of Microbial Growth Rate versus Biomass Yield by a Metabolic Network with Kinetic Parameters, 8(7). doi:10.1371/journal.pcbi.1002575

mw — *Escherichia coli molecular weight values used in MOMENT method*

Description

The dataset is a list of molecular weights values used in method MOMENT. Values are in unit g/mol.

Usage

```
data(mw)
```

Format

A data frame with two columns

References

Adadi, R., Volkmer, B., Milo, R., Heinemann, M., & Shlomi, T. (2012). Prediction of Microbial Growth Rate versus Biomass Yield by a Metabolic Network with Kinetic Parameters, 8(7). doi:10.1371/journal.pcbi.1002575

readmodel — *read MOMENT model*

Description

create lp from lists generated from MATLAB MOMENT model.

Usage

```
readmodel(mat, mets, rxns, rbnds, cbnds, solver = "glpkAPI")
```

Arguments

mat	contain the constraints matrix
mets	list of metabolites
rxns	list of reactions and their bounds
rbnds	bounds of rows of constraint matrix
cbnds	bounds of columns of constraint matrix
solver	solver used to solve the lp, can be glpkAPI or cplexAPI

Value

return fluxes obtained using the lp.

Author(s)

Abdelmoneim Amer Desouki

References

Adadi, R., Volkmer, B., Milo, R., Heinemann, M., & Shlomi, T. (2012). Prediction of Microbial Growth Rate versus Biomass Yield by a Metabolic Network with Kinetic Parameters, 8(7). doi:10.1371/journal.pcbi.1002575

See Also

cfba_moment

Examples

```
##---- 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 (mat, mets, rxns, rbnds, cbnds, solver = "glpkAPI")
{
    nr = 6705
    nc = 4991
    nf = 3234
    nm = 1674
    LHS <- Matrix::Matrix(0, nrow = nr, ncol = nc)
    for (i in c(1:length(mat[, 1]))) {
        LHS[mat[i, 1], mat[i, 2]] = mat[i, 3]
    }
    cobj = c(rxns[, "ocf"], rep(0, nc - nf))
    if (solver == "cplexAPI") {
        prob <- cplexAPI::openProbCPLEX()
        out <- cplexAPI::setIntParmCPLEX(prob$env, cplexAPI::CPX_PARAM_SCRIND,
            cplexAPI::CPX_OFF)
        cplexAPI::chgProbNameCPLEX(prob$env, prob$lp, "Moment cplex")
        rtype <- c(rep("E", nm), rep("L", nr - nm))
        cplexAPI::setObjDirCPLEX(prob$env, prob$lp, cplexAPI::CPX_MAX)
        rupper = c(rbnds[1:nm, 1], rbnds[c((nm + 1):nr), 2])
        rupper[nr] = 0.27
        cplexAPI::newRowsCPLEX(prob$env, prob$lp, nrows = nr,
            rhs = rupper, sense = rtype)
        upper = cbnds[, 2]
        lower = cbnds[, 1]
        upper[2609] = 0
        upper[2729] = 1000
        upper[2835] = 0
        upper[2705] = 0
        upper[2774] = 0
        cplexAPI::newColsCPLEX(prob$env, prob$lp, nc, obj = cobj,
            lb = lower, ub = upper)
        print(sprintf("%s : step 2: nzijr....", format(Sys.time(),
```

```
            "%d-%m-%Y %X")))
        TMPmat <- as(LHS, "TsparseMatrix")
        cplexAPI::chgCoefListCPLEX(prob$env, prob$lp, nnz = length(TMPmat@x),
            ia = TMPmat@i, ja = TMPmat@j, ra = TMPmat@x)
        fname = format(Sys.time(), "Cplex_moment_%Y%m%d_%H%M.lp")
        print(sprintf("Writing problem to file: %s/%s  ...",
            getwd(), fname))
        cplexAPI::writeProbCPLEX(prob$env, prob$lp, fname)
        lp_ok <- cplexAPI::lpoptCPLEX(prob$env, prob$lp)
        print(lp_ok)
        sol = cplexAPI::solutionCPLEX(prob$env, prob$lp)
        print(sprintf("GLC upt=%f, AC=%f Pyr=%f fruc=%f Lac=%f",
            sol$x[2729], sol$x[2609], sol$x[2835], sol$x[2705],
            sol$x[2774]))
        colst = sol$x
    }
    else {
        prob <- glpkAPI::initProbGLPK()
        glpkAPI::addRowsGLPK(prob, nrows = nr)
        outj <- glpkAPI::addColsGLPK(prob, ncols = nc)
        glpkAPI::setObjDirGLPK(prob, glpkAPI::GLP_MAX)
        rtype <- c(rep(glpkAPI::GLP_FX, nm), rep(glpkAPI::GLP_UP,
            nr - nm))
        rlower = c(rbnds[1:nm, 1], rbnds[(nm + 1):nr, 2])
        rupper = rbnds[1:nr, 2]
        rupper[nr] = 0.27
        glpkAPI::setRowsBndsGLPK(prob, c(1:nr), lb = rlower,
            ub = rupper, type = rtype)
        upper = cbnds[, 2]
        lower = cbnds[, 1]
        cc <- glpkAPI::setColsBndsObjCoefsGLPK(prob, c(1:nc),
            lower, upper, cobj)
        TMPmat <- as(LHS, "TsparseMatrix")
        cc <- glpkAPI::loadMatrixGLPK(prob, length(TMPmat@x),
            ia = TMPmat@i + 1, ja = TMPmat@j + 1, ra = TMPmat@x)
        fname = format(Sys.time(), "glpk_eFBA_%Y%m%d_%H%M.lp")
        print(sprintf("Writing problem to file: %s/%s ...", getwd(),
            fname))
        glpkAPI::writeLPGLPK(prob, fname)
        print(format(Sys.time(), "Testing time : %Y%m%d %X Solving..."))
        lp_ok = glpkAPI::solveSimplexGLPK(prob)
        glpkAPI::return_codeGLPK(lp_ok)
        lp_stat = glpkAPI::getSolStatGLPK(prob)
        glpkAPI::status_codeGLPK(lp_stat)
        lp_obj = glpkAPI::getObjValGLPK(prob)
        colst = glpkAPI::getColsPrimGLPK(prob)
        newFlux = colst
        print(sprintf("GLC upt=%f, AC=%f Pyr=%f fruc=%f Lac=%f galt=%f",
            colst[2729], colst[2609], colst[2835], colst[2705],
            colst[2774], colst[2723]))
    }
    return(colst)
}
```

Index