Interfacing COBRA and *CellNetAnalyzer*

Author: Steffen Klamt [1] and Susan Ghaderi[2]: [1] Max Planck Institute for Dynamics of Comple Technical Systems, Magdeburg; [2] Systems Biochemistry Group, Luxembourg Centre for Systems Biomedicine

Reviewers:

INTRODUCTION

CellNetAnalyzer (CNA) is a MATLAB toolbox for exploring structural and functional properties of metabolic, signaling, and regulatory networks. Metabolic networks can be analyzed with stoichiometric and constraint-based modeling techniques, including flux balance analysis (FBA), metabolic flux analysis, elementary-modes analysis, computational strain design (based on minimal cut sets) and others. Signal transduction and (gene) regulatory networks are represented and analyzed as logical networks or interaction graphs but will not be considered herein. CNA can be used either via graphical user interface (GUI) or from command line via an application programming interface (API). In the following we will focus on CNA's API functions for constrained based modeling of metabolic networks which can easily be interfaced with the COBRA toolbox.

CNA and the COBRA toolbox have some overlap in their functionality for constraint-based modeling of metabolic networks (e.g., FBA, FVA, etc), however, each tool provides also methods not supported by the other. For example, unique functions of CNA are related to metabolic pathways analysis (calculation of elementary flux modes, elementary flux vectors, convex basis) and computational strain/network design via minimal cut sets. Here we will show how a COBRA model can be converted to a CNA model which can then be analyzed with CNA's API functions.

CNA can be downloaded at http://www.mpi-magdeburg.mpg.de/projects/cna/cna.html. Follow the instructions for installation and path settings as described in the manual which is distributed along with the software package. An online version of the manual as well as a tutorial can also be found at the CNA web site given above (see links at the end of this document).

In general, interfacing COBRA and CNA is rather simple as they both run under MATLAB and use partially similar model structures. If you have a COBRA model and you would like to use a function of CNA you first have to convert your COBRA model to a CNA model before you can apply API functions of CNA.

MATERIALS

- *Please ensure that the COBRA Toolbox has been properly installed and initialised.*
- *Also, you should install CNA (CellNetAnalyzer) software and initialise it. CNA web site (with manual):* https://www2.mpi-magdeburg.mpg.de/projects/cna/cna.html

EQUIPMENT SETUP

Requirements for using CellNetAnalyzer are:

- MATLAB Version 7.5 (Release 18) or higher.

- some functions require an LP or (M)ILP solver; CNA supports the optimization toolbox of MATLAB, GLPKMEX, and CPLEX).
- More information can be found on: https://www2.mpi-magdeburg.mpg.de/projects/cna/cna.html where also a how-to tutorial on CellNetAnalyzer is provided.

PROCEDURE

Before you start with CNA *codes*, you should initialise *the* COBRA Toolbox and CNA software by the following commands

```
initCobraToolbox;
```

```
% Add path to Cell Net Analyzer
CNAPath = '~/work/CellNetAnalyzer';
addpath(genpath(CNAPath));
startcna(1)

% Define the COBRA model
global CBTDIR
addpath([CBTDIR filesep 'tutorials' filesep 'additionalTutorials' filesep 'pathVectors'])
load('tutorial_COBRAmodel.mat')% a simple MATLAB struct with fields defined in the Documentati
% Define the directory (the place that CNA model will be saved there)
directory = 'Pathwaysvector';
```

Converting a COBRA model to a CNA model

The next step is the conversion of the COBRA model 'tutorial_COBRAmodel' to a CNA model (project) "cnap" which is achieved by the CNA function "cobra2cna":

```
cnap =  CNAcobra2cna(tutorial_COBRAmodel);
```

```
Field 'type' not defined. Initialized with '1' (for mass-flow).
Field 'nums' not defined. Initialized with 6.
Field 'numr' not defined. Initialized with 10.
Field 'specNotes' not defined. Initialized empty 'specNotes'.
Field 'specExternal' not defined. Initialized with zero vector (i.e. all species are configured as inter
Field 'reacNotes' not defined. Initialized empty 'reacNotes'.
Field 'reacVariance' not defined. Initialized for all reactions a variance level of 0.01.
Field 'reacDefault' not defined. Initialized with NaN vector (empty default values).
Field 'mue' not defined. Initialized according to existence of string 'mue' in 'reacID'.
Field 'reacBoxes' not defined. Initialized with default values.
Field 'macroComposition' not defined. Initialized as empty matrix (no macromolecules defined; other fiel
Field 'epsilon' not defined. Initialized with '1e-10'.
Field 'has_gui' not defined. Initialized with 'false'.
```

That's all. With the CNA project variable 'cnap' you can now call the API functions of CNA.

Most fields of the COBRA model have an associated field in the CNA model (e.g. `S -> stoichMat; lb -> reacMin; ub->reacMax, rxns->reacID, mets -> specID; metNames -> specLongName` etc.)

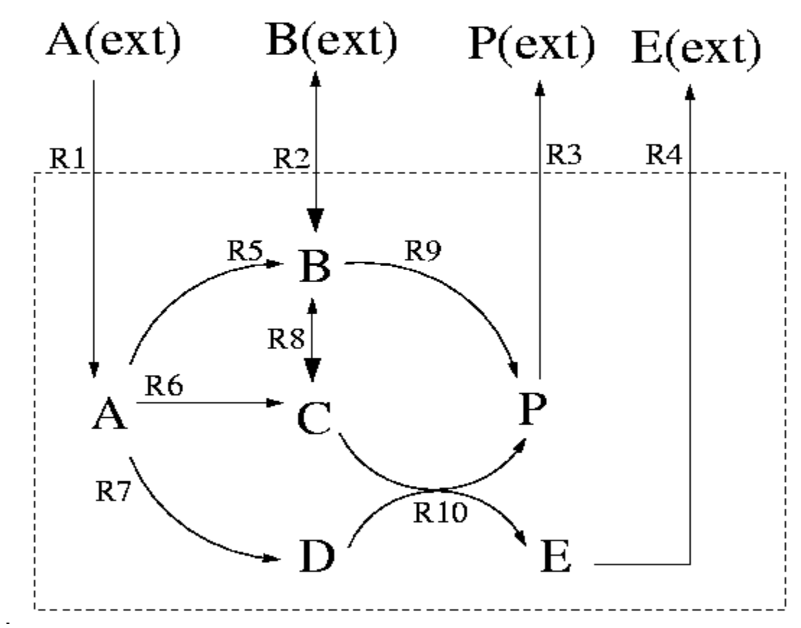
Relevant API functions of CNA

Here is a list of most relevant CNA functions for stoichiometric and constraint-based network analysis (the complete set of CNA's API functions can be found in the directory ../CellNetAnalyzer/code/api and a detailed description is given in the CNA manual):

- **CNAcomputeEFM:** Computes elementary (flux) modes (or, alternatively, a convex basis or elementary flux vectors) from a metabolic network model.
- **CNAcomputeCutsets:** Computes minimal cut sets from given sets of target modes and protected modes.
- **CNAMCSEnumerator:** Computes minimal cut sets via the "dual approach" directly from the network (elementary modes are not needed; can deal with genome-scale networks). See ref [3].
- **CNAregMCSEnumerator:** Similar to CNAMCSEnumerator, computes minimal cut sets via the "dual approach" directly from the network. As an additional feature, up- and downregulation of fluxes is allowed as additional intervention to cuts (knockouts) then yielding *regulatory cut sets* (see ref [4]).
- **CNAgeneMCSEnumerator:** Computes minimal cut sets via the "dual approach" directly from the network similar to CNAMCSEnumerator. But here, true "gene cut sets" (instead of "reaction cut sets) can be calculated (for this purpose, gene-enzyme-reaction associations must be provided).
- **CNAfindRegMCS:** Computes minimal (regulatory) cut sets via the "dual approach" directly from the network similar to CNAregMCSEnumerator. But here only a single cut set is calculated (no enumeration).
- **CNAfluxVariability:** Flux variability analysis.
- **CNAoptimizeFlux:** Flux balance analysis (optimization of fluxes with respect to a linear objective function).
- **CNAoptimizeYield:** Calculates a yield-optimal flux vector maximizing a given yield function (uses linear-fractional programming).
- **CNAplotPhasePlane:** Calculates and plots phase planes (or production envelops).
- **CNAplotYieldSpace:** Calculates and plots yield spaces.
- **CNAapplyCASOP:** Applies the CASOP strain design method (see ref [1]).
- **CNAremoveConsRel:** Detects and helps to remove conservation relations from the network (useful for certain calculations).
- **CNAcompressMFNetwork:** Loss-free compression of a stoichiometric network to a smaller one yielding, for example, the same set of elementary modes.
- **CNAreduceMFNetwork:** Reduction of network to a smaller subnetwork where specified protected functions are maintained (see ref [2]).
- **CNAcna2cobra:** Converts a (mass-flow) CNA model to a COBRA model.
- **CNAcobra2cna:** Converts a COBRA model to a CNA (mass-flow) model.

Example: calculating elementary flux modes of a COBRA model with *CellNetAnalyzer*

Here we show an example how to calculate elementary flux modes (EFMs) for a COBRA model using an API function of CNA. The metabolic network of `tutorial_COBRAmodel` with 10 metabolite (6 internal and four external) and 10 reactions (6 internal and four exchange) looks as follows:



The API function of CNA to calculate elementary flux modes (or, alternatively, elementary flux vectors or convex basis vectors; for a general introduction see [5]) is `CNAcomputeEFM`.

Input

Based on computing computing elementary modes and extreme pathways, the inputs can be different. You can see the optional inputs by the command

```
help CNAcomputeEFM
```

```
CellNetAnalyzer API function 'CNAcomputeEFM'
---------------------------------------------

--> Computes elementary (flux) modes / elementary (flux) vectors or a minimal generating set
    (convex basis) of flux cones or flux polyhedra associated with mass-flow networks.
    Two different scenarios can be considered:
    (i) In the homogeneous case, the solution space defined by the steady state assumption and
        reversibility constraints form a polyhedral (flux) cone. Here, elementary modes can be
        computed which are particular flux vectors of this cone with minimal support
        (= irreducible set of reactions with non-zero rate).
        The set of elementary modes includes all extreme rays of the flux cone, but usally many
        more (support-minimal) vectors. CNAcomputeEFM can also compute a convex basis of the flux cone
        In contrast to elementary modes, the convex basis is a minimal set of vectors sufficient
        to generate all flux vectors within the flux cone by non-negative linear combinations
        of convex basis vactors. As implemented herein, the convex basis will always represent
        a subset of the elementary modes. Note that the convex basis is only unique if there
        is no reversible flux vector.
    (ii) Specifying inhomogeneous constraints (e.g., fixing a reaction rate to a non-zero
        value or introducing non-zero upper and/or lower boundaries for the reaction rates) leads
        to the inhomogeneous case where the solution space becomes a more general flux polyhedron.
        CNAcomputeEFM computes then either the elementary vectors of the flux polyhedron
        (a generalization of elementary modes; the set of elementary vectors  will, for example,
        include all extreme rays and extreme points (if existent) of the resulting solution space).
        Again, alternatively, a convex basis  (= minimal set of generators spanning the resulting
        flux polyhedron) of the flux polyhedron can also be calculated.
        Note that the zero point will not be delivered, even if it is an extreme point of
        the flux polyhedron.

Usage: [efm,irrev,idx,ray,efv_with_zero] = CNAcomputeEFM(cnap, constraints,...
solver, irrev_flag, convbasis_flag, iso_flag, c_macro, display, efmtool_options)
```

```
 cnap is a CellNetAnalyzer (mass-flow) project variable and mandatory argument.

All other arguments are optional:

constraints: is either empty (=default) or a column vector or a struct specifying
    (homogeneous or inhomogeneous) constraints to be added to the standard (steady state
    and reversibility) constraints of the flux cone.
    If constraints is a COLUMN vector it must have dimension (cnap.numr x 1) and
        specifies excluded/enforced reactions: if(constraints(i)==0) then only those
        elementary modes/vectors will be computed that do not include reaction i.
        If constraints(i)~=0 and constraints(i)~=NaN then reaction i is enforced, i.e.
        only those elementary modes / vectors will be computed that involve reaction i
        (with a non-zero rate). For all other reactions choose constraint(i)=NaN.
        Several reactions may be suppressed/enforced simultaneously
    If constraints is a struct it may contain the following fields (all optional):
      - constraints.reaconoff: a column vector with dimension (cnap.numr x 1) which
                specifies excluded/enforced reactions: if(constraints.reaconoff)==0)
                then only those  elementary modes/vectors will be computed that do not
                include reaction i; if constraints.reaconoff(i)~=0 and constraints.reaconoff(i)~=NaN
                then reaction i is enforced, i.e. only those elementary modes / vectors will
                be computed that involve reaction i (with a non-zero rate); for all other
                reactions choose constraint(i)=NaN. Several reactions may be suppressed/enforced
                simultaneously.
      - constraints.lb: a column vector with dimension (cnap.numr x 1) which specifies lower
                boundaries for the reaction rates (choose constraints.lb(i)=NaN if there
                is no lower bound). Note that a lower bound of zero (irreversibilities) set
                in cnap.reacMin will be considered in any case and need not be specified via
                constraints.lb (but note that no other bound of cnap.reacMin will be considered!).
      - constraints.ub: a column vector with dimension (cnap.numr x 1) which specifies upper
                boundaries for the reaction rates (choose constraints.ub(i)=NaN if there
                is no upper bound). Note that no upper bound possibly defined in cnap.reacMax
                will be considered!.
      - constraints.eqto: a column vector with dimension (cnap.numr x 1) which specifies equalities
                constraints for the reaction rates (choose constraints.eqto(i) NaN if none is active
                for reaction i).
      - constraints.A (an h x cnap.numr matrix) and constraints.b (an h x 1 column vector) specify
                inequalities of the type A*r<=b (r: flux vector of the network). Note that
                constraints.lb, constraints.ub and constraints.eqto could also be formulated via
                constraints.A and constrainbts.b.

    One should be careful to not define inconsistent constraints, such as constraints.lb=3 and
    constraints.ub=2. Again note that cnap.reacMin and cnap.reacMax will not be considered for
    inhomogeneous constraints (cnap.reacMin is only used for marking reaction reversible (cnap.reacM
    and irreversible). However, you could, easily enforce those constraints  by setting
    constraints.lb=cnap.reacMin and constraints.ub=cnap.reacMax (Inf and 0 values in
    constraints(:2,) and constraints(:,3) should afterwards be set to NaN as they represent
    homogeneous constraints).
    If columns constraints.lb, constraints.ub, constraints.eqto and constraints.A / constraints.b
    are not specified or contain only zeros/NaN then elementary modes (if convbasis_flag=0; see belo
    or minimal generating sets (if convbasis_flag=1) of the flux cone will be computed (homogeneous
    problem). Any non-zero non-NaN value in in these vectors/matrix renders the problem inhomogeneou
    and the solution space to be a (flux) polyhedron. This function will then compute the elementary
    vectors of the flux polyhedron (if convbasis_flag=0) OR, again (if convbasis_flag=0), only a
    minimal set of unbounded  and bounded generators spanning the resulting flux polyhedron.
    The returned vector 'ray' indicates whether the i-th vector in 'efm' is unbounded (ray(i)==1)
    or bounded (e.g. an extreme point; ray(i)==0) (see also below). In case of the flux cone
    (elementary modes) they are always unbounded. The returned vector 'irrev' indicates whether an
    elementary mode/vector is reversible or not (see below).

solver: [0|1|3|4] 0: pure Matlab CNA function; 1: CNA mex files; 3: Metatool mex files;
    4: Marco Terzer's EFMtool (see http://www.csb.ethz.ch/tools/index).
    Default:4.
    Note that EFMtool cannot be used if some reactions are enforced (see constraints)
    and it cannot be used for calculating the convex basis.

irrev_flag: [0|1] whether (1) or not (0) to consider reversibilities of reactions
    0: all reactions are reversible (default: 1)
```

```
    convbasis_flag: [0|1] whether only a minimal generating set (convex basis) [1] is to be calculated
        or all elementary modes/vectors [0].  Default: 0.

    iso_flag: [0|1] whether parallel reactions should be considered only once (default: 0)

    c_macro: vector containing the concentrations (g/gDW) of the macromolecules
        if a variable biomass composition has been defined (cnap.mue not empty).
        Can be empty when cnap.mue or cnap.macroComposition is empty. If it
        is empty and cnap.mue is not empty then cnap.macroDefault is used.
        (default: cnap.macroDefault)

    display: controls the detail of console output; choose one of
        {'None', 'Iteration', 'All', 'Details'}
        default: 'All'

    efmtool_options: cell array with input for the CreateFluxModeOpts function.
        Default: {} (some options will be set by default; cf. console output for the actual options usec


  The following results are returned:

    efm: matrix that contains (row-wise) the elementary modes (or elemenatry vectors) or a minimal set c
        generators depending on the chosen scenario. The columns correspond to the reactions; the columr
        indices of efms (with respect to the columns in cnap.stoichMat) are stored in the returned varia
        idx (see below; note that columns are removed in efms if the corresponding reactions are not
        contained in any mode)

    irrev: vector indicating for each elementary mode/vector whether it is reversible(0)/irreversible (1

    idx:   maps the columns in efm onto the column indices in cnap.stoichmat,
     i.e. idx(i) refers to the column number in cnap.stoichmat (and to
     the row number in cnap.reacID)

    ray: indicates whether the i-th row (vector) in efm is an unbounded (1) or bounded (0) direction
        of the flux cone / flux polyhedron. Bounded directions (such as extreme points) can only arise
        if an inhomogeneous problem was defined (see also above for 'constraints').
```

So there are quite a number of different options and arguments. For example, `CNAsolver` can be 0, 1, 3 or 4, which

- 0: pure Matlab CNA function;
- 1: CNA mex files;
- 3: Metatool mex files;
- 4: Marco Terzer's EFMtool (see http://www.csb.ethz.ch/tools/index).

The Default is 4. Note that EFMtool cannot be used if some reactions are enforced (see constraints) and it cannot be used for calculating the convex basis.

However, for a standard calculation of elementary flux modes you just need to call the function with the CNA model (project) variable (which enforces default parameter setting). Hence, enter:

```
[efm,rev,idx,ray,efv_with_zero] = CNAcomputeEFM(cnap)
```

```
Network compression ...
```

```
Can't load log handler "ch.javasoft.util.logging.StandardOutHandler"
java.lang.ClassNotFoundException: ch.javasoft.util.logging.StandardOutHandler
java.lang.ClassNotFoundException: ch.javasoft.util.logging.StandardOutHandler
```

```
 at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
 at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
 at java.security.AccessController.doPrivileged(Native Method)
 at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
 at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
 at java.util.logging.LogManager$4.run(LogManager.java:808)
 at java.security.AccessController.doPrivileged(Native Method)
 at java.util.logging.LogManager.loadLoggerHandlers(LogManager.java:802)
 at java.util.logging.LogManager.initializeGlobalHandlers(LogManager.java:1406)
 at java.util.logging.LogManager.access$1500(LogManager.java:148)
 at java.util.logging.LogManager$RootLogger.accessCheckedHandlers(LogManager.java:1493)
 at java.util.logging.Logger.getHandlers(Logger.java:1350)
 at java.util.logging.Logger.log(Logger.java:612)
 at java.util.logging.Logger.doLog(Logger.java:641)
 at java.util.logging.Logger.log(Logger.java:664)
 at ch.javasoft.metabolic.compress.CompressionMethod.logUnsupported(CompressionMethod.java:125)
 at ch.javasoft.metabolic.compress.StoichMatrixCompressor.<init>(StoichMatrixCompressor.java:92)
Can't load log handler "ch.javasoft.util.logging.StandardErrHandler"
java.lang.ClassNotFoundException: ch.javasoft.util.logging.StandardErrHandler
java.lang.ClassNotFoundException: ch.javasoft.util.logging.StandardErrHandler
 at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
 at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
 at java.security.AccessController.doPrivileged(Native Method)
 at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
 at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
 at java.util.logging.LogManager$4.run(LogManager.java:808)
 at java.security.AccessController.doPrivileged(Native Method)
 at java.util.logging.LogManager.loadLoggerHandlers(LogManager.java:802)
 at java.util.logging.LogManager.initializeGlobalHandlers(LogManager.java:1406)
 at java.util.logging.LogManager.access$1500(LogManager.java:148)
 at java.util.logging.LogManager$RootLogger.accessCheckedHandlers(LogManager.java:1493)
 at java.util.logging.Logger.getHandlers(Logger.java:1350)
 at java.util.logging.Logger.log(Logger.java:612)
 at java.util.logging.Logger.doLog(Logger.java:641)
 at java.util.logging.Logger.log(Logger.java:664)
 at ch.javasoft.metabolic.compress.CompressionMethod.logUnsupported(CompressionMethod.java:125)
 at ch.javasoft.metabolic.compress.StoichMatrixCompressor.<init>(StoichMatrixCompressor.java:92)
```

```
Matrix size before network compression: 6  10
Matrix size after network compression: 2  6

Using 'CalculateFluxModes'
'/Applications/MATLAB_R2016b.app/sys/java/jre/maci64/jre/bin/java' -Xms1024m -Xmx1024m -XX:+UseParallel(

2017-12-20  15:34:34.758  main                     INFO     | ==========================================
2017-12-20  15:34:34.761  main                     INFO     | efmtool version 4.7.1, 2009-12-04 18:30:05
2017-12-20  15:34:34.761  main                     INFO     | Copyright (c) 2009, Marco Terzer, Zurich,
2017-12-20  15:34:34.761  main                     INFO     | This is free software, !!! NO WARRANTY !!!
2017-12-20  15:34:34.761  main                     INFO     | See LICENCE.txt for redistribution conditi
2017-12-20  15:34:34.762  main                     INFO     | ==========================================
2017-12-20  15:34:35.316  main    efm.output.mat   INFO     | estimated efms-per-file: 44000000
2017-12-20  15:34:35.346  main    efm.impl         INFO     | Elemetary flux mode computation
2017-12-20  15:34:35.347  main    efm.impl         INFO     | Implementation:
2017-12-20  15:34:35.349  main    efm.impl         INFO     | ..algorithm name   : SequentialDoubleDescr
2017-12-20  15:34:35.349  main    efm.impl         INFO     | ..model type       : NullspaceEfmModel
2017-12-20  15:34:35.350  main    efm.impl         INFO     | ..memory type      : InCoreMemory
2017-12-20  15:34:35.350  main    efm.impl         INFO     | ..output type      : MatFile
2017-12-20  15:34:35.350  main    efm.impl         INFO     | System:
2017-12-20  15:34:35.540  main    efm.impl         INFO     | ..hostname         : bmp00191.local
2017-12-20  15:34:35.540  main    efm.impl         INFO     | ..operating system : x86_64/Mac OS X/10.12
2017-12-20  15:34:35.541  main    efm.impl         INFO     | ..processors       : 4
2017-12-20  15:34:35.541  main    efm.impl         INFO     | ..vm                : Oracle Corporation/Ja
2017-12-20  15:34:35.541  main    efm.impl         INFO     | ..vm-spec          : Oracle Corporation/Ja
```

```
2017-12-20  15:34:35.543  main    efm.impl         INFO     | ..vm arguments     : [-Xms1024m, -Xmx1024m
2017-12-20  15:34:35.545  main    efm.impl         INFO     | ..memory, commited : 1029M
2017-12-20  15:34:35.545  main    efm.impl         INFO     | ..memory, used     : 21M
2017-12-20  15:34:35.545  main    efm.impl         INFO     | Config:
2017-12-20  15:34:35.545  main    efm.impl         INFO     | ..generator        : Efm
2017-12-20  15:34:35.545  main    efm.impl         INFO     | ..adj method       : pattern-tree-minzero
2017-12-20  15:34:35.545  main    efm.impl         INFO     | ..row ordering     : MostZerosOrAbsLexMin
2017-12-20  15:34:35.548  main    efm.impl         INFO     | ..arithmetic       : double (prec: -1 / ze
2017-12-20  15:34:35.548  main    efm.impl         INFO     | ..compression      : off
2017-12-20  15:34:35.548  main    efm.impl         INFO     | ..compr. methods   : []
2017-12-20  15:34:35.549  main    efm.impl         INFO     | ..normalize        : min
2017-12-20  15:34:35.549  main    efm.impl         INFO     | ..max threads      : 4
2017-12-20  15:34:35.549  main    efm.impl         INFO     | ..self test        : off
2017-12-20  15:34:35.549  main    efm.impl         INFO     | ..progress type    : None
2017-12-20  15:34:35.549  main    efm.impl         INFO     | ..progress part.   : 100
2017-12-20  15:34:35.550  main    efm.impl         INFO     | ..suppress         : []
2017-12-20  15:34:35.550  main    efm.impl         INFO     | ..enforce          : []
2017-12-20  15:34:35.550  main    efm.impl         INFO     | ..nosplit          : []
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..temp dir         : /Users/susan.ghaderi/
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..flag             : (none)
2017-12-20  15:34:35.551  main    efm.impl         INFO     | Distributed Config:
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..node count       : 2
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..nodes            : [localhost, localhost
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..vmargs           : [-Xmx800M, -Xmx500M]
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..command          : /usr/bin/java [vmargs
2017-12-20  15:34:35.551  main    efm.impl         INFO     | ..partition        : 256
2017-12-20  15:34:35.552  main    efm.impl         INFO     | ..cand. threshold  : 100000
2017-12-20  15:34:35.556  main    efm.impl         INFO     | original network: 2 metabolites, 6 reactio
2017-12-20  15:34:35.688  main    efm.impl         INFO     | stoich expanded has dimensions 2x8
2017-12-20  15:34:35.688  main    efm.impl         INFO     | kernel matrix has dimensions 8x6
2017-12-20  15:34:35.689  main    efm.impl         INFO     | TIME preprocessing: 354ms
2017-12-20  15:34:35.713  main    efm.impl         INFO     | iteration 0/2: 6 modes, dt=0ms. { next 1/2
2017-12-20  15:34:35.762  main    efm.impl         INFO     | iteration 1/2: 7 modes, dt=49ms. { next 2/
2017-12-20  15:34:35.776  main    efm.impl         INFO     | iteration 2/2: 10 modes, dt=13ms.
2017-12-20  15:34:35.776  main    efm.impl         INFO     | TIME iterate: 69ms
2017-12-20  15:34:35.776  main    efm.impl         INFO     | efm count before postprocessing: 10
2017-12-20  15:34:35.779  main    efm.impl         INFO     | efm count after filtering/consolidation: 8
2017-12-20  15:34:35.779  main    efm.impl         INFO     | uncompressing modes (can take a while)
2017-12-20  15:34:35.803  main    efm.output.mat   INFO     | estimated efms-per-file: 44000000
2017-12-20  15:34:35.865  main    efm.output.mat   INFO     | estimated efms-per-file: 44000000
2017-12-20  15:34:35.878  main    efm.output.mat   INFO     | estimated efms-per-file: 44000000
2017-12-20  15:34:35.902  main    efm.output.mat   INFO     | estimated efms-per-file: 44000000
2017-12-20  15:34:35.943  main    efm.impl         INFO     | TIME postprocessing: 167ms
2017-12-20  15:34:35.943  main    efm.impl         INFO     | overall computation time: 608ms

Don't forget to delete temporary files in the CellNetAnalyzer/code/ext/efmtool/tmp directory!


Final number of elementary modes: 8
```

```
efm =

     1    -1     0     0     0     1     0    -1     0     0
     1     0     1     0     1     0     0     0     1     0
     1     0     1     0     0     1     0    -1     1     0
     1     1     1     1     0     0     1     1     0     1
     1    -1     0     0     1     0     0     0     0     0
     0     1     1     0     0     0     0     0     1     0
     2     0     1     1     1     0     1     1     0     1
     2     0     1     1     0     1     1     0     0     1

rev =
   1
   1
   1
   1
   1
   1
```

```
     1
     1

idx =

     1     2     3     4     5     6     7     8     9    10
ray =

     1     1     1     1     1     1     1     1
efv_with_zero = 1
```

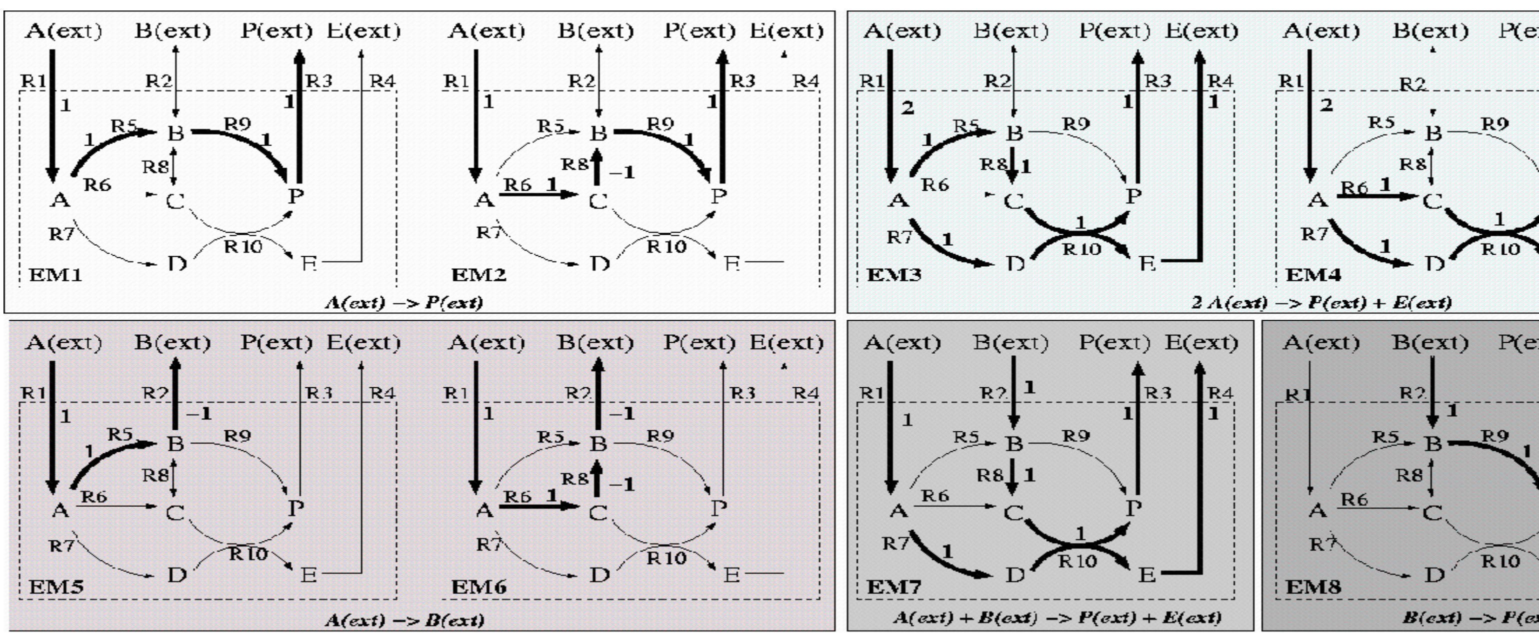
In this (standard) case, the EFMs are calculated with efmtool (without inhomogeneous constraints).

Output

The output of `pathVectors.m is`

- `efm`: contains the elementary flux modes in the rows. The columns contain the stoichiometric coefficients of the reactions - but note that the columns do not necessarily have the same order as in the stoichiometric matrix;
- `idx:` maps the columns in `efm` onto the original columns/reactions, i.e. `idx(i)` maps the `i`-th column (reaction) in `efm` to the column (reaction) number in the stoichiometric matrix (`cnap.stoichmat`). In our example, `idx` exhibits a 1:1 mapping. The vector `rev` indicates the reversibility of the calculated modes (0: reversible; 1: irreversible; hence, in this example, all modes are irreversible);
- `rev` : indicates the reversibility of the calculated modes (0: reversible; 1: irreversible; hence, in this example, all modes are irreversible).
- `ray`: indicates which of the calculated modes is unbounded (always true for EFMs);
- `efv_with_zero:` indicates whether the zero vector is part of the solution space (always true for EFMs).

These eoght elementary modes are shown in figuers 2:



If you now want to calculate the convex basis vectors (a subset of EFMs that are sufficient to span the flux cone; e.g. useful for calculation of extreme pathways) we have to set the `conbasis_flag` argument (the 5-th argument) to `1`. We also choose now the Metatool routine for calculating the convex basis (efmtool cannot be used for calculating convex bases). For the other arguments we chose the standard values:

```
[efm,rev,idx,ray,efv_with_zero] = CNAcomputeEFM(cnap,[ ],3,1,1)
```

```
Cannot use METATOOL MEX version: no executable MEX file found
Flux cone is pointed - reversible modes do not exist

Remaining metabolites: 2
Remaining reactions: 6 (4 reactions are irreversible)

column 1/2: C (6 preliminary vectors, 20-Dec-2017 15:34:36 cput: 0.04
column 2/2: B (5 preliminary vectors, 20-Dec-2017 15:34:36 cput: 0.08
4 convex basis vectors found  20-Dec-2017 15:34:36 cput: 0.09

Final number of convex basis vectors: 4

efm =

     0     1     1     0     0     0     0     0     1     0
     1    -1     0     0     1     0     0     0     0     0
     1     1     1     1     0     0     1     1     0     1
     1    -1     0     0     0     1     0    -1     0     0
rev =

     1
     1
     1
     1
idx =

     1     2     3     4     5     6     7     8     9    10
ray =

     1     1     1     1
efv_with_zero = 1
```

Converting a CNA model to a COBRA model

A CNA model ‘cnap’ can also be converted to a COBRA model by

```
 cbmodel =  CNAcna2cobra(cnap);
```

```
Removing 0 external metabolites
```

Then, COBRA functions can be applied to a (former) CNA model as well.

Note that if you convert a COBRA model to CNA and then back to a COBRA model, some model information can be lost since a CNA model does not capture all (potential) fields of a COBRA model. The following fields will definitely be maintained when converting a COBRA model to a CNA model and back:

- cbmodel.rxns

- cbmodel.mets
- cbmodel.S
- cbmodel.lb
- cbmodel.ub
- cbmodel.rev
- cbmodel.c
- cbmodel.b
- cbmodel.metNames

References

[1] Hädicke O and Klamt S. (2010) CASOP: A Computational Approach for Strain Optimization aiming at high Productivity. **Journal of Biotechnology,** 147, 88-101.

[2] Erdrich P, Steuer R, Klamt S. (2015) An algorithm for the reduction of genome-scale metabolic network models to meaningful core models. **BMC Systems Biology** 9:48.

[3] von Kamp A, Klamt S (2014) Enumeration of smallest intervention strategies in genome-scale metabolic networks. **PLoS Computational Biology**, 10:e1003378.

[4] Mahadevan R, von Kamp A, Klamt S. (2015) Genome-scale strain designs based on regulatory minimal cut sets. **Bioinformatics** 31:2844-2851.

[5] Klamt S, Regensburger G, Gerstl MP, Jungreuthmayer C, Schuster S, Mahadevan R, Zanghellini J, Müller S. (2017) From elementary flux modes to elementary flux vectors: Metabolic pathway analysis with arbitrary linear flux constraints. **PLoS Comput Biol** 13:e1005409.