Manual

CellNetAnalyzer

Version 2012.1

For an online version see:
http://www.mpi-magdeburg.mpg.de/projects/cna/manual/toc_frame.htm

This is the user's manual of *CellNetAnalyzer (*CNA*)*. CNA is a package for MATLAB and provides a comprehensive and user-friendly environment for structural and functional analysis of biochemical networks. CNA facilitates the analysis of metabolic (stoichiometric) as well as signaling and regulatory networks solely on their network topology, i.e. independent of kinetic mechanisms and parameters. CNA provides a powerful collection of tools and algorithms for structural network analysis which can be started in a menu-controlled manner within interactive network maps. Recently, API functionalities have been added to enable interested users to call algorithms of CNA from external programs. Applications of CNA can be found in systems biology, biotechnology, metabolic engineering, pharmacology, microbiology, chemical engineering.
CellNetAnalyzer is the successor and further development of *FluxAnalyzer*. *FluxAnalyzer* was originally developed for structural analysis of mass-flow (metabolic) networks; CNA provides in addition a framework for computational analysis of signal-flow networks.

The foundations and methods of network analysis are not described herein. The reader should be familiar with these methods; recommended references are given at the end of this manual. Some of these references are distributed together with CNA (see sub-directory 'doc').

CellNetAnalyzer is free for scientific purposes but the user has to agree with our licensing conditions when downloading CNA.
Requirements for using *CellNetAnalyzer* 9.1 are:

- MATLAB version 7.0 (R14) or higher. CNA will not work with lower versions.
- operating systems: Linux (32-bit), Windows XP (32-bit), Intel Mac
- some functions require the optimization toolbox of MATLAB (or/and GLPKMEX with GLPK library)

We have also started to support Octave 3.0 (www.octave.org), but this is work in progress and will not cover the CNA GUI.

CellNetAnalyzer has been developed at the Magdeburg Max Planck Institute by Steffen Klamt (since 2000) and Axel von Kamp (since 2007). Parts have been contributed by colleagues and collaborators:

- Regina Samaga (MPI Magdeburg): programmed the algorithm for identifying species equivalence classes for signal-flow networks and the function for comparing exp. data with the dependency matrix
- Tamon Stephen: improved the computation of minimal cut sets
- Oliver Hädicke (MPI Magdeburg): programmed CASOP algorithm
- Jan Krumsiek, Dominik Wittman and Fabian Theis developed the plugin "Odefy".
- We also thank many users for their valuable input.

CellNetAnalyzer is still (always ...) under construction. Thus, if you have any comments or suggestions for improvements, or if you find bugs please don't hesitate to contact us:

Steffen Klamt
Max-Planck-Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, D-39106 Magdeburg, Germany
klamt@mpi-magdeburg.mpg.de
phone: ++49-391-6110-480;   fax: ++49-391-6110-509

Thank you for using *CellNetAnalyzer*!

Contents:

0. Quick start and general hints

0.1 Installation

After unzipping the downloaded file at an appropriate place, *CellNetAnalyzer* (in the following abbreviated by CNA) consists of a main directory (here indicated as “CellNetAnalyzer“), the “code” directory with the MATLAB functions and libraries, and several subdirectories, each related to one network project:

../CellNetAnalyzer - main directory of the CNA
../CellNetAnalyzer - directory containing the manual and several related papers
../CellNetAnalyzer/code - directory containing MATLAB functions and libraries
../CellNetAnalyzer/coli - directory with specific files for the network project of “*E. coli*“
../CellNetAnalyzer/SmallExample - specific files for the network project “SmallExample“
../CellNetAnalyzer/<.....> - other network projects

Network projects created by the user will be added as new subdirectories in CNA’s main directory by default.

0.2 Starting and closing CellNetAnalyzer

For starting CNA proceed as follows:

- Go into the main directory of CNA and start MATLAB. Alternatively you have to ensure that the main directory of CNA is the current working directory after MATLAB has been started. If you want to use the Java functions, MATLAB *must* be started from the CNA main directory (in order to execute startup.m from this directory).

- Enter “**startcna**“. First, a window with general information on CNA appears (click OK), then the “Project Manager” comes up. Mass-flow (stoichiometric/metabolic) networks are indicated by “(M)” and signaling networks by “(S)”. Select a network project and click on „Start“.

- The selected network will be loaded resulting (after a while) in the display of the associated network maps in separate windows. Each network map has a menu item “CellNetAnalyzer” (see Fig. 2), from which the procedures related to network analysis can be started (see section 3.3 and 4.3). Some frequently used actions (e.g. load/clear/reset scenario) can be accessed from CNA’s toolbar.
  Note: CNA tries to display the network maps in full resolution. If this is not possible (e.g. due to a low resolution of the monitor, recommended is 1600x1200), the network map might be resized to a smaller resolution often leading to a badly readable display. In this case try to resize the window of the network map or use the menu command “Set original network map size” (see section 3.3). If this does not succeed, you can either use CNA’s zoom facilities (section 3.3. and 4.3) or, for a first use of the CNA, you can try to start a smaller network (e.g. the “Small Network Example”). These problems are further discussed in section 2.

- After loading a network project, the default values are displayed in the text boxes on the network map(s). The sign “#” indicates an unknown (undefined) value. You can now perform calculations provided by the menu “CellNetAnalyzer”. For example, when a mass-flow network has been loaded, your first action could be to determine a flux distribution resulting from some given values. For this purpose, write values into some text boxes (or use the default values) and choose in the “CellNetAnalyzer”-menu the item “Flux analysis ... ”. The calculable / balanceable rates that follow from the given values are displayed in the text boxes. Also take a look at the messages given by CNA in the command window during and after computations. After calculating a flux distribution (or anything else displayed in the text boxes) you can “Reset last values” or “Set default values” or “Clear all values” using the respective menu entries.

- A network session can be finished by closing any of the network maps. CNA will ask you whether you want to save your changes and whether the network variable is to be cleared from MATLAB’s workspace. A new feature of CNA 9.0 is that you may open several projects in parallel (even the same project can be loaded in several copies).

0.3. Hints for using CellNetAnalyzer with MATLAB under WINDOWS

CNA is programmed with MATLAB under LINUX as operating system (KDE desktop). Thus, the display of maps, text boxes, control panels, message boxes and so on are checked on this platform. When using MATLAB under WINDOWS, the calculations work also fine but, unfortunately, the appearance of these graphical user interfaces differs sometimes pretty much from LINUX and they do then not look very well. We will check regularly how CNA appears under WINDOWS and try to avoid large discrepancies (but some might nevertheless occur).

Special hints for WINDOWS user are:

- It can happen that the cursor is not visible when clicking in the text boxes. Nevertheless, you will be able to enter values.
- After entering values in text boxes **you have to press the ENTER-key**, otherwise the new values you have entered will probably not be recognized by MATLAB!

CNA has been tested for different MATLAB versions. Unfortunately, sometimes it might happen that the text boxes and the font size of the text appear differently. In such a case, you may try to adjust text box size and font size properly (by editing the project properties; see section 2).

0.4 MEX files

Some procedures of *CellNetAnalyzer* use (optionally) MEX files which have the extension “mexglx” (Linux), “dll” (WINDOWS) or “mexmaci” (Intel Mac). These files were written in C/C++ and contain an interface to MATLAB. They were compiled with the MATLAB “mex” command and can be called directly from MATLAB or a MATLAB program. The MEX-files provided with this version were compiled with MATLAB 6.5 resp. 7.5 and can be used under Linux (32-bit), Windows XP (32-bit) and Intel Mac. Other MATLAB versions (and other operating systems) may require a recompilation. There are three groups of MEX-files:

- those developed natively for CNA
- those for METATOOL and finally
- GLPK/GLPKMEX.

The source code for native CNA MEX-files is located in directory “code/mexfiles” and the provided function “elmodes_compile.m” can be used to start the compilation process. Note that this can only be done if a compiler for MATLAB was configured (see MEX setup). After successful compilation, copy the created executable MEX files into the “code/lib” (MATLAB versions 6.5 – 7.3) or “code/lib_2007b” (MATLAB version 7.5 and higher) directory. If a compilation is not possible or if any problems arise, the respective computation procedures should be started without using the MEX files,
Note that the source code for METATOOL- and GLPK-MEX-files is not distributed with CNA. Therefore METATOOL-MEX-files for all supported CNA platforms are distributed with the CNA.
Due to license restrictions, MEX-files for GLPK are not distributed with CNA. If you want to use GLPK (http://www.gnu.org/software/glpk/) you need the GLPKMEX interface (http://sourceforge.net/projects/glpkmex). However, for some combinations of OS and MATLAB version precompiled MEX-files can be separately downloaded from the CNA homepage (http://www.mpi-magdeburg.mpg.de/projects/cna/cna.html). In order to create your own MEX-files follow the instructions provided by GLPKMEX and place the necessary files (currently glpk.m, glpkmex.m and the corresponding glpkcc MEX-file) into the code/lib (MATLAB versions 6.5 – 7.3) or code/lib_2007b (MATLAB version 2007b and higher) directory.

0.5 New features

New features in CellNetAnalyzer 2012.1

- Version numbering format changed from 9.9 to 2012.1 (year + version number of the respective year)
- Mass-flow networks: the value for epsilon and the display format for fluxes can now be changed via the GUI.
- TRANSWESD can now also deal with maximal path lengths.

New features in CellNetAnalyzer 9.9

- For mass-flow-networks: the computation of elementary modes and convex bases (in flux cones) has been generalized to flux polyhedra (within the GUI and also as API function). This allows the specification of inhomogeneous constraints (e.g. fixing a rate to a particular value or restricting it within certain boundaries) and the subsequent calculation of elementary vectors, extreme points and rays characterizing/generating the solution space. This enables one, for example, to calculate the inner description of a polyhedron containing all optimal solutions for a given objective function.
- We have started to implement some computationally intensive algorithms in Java (which are then used if a Java Virtual Machine is installed). First methods ported to Java include CNATranswesd and some path/cycle calculations. To properly set the Java path, MATLAB must be started from the CNA main directory (in order to execute startup.m from this directory).

New features in CellNetAnalyzer 9.8

- For all networks: attributes of species, reactions and macromolecules can now directly be inspected and edited by right-clicking on the text boxes. It is not necessary anymore to search for a network element in the network composer and to open then the poperty dialog box.

- For mass-flow-networks: a two-dimensional yield-space plot can now be generated from the selected set of elementary modes (function is accessible from EM control panel).

New features in CellNetAnalyzer 9.7

- Major revision of the API functionality. Networks and scenarios can now be saved/loaded with or without GUI from command line.
- efmtool can now be used with rational numbers for computing elementary modes without numerical problems.

New features in CellNetAnalyzer 9.6

- Minimal cut sets (MCSs) calculation revised now allowing one to consider constraints: one may define sets of paths/cycles/modes that should not be hit (or where at least a minimum number of these paths/cycles/modes is not hit) by the cut sets to be computed. Relevant for MCSs calculations in signal-flow and mass-flow networks.
- Mass-flow networks: Flux Variability Analysis (in GUI + as API function). Useful for checking the feasible upper and lower boundaries of reaction rates for a given set of fixed fluxes.

New features in CellNetAnalyzer 9.5

- A project can now be started by double-clicking in the project manager.
- Bugfixes.
- Mass-flow networks:
  - CASOP: a new computational Approach for identifying intervebtion strategies for strain optimization and Metabolic Engineering (as menu item + API function) [18].
  - Elementary modes can now also be selected with respect to their (product) yield.
- Signal-flow networks:
  - API function for TRANSWESD: a new method for transitive reduction useful for reverse engineering of regulatory networks (interaction graphs) [19].

New features in CellNetAnalyzer 9.4

- CellNetAnalyzer has now a logo!
- some bug fixes
- Mass-flow networks:
  - New API functions for loading/saving scenarios
- Signal-flow networks:
  - New *Odefy* version integrated (which reads species and reaction values before starting a simulation).
  - New API functions for loading/saving scenarios

New features in CellNetAnalyzer 9.3

New online manual: http://www.mpi-magdeburg.mpg.de/projects/cna/manual/toc_frame.htm

Mass-flow networks:

- EFMTool ([15], developed by Marco Terzer, ETH Zürich), currently the most efficient method for computing elementary modes in large networks, can now

optionally be used (on-the-fly) within CNA.
- Editing network elements directly from element selector.

Signal-flow networks:
- Computation of minimal intervention sets accelerated.
- Editing network elements directly from element selector.
- New network project: EGF/ErbB receptor signaling.

New features in CellNetAnalyzer 9.2

Signal-flow networks:
- Improved computation of minimal intervention sets with more options; the new API function (CNAcomputeMIS) also allows for considering multiple scenarios
- Element selector allows efficient searching and highlighting of network elements

Mass-flow networks:
- Element selector allows efficient searching and highlighting of network elements
- New API function CNAoptimizeFlux (for Flux Balance Analysis)

New features in CellNetAnalyzer 9.1
- API functionality improved and extended considerably (computation of logical steady states, dependency matrix, cut sets and others now possible via API)

Signal-flow networks:
- Improved calculation of dependency matrix and shortest signed paths
- Automatic comparison of (experimental) data with dependency matrix
- *Odefy*: Plugin developed by Fabian Theis' group (Jan Krumsiek, Dominik Wittmann) at the Helmholtz-Zentrum Munich for translating logical model into ODE models and exporting/simulating these ODE models (odefy also allows to simulate Boolean models with synchronous and asynchronous switching)

Mass-flow Networks
- Improved import/export of SBML models

New features in CellNetAnalyzer 9.0

CNA 9.0 has undergone a major revision: the internal variable structure of CNA has been changed in order to facilitate API functionalities. Also, several new functions have been added and other improvements (e.g. toolbar buttons) make daily work with CNA simply more convenient. The main changes are:
- Computation of paths and loops in signal-flow networks has been revised: more convenient and more efficient.
- Computation of convex basis (also of extreme pathways) now more efficient as it also uses Metatool.
- Convex basis and elementary modes are computed with Metatool's integer arithmetic (if the conversion of the stoichiometric matrix into integers is possible).
- Computation of strongly connected components (signal-flow networks).
- Improved computation of Minimal Cut Sets
- Statistical features of minimal intervention sets can be computed (signal-flow networks)
- Numerical values in text boxes can be displayed in "heatmap style" (text boxes become gradually green, red, and yellow; depending on their value).

- Conversion of (logical) hypegraph into interaction graphs (signal-flow networks) can be done with optional removal of duplicated arcs.
- Toolbar allowing quick access to CNA's zoom tools and for opening, closing, saving and resetting scenarios.
- Color selector can be used to define the text box / text colors of a network project.
- A network project can now be started without GUI. The user may use CNA's API functionality directly from the command line or do his own computations using the CNA project variable.
- Several network projects can now be loaded in parallel.
- A network project can be cloned.
- API functionality has been extended considerably.

New features in CellNetAnalyzer 8.0

- Bar chart showing in/out fluxes around a metabolite (mass-flow networks)
- Computation of (species) equivalence classes in (logical) signal-flow networks.
- Computation of Minimal Intervention Sets strongly accelerated.
- The internal variable structure in CNA has been revised cleaning up the workspace
- New algorithm for computing shortest signed (pos./neg.) paths and cycles in interaction graphs as required for building the dependency matrix (see section 4.3).

New features in CellNetAnalyzer 7.0

- CNA provides now a (small) API (Application Programming Interface) which enables the exchange of data and variables between CNA and other MATLAB functions and the integration of user-created functions in CNA's menu.
- For flux optimizations, as an alternative to MATLAB's *linprog* function from the optimization toolbox, the user may no optionally use the GLPK (GNU Linear Programming Kit) library via the GLPKMEX interface.
- Signal flow networks: species values can now be directly fixed by entering a value in the respective text box (instead of fixing signal flows pointing into that species).
- Improved algorithm for minimal intervention sets.

New features in CellNetAnalyzer 6.3

- zoom tools (scrollbars; zooming in and out) enable better handling of large maps with large resolution
- in signal-flow networks, an interaction can be defined to be non-monotone or monotone (in earlier versions, monotone behavior was implicitly assumed; read section 4). This feature is relevant for logical networks with multiple-valued discrete states.

New features in CellNetAnalyzer 6.2

- comments and notes can now be added to reactions and species making a documentation possible

New features in CellNetAnalyzer 6.1

- "interaction networks" are now referred to as signal-flow (or signaling) networks (S) whereas stoichiometric (metabolic) networks are now referred to as "mass-flow" networks (M)

- as a generalisation of logical minimal cut sets, logical minimal intervention sets can be computed in signal-flow networks: intervention goals can now be achieved not only by cutting (knocking-out) components but also by a (constitutive) ACTIVATION (knock-in) of network species.
- an interaction can now be assigned an INCOMPLETE truth table. This is especially useful if is not clear whether several species have to be combined with an AND or with an OR.
- in signaling networks, all input species / reactions can be set to zero in one step ("closing input/output gates").

New features in CellNetAnalyzer / FluxAnalyzer 6.0

- Apart from stoichiometric networks, *CellNetAnalyzer* allows now also the construction and detailed analysis of signal-flow networks useful for analyzing the structure of signaling and regulatory networks. A number of tools is provided for studying graphical, hypergraphical and logical properties of signal-flow networks. Practically, *CellNetAnalyzer* facilitates now a functional topological analysis of any kind of cellular networks.
- for stoichiometric networks, most functions of *FluxAnalyzer* have been taken over in CNA, however, a few menu entries and functions have been revised.
- different sets of elementary modes (stored in several files) can now be merged.
- new MEX files (running also under MATLAB 7)

New features in FluxAnalyzer 5.3

- more efficient computation of elementary modes; a newly provided METATOOL module for computing the modes (which uses exclusively C-code and is therefore faster than the routines from *FluxAnalyzer*) can optionally be used
- some rates may be enforced to be involved in elementary modes to be computed; in some cases this can drastically reduce running time compared to the case where all modes are computed (however, sometimes it can even be slower)
- some more basic topological and graph-theoretical features are computed
- a new feature for minimal cut sets: searching for cut sets with lowest side-effects
- "variance weighted least squares method II" for flux analysis in redundant systems is no longer supported (hardly used in the literature; low relevance)

New features in FluxAnalyzer 5.2

- import/export SBML(Level 2)
- import/export from/to METATOOL format

New features in FluxAnalyzer 5.1

- using a new algorithm, the computation of elementary modes has been improved considerably, both with respect to time and memory demand.
- The menu function "Basic topological properties" has been revised and comprises some new features.
- Computation of (elementary) conservation relations: the user may choose to compute *all* or only *non-negative* elementary conservation relations.

New features in FluxAnalyzer 5.0

- Minimal cut sets (MCS; see [9]): calculation, display and detailed analysis. An MCS is a minimal set of reactions whose removal will make a certain function impossible; MCSs are useful for searching for targets for genetic modifications, for assessing network fragility, for finding a appropriate set of (flux) measurements making a certain reaction rate observable and for predicting inviable mutant phenotypes

New features in FluxAnalyzer 4.3

- calculation of graph-theoretical path lengths between metabolites (including the average path length and network diameter); can be done in a directed as well as undirected graph representation of the reaction network
- more intuitive arrangement of menu items
- separate calculation of (elementary) conservation relations
- flux analysis: calculation of variances of the estimated (calculable) rates in redundant systems
- new example networks

New features in FluxAnalyzer 4.2

- calculation of elementary (flux) modes /extreme pathways: the user has the option to consider isozymes only once
- elementary (flux) modes / extreme pathway matrices can be exported in ASCII format
- more convenient arrangement of text boxes on the network maps (see button “Move text boxes ...” in the network composer window)
- procedures for metabolic flux analysis do now automatically chosoe the procedures for redundant/non-redundant scenarios (menu-item: “Flux analysis ...”)
- text (“tooltip”) appears when the user moves and leaves the mouse pointer over a text box associated with a network element
- some new network properties can be calculated (e.g. detection of isozymes)
- manual construction of network projects is now described in the manual

New features in FluxAnalyzer 4.1

- computation of mutually excluding reaction pairs occurring in a set of elementary modes
- “Select all” button allows for quick selection of the complete set of elementary modes
- improved graphical display of the stoichiometric matrix
- computation of elementary modes now also possible in MATLAB 6.5 (incompatibility removed)

New features in FluxAnalyzer 4.0

- calculation of “control effective fluxes” with respect to elementary flux modes (applied in [2])
- histogram of product yields in elementary flux modes
- arbitrary normalization of elementary modes
- improved display of control panels for low monitor resolution and for WINDOWS

New features in FluxAnalyzer 3.1

- bar chart for comparing the fluxes of a given scenario
- histogram displaying the frequency distribution of the connectivity number of

metabolites
- histogram displaying the frequency distribution of the number of involved reactions in elementary modes
- new examples of network projects

New features in FluxAnalyzer 3.0
- metabolites can be defined as being external (not balanced) or internal (balanced)
- graphical display of the stoichiometric matrix including connectivity number of each metabolite
- the stoichiometric matrix can be exported as MATLAB or ASCII file
- enhanced algorithm for calculating elementary flux modes or of a convex basis; fast MATLAB-MEX-files can be used (needs MATLAB-Compiler)
- some small bugs have been removed
- start the FluxAnalyzer now by the command "startfa" (instead of "startmfa"); the menu item is now "FluxAnalyzer" (instead of MFA)

New features in FluxAnalyzer 2.4
- calculation of convex basis
- Minimum and Maximum rates may now also be –Infinity/ +Infinity
- checking the feasibility of a given network scenario (takes the given rates *and* the capacity constraints for the reaction rates into account) – see menu item "Check feasibility"
- layout of non-editable text boxes has been changed slightly

1. *CellNetAnalyzer*: Introduction and Overview

CNA is a package for MATLAB that runs in the environment of this widely used commercial program. Basically, CNA is composed of two parts (Fig. 1). The first part comprises network projects created and designed by the user. A network project is of type "mass-flow" (modeling material flows as in metabolic or other stoichiometric reaction networks) or of type "signal-flow" (modeling information or signal flows as in signaling or regulatory networks). Each network project contains an abstract (formal) network representation as well as one or several network graphics visualizing the network under investigation. The abstract network model is constructed by declaring network elements such as reactions or compounds, whereas the network graphics have to be imported and thus created by other programs (see below). Network model and graphical network representation can then be linked in CNA with the help of user interfaces (text boxes) leading to *interactive network maps* (Figure 2 shows an example). Interactive network maps are the central components of the GUI-based work with CNA. The text boxes facilitate input and output (e.g. of reaction rates) directly within the network visualization.

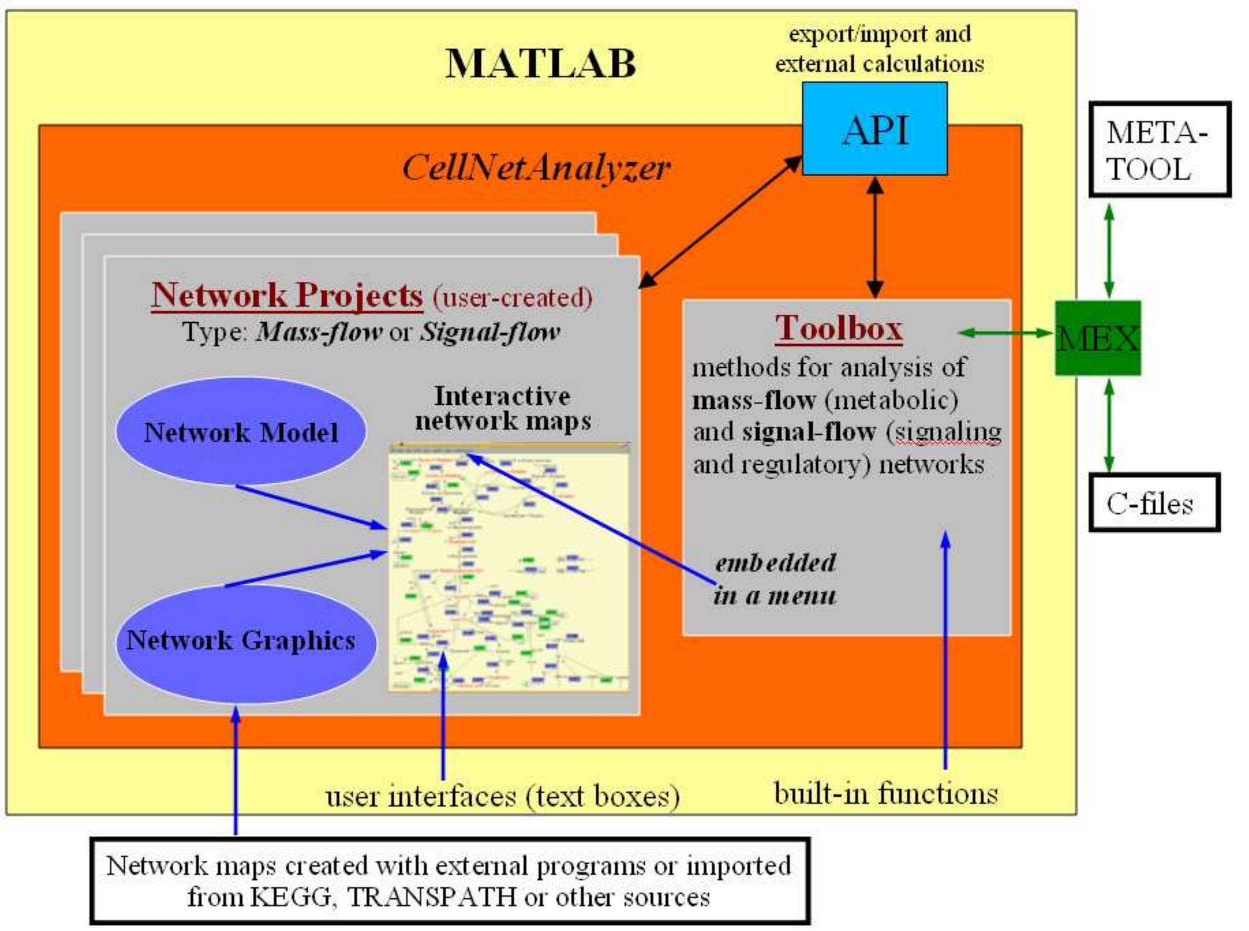

MATLAB
export/import and external calculations
API
CellNetAnalyzer
META-TOOL
Network Projects (user-created)
Type: Mass-flow or Signal-flow
Toolbox
methods for analysis of mass-flow (metabolic) and signal-flow (signaling and regulatory) networks
MEX
Interactive network maps
Network Model
embedded in a menu
C-files
Network Graphics
user interfaces (text boxes)
built-in functions
Network maps created with external programs or imported from KEGG, TRANSPATH or other sources

Fig. 1: Internal structure and architecture of CellNetAnalyzer

The second part of CNA comprises a toolbox of functions and procedures which, depending on the network type, facilitate either stoichiometric network analysis (including steady-state methods such as flux analysis, pathway analysis, flux optimization as well as graph-theoretical analyses and others) or the analysis of signal-flow networks (logical (Boolean) analysis, signaling paths and feedback loop analysis, dependency analysis). They can be started conveniently by a pull-down menu (installed in all network maps) and the user does not have to be

aware of mathematical details. Some of these procedures were developed by the help of MATLAB’s built-in functions, others were developed completely new. A few algorithms use the MEX interface of MATLAB to call precompiled C/C++ functions (including those provided by Metatool). This is in particular useful for speeding up extensive computations.
Finally, we have started to implement an API, which enables to import/export a network from/to the MATLAB workspace and to call some selected CNA functions without having the CNA GUI loaded (chapter 7).

Interactive network maps

When the user initializes a new network project in *CellNetAnalyzer*, he can provide a graphical representation of his network (albeit this is not mandatory, see hints below). CNA itself is not a pathway drawing engine and can therefore not be used to create the network maps from the network model! The main reason for avoiding an automatic creation of such maps is the fact that algorithms for an automatic drawing of networks, particularly of metabolic ones, using a symbolic network description are complex and do not always lead to a representation as desired by the user. Instead, arbitrary external programs (such as “xfig”, “Paint” or “CorelDraw”, “inkscape”) can be employed to generate the maps with a design the user prefers. Arbitrary annotations are possible because these are independent of the abstract network model in CNA. Network graphics may also be obtained from other sources (e.g. KEGG, BioPath, CellDesigner). A new opportunity for signal-flow networks is provided by ProMoT (also developed at the MPI in Magdeburg; www.mpi-magdeburg.mpg.de/projects/Promot): with this program a logical network can be constructed in a visual manner and then exported (together with the abstract model) to CNA.
With one exception (see below), MATLAB can currently only read pixel-based graphics (formats tif, bmp, pcx; the jpg format can but should not be used). Therefore, the map size (in pixels) should ideally not be larger than the resolution of the monitor since the graphics can otherwise not be displayed in full resolution. However, the new zoom tools of CNA (from version 6.3 on) facilitate also the use of large pixel-based network maps: zooming in and out and scrollbars provide a convenient way to navigate through large network maps.
A special opportunity for creating network maps are MATLAB figures (file extension “fig”) which may be drawn by using the “Plot editing tools” of MATLAB (which are rather limited, however). MATLAB figures are vector-oriented and can therefore easily be resized without loss of quality. Once a network map has been resized, the menu command “Set original map size” resizes all network maps to the original resolution (or, if the resolution of the map is larger than that of the display, to the largest possible size).
When loading a network project in CNA each registered network map serves as a background in a MATLAB figure, where text boxes - associated with elements of the abstract network model - can be placed by the user. The linkage of network graphics and user interfaces leads to *interactive network maps*. The fact that the network visualization is not directly coupled to the abstract network representation allows one to design network graphics in an arbitrary way.

NOTE: If the user does not have any map visualizing his network (e.g. when importing a large network from a database) and if he does not want to create one, he still has two possibilities: (i) one may choose a “dummy” picture, for example the one provided with CNA (dummy.pcx). Then, obviously, the user will not have a meaningful visualization of his network. However, as the internal network structure of a project will be independent from the network map, he can nevertheless compose a network and analyze this network with the provided menu functions. The text boxes can be placed anywhere or even outside of the map. Results, for example a computed set of elementary modes or a calculated flux distribution can usually be exported in ASCII files. (ii) A second possibility is to load the network without a GUI (either using the ‘Start w/o GUI’ button in the project manager or just from command line; see chapter 7). One may then analyze the network with the provided API functions of CNA (chapter 7).

As an example, Fig. 2 shows the interactive network map belonging to a simple stoichiometric network project created by CNA. The underlying network map was drawn by an external graphic program (xfig). Graphical user interfaces (text boxes) refer to reactions or biomass components linking the symbolically defined network (not shown) with the graphical scheme resulting in the interactive network map. The menu-item „CellNetAnalyzer“ in the menu-bar of the interactive map provides the pull-down menu with functions for analyzing and editing the network. For some frequently used functions (such as loading or saving a scenario), the CNA toolbar provides a short-cut to start the related actions.

Section 2 will explain how network projects can be created in CNA and how the existing projects can be managed with the Project Manager. The set-up of abstract network models and of interactive maps is described in sections 3.1, 3.2 (mass-flow networks) and 4.1, 4.2 (signal-flow networks), respectively. The toolboxes facilitating integrated network analysis are then described in detail in sections 3.3 and 4.3, respectively.

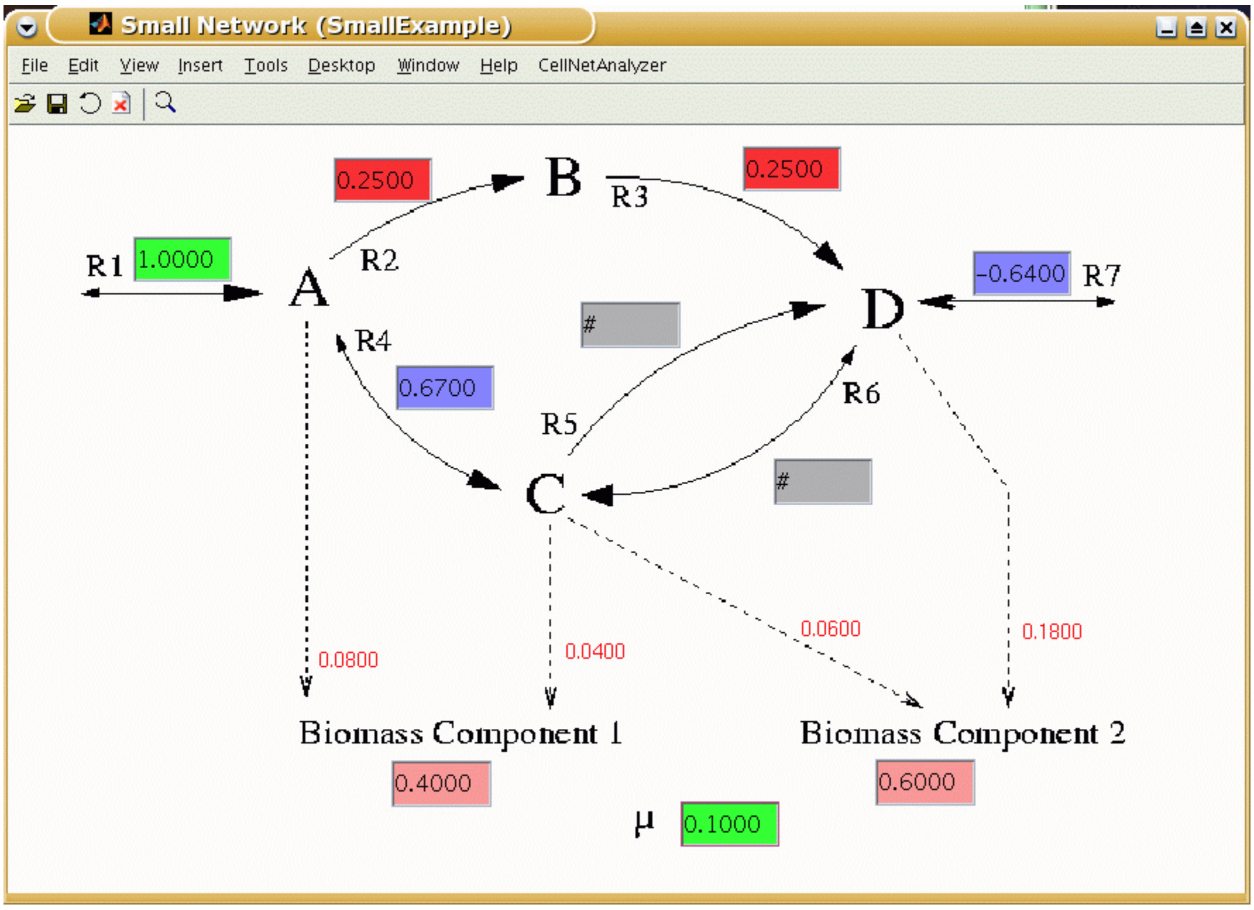
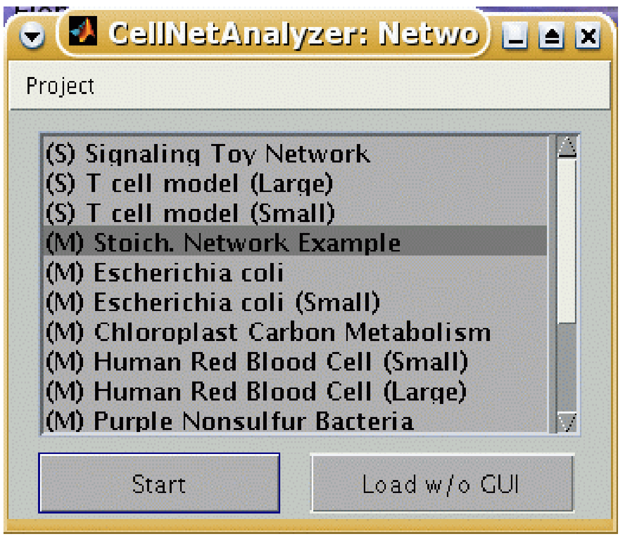


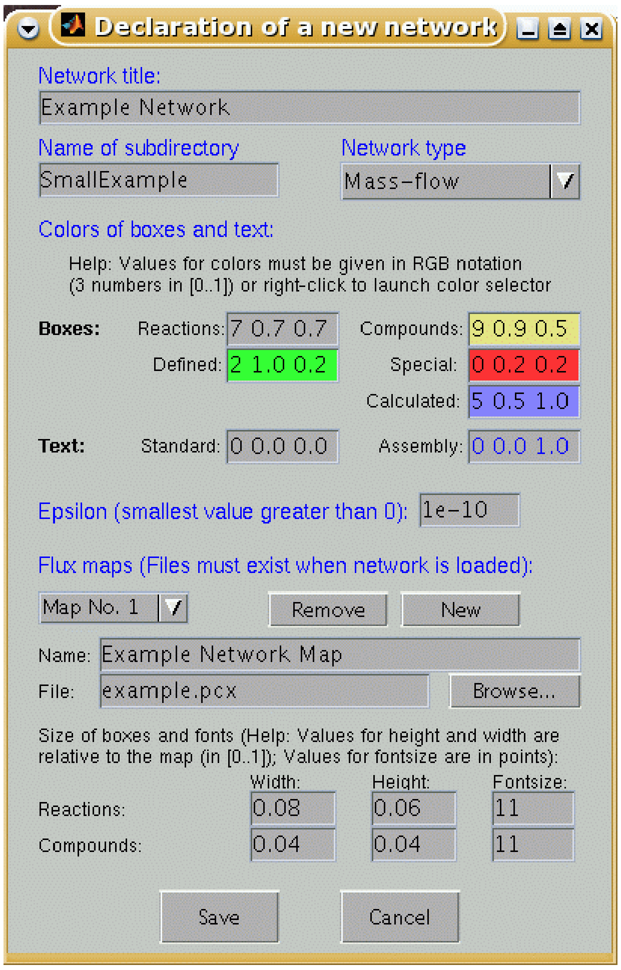
Fig. 2: Example of a stoichiometric (mass-flow) network project in CellNetAnalyzer.

2. Creating and managing network projects

After starting CNA in MATLAB (ensure that the main directory of CNA is the working directory) the Project Manager (Fig. 3) appears showing all registered network projects. “(M)” indicates mass-flow (metabolic, stoichiometric) networks, “(S)” signal-flow (regulatory) networks. Existing network projects can be selected and then loaded, normally with GUI (button “Start”) but sometimes also without GUI (‘Load w/o GUI’; see also chapter 7). In the only entry of the menu bar (‘Project’) you may alternatively “edit” general properties or “clone” (duplicate) or “remove” the selected project. Finally, you may initialize a new project. Initialization of a new project requires that the user defines some general properties of this project using the mask shown in Figure 4. First, the name of the network project has to be defined (as it will appear later in the list of networks in the Project Manager). Then, the name of the directory must be defined under which the network files will be located. One should use a subdirectory of the CNA main directory. In this case it is sufficient and recommended to declare only the name of the directory (e.g. ‘MyNetwork’) instead of defining the absolute path as ‘home/CNA/MyNetwork’. If the directory does not exist, CNA will create a new one.
Next, the user has to define the network type (“Mass-flow” or “Signal-flow”). Users not familiar with the fundamental differences between these two types of networks should first read sections 3 and 4 before assigning the respective network type. Note that the network type of a project cannot be changed once the project has been created.



(S) Signaling Toy Network
(S) T cell model (Large)
(S) T cell model (Small)
(M) Stoich. Network Example
(M) Escherichia coli
(M) Escherichia coli (Small)
(M) Chloroplast Carbon Metabolism
(M) Human Red Blood Cell (Small)
(M) Human Red Blood Cell (Large)
(M) Purple Nonsulfur Bacteria



Network title: Example Network

Name of subdirectory: SmallExample

Network type: Mass-flow

Colors of boxes and text:

Help: Values for colors must be given in RGB notation (3 numbers in [0..1]) or right-click to launch color selector

Boxes: Reactions: 7 0.7 0.7 Compounds: 9 0.9 0.5
Defined: 2 1.0 0.2 Special: 0 0.2 0.2
Calculated: 5 0.5 1.0

Text: Standard: 0 0.0 0.0 Assembly: 0 0.0 1.0

Epsilon (smallest value greater than 0): 1e-10

Flux maps (Files must exist when network is loaded):

Map No. 1

Name: Example Network Map

File: example.pcx

Size of boxes and fonts (Help: Values for height and width are relative to the map (in [0..1]); Values for fontsize are in points):

	Width:	Height:	Fontsize:
Reactions:	0.08	0.06	11
Compounds:	0.04	0.04	11

Fig. 3 (left): Project Manager
Fig. 4 (right): Declaring the properties of a new network project

Next, the background colors and text colors for the text boxes can be defined. Colors have to be given in RGB notation (three numbers between 0 and 1 defining the proportions of the colors red, green and blue). The text box color / text color resulting from the entered RGB values is directly exemplified in the respective box (see Figure 4). When initializing a new network, CNA inserts default color values which can then be changed by the user. A convenient way for selecting a color is to right-click on the respective text box which launches a color selector dialog.
The different colors have the following meaning (cf. colors of boxes and text in Fig. 2):

Text boxes:
- *Reactions* (default: gray): Standard background color for text boxes associated with elements of type “reaction/interaction” (see sections 3.1 and 4.1).
- *Compounds (*default: yellow): Standard background color for text boxes associated with compounds (“biomass components” in mass-flow networks and “species” in signal-flow networks; see sections 3.1, 3.2 and 4.1, 4.2)

After computations, box colors are often changed to indicate the context of each network element (cf. Fig. 2):

- *Calculated* (default: blue): Often used as text box color for network elements, for which a value has been calculated in a certain computation.
- *Defined* (default: green): Background color for boxes of those network elements whose value was defined before a computation.
- *Special* (default: red): Background color for marking elements which had a particular meaning in this computation (for example: balanceable rates in a redundant system; cf. section 3.2).

Text:
- *Standard* (default: black): Color of the text in all boxes, except:
- *Assembly* (default: blue, in Fig. 2 light red): Color of text used for displaying assembly rates. This is only of relevance in mass-flow networks.

Another project parameter that has to be defined by the user is *epsilon*, the smallest number larger than zero. *Epsilon* is used in various calculations for distinguishing “zero” from “non-zero”.

Furthermore, as explained in section 1, the associated networks maps of the current project (i.e. the graphics files) must be declared. An arbitrary number of network maps can be involved; at least one must be declared. By clicking on “New” in Fig. 4, a new interactive map can be declared, by “Remove” the currently selected map is removed from the network project (the graphics file itself is not deleted). Using the pull-down menu each map can be selected for editing its properties:
- *Name of the network map*: The map name appearing in the top of the window displaying the network map.
- *Name of the file*: Represents the graphic file storing the map (see section 1). Possible graphic formats are: pcx, bmp, tif, MATLAB-fig (jpg is possible but should be avoided). The button “Browse...” helps to search for the file, recommended is to have the file in the respective network directory (you then need only to specify the file name (without giving an explicit path)). The file will be loaded when the network project is started and each map serves as background for an interactive network map in a separate window.

- *Size of boxes and fonts*: For each network map the *relative* width and height of the text boxes (separately for reactions and compounds) and the text box font size must be defined. These values are specific for each map. For example, the value 0.04 for height (or width) means that the box has a height (width) of 4% of the complete height (width) of the map. Maybe, you must try a little bit to find appropriate values.

NOTE again: If the user does not have any map visualizing his network (e.g. when importing a large network from a database) and if he does not want to create one, he still has two possibilities: (i) one may choose a "dummy" picture, for example the one provided with CNA (dummy.pcx). Then, obviously, the user will not have a meaningful visualization of his network. However, as the internal network structure of a project will be independent from the network map, he can nevertheless compose a network and analyze this network with the provided menu functions. The text boxes can be placed anywhere or even outside of the map. Results, for example a computed set of elementary modes or a calculated flux distribution can usually be exported in ASCII files. (ii) A second possibility is to load the network without a GUI (either using the 'Start w/o GUI' button in the project manager or just from command line; see chapter 7). One may then analyze the network with the provided API functions of CNA (chapter 7).

If all properties and parameters for the new network project have been defined the initialization of the network project is finalized by clicking on "Save". CNA creates then all files (most of them are empty) needed for the new network project. By clicking on "Cancel" you can quit without saving.

As mentioned above, having a network project selected in the Project Manager and clicking on "Start" loads the selected network. If the network project has just been initialized (as explained above) only the network maps will be displayed because a network model does not exist yet. In general, in all maps the menu item "CellNetAnalyzer" is available from which you can start various actions including the Network Composer for building up the network structure (section 3.2 and 4.2). Note that you can load several copies of the same network project (e.g. for comparing different scenarios). But be careful: when saving a modified network structure in one copy, it will not yet be visible in the other copy in the memory. A network session can be finished by closing any of the network maps. CNA will ask you whether you want to save your changes and whether the network variable is to be cleared from MATLAB's workspace. One may avoid clearing the variable in case one wants to apply API functions to the network project afterwards (without having the GUI loaded).

Using the "Edit" menu entry (under Project) in the Project Manager all properties described above can be changed for the selected network (same mask as in Fig. 4), except the network type. With "Remove" one may delete the selected network project from the network list (the files of the network project are *not* deleted and can thus be re-installed later on).

With "Clone ..." in the Project Manager menu, it is possible to create a duplicate of an existing project in a separate directory. First the user is asked to specify the target directory for the project clone. If this directory does not exist it is created automatically. Then one can enter the name under which the clone will be registered in the project manager. If this is left empty, the project will be cloned but not registered. During cloning, the files describing the network and the network maps are copied. However, the network maps are copied only if they are located in the project directory of the source project. This allows sharing of network maps between projects. When cloning a project, possibly existing files in the target directory will not be overwritten and any attempt to do so will result in an error.

If one wants to register an existing network project (e.g. a copy of a network project received from a colleague), one should click on “New” in the project manager, insert the name of the project and the name of the (existing) project directory and then click on “Save”. You are then asked whether the “app_para.m – file” in the network directory should be overwritten. Click here on “No”. After this, the project is “visible” for *CellNetAnalyzer*.

NOTE: If a network map of an existing network project has been removed from the project, the user will possibly be requested to reassign the old map numbers to the new network maps (after loading the network project). This is necessary for positioning the text boxes in the correct network map.

3. Constructing and analyzing mass-flow (stoichiometric, metabolic) networks

3.1 How mass-flow networks are represented in CellNetAnalyzer

In CNA, a metabolic or mass-flow network is composed of elements of 4 distinct types (table 1):

Element Type	**Metabolite (Species)** network node	**Reaction** (bio)chemical conversion	**Biomass Constituent** substantial macro-molecule such as protein or RNA	**Assembly Route** efflux of a metabolite for synthesis of a biomass constituent
Text Box	none	reaction rate ([mmol/(gDW*h)])	relative biomass concentration ([g/gDW])	rate of metabolite consumption ([mmol/(gDW*h)])
Attributes	• *full name* • *identifier* • *external-flag* • *notes/comments* (External metabolites are not considered to be in pseudo-steady state and therefore not balanced by eq. (1))	• *identifier* • *reaction equation* • *default rate* • *rate minimum* • *rate maximum* • *coefficient* in linear objective function • *variance* of measurements • *text box parameters* • *notes/comments*	• *full name* • *identifier* • *default concentration* • *cumulative synthesis equation* • *text box parameters*	• *identifier* • *metabolite* • *biomass constituent* • *text box parameters*

Table 1: Element types (and their attributes) in mass-flow networks

- **Metabolites (species, storage)**: Metabolites are the network nodes. Usually, in metabolic network models, metabolites represent small molecules like substrates, intermediary products, precursors and building blocks. The basic assumption of Metabolite Balancing techniques is that the concentration of these metabolites is in steady state (i.e. the sum of all fluxes producing or consuming a certain metabolite is zero):

  $$\boldsymbol{0} = \mathbf{N}\,\boldsymbol{r} \qquad (1)$$

  (**N**: stoichiometric matrix (m x q) with m metabolites (rows) and q reactions (columns); $\boldsymbol{r}$: vector (q x 1) of reaction rates, $\boldsymbol{0}$: null vector (m x 1))

  In CNA, a metabolite can also be considered to be "external", i.e. it has not to fulfill eq. (1) and is therefore not included in the stoichiometric matrix if calculations are performed. Note that species are the only network elements that have no associated text box in interactive maps. (The reason is that almost all results are related to reactions).
  In Fig. 2, for example, the capitals (A-D) are the (internal) metabolites and R1 and R7 represent uptake or excretion "reactions" of metabolites.

- **Reactions**: These elements correspond to biochemical reactions, each having a certain stoichiometry. Reactions are the (hyper-)edges in the network connecting the metabolites. The reaction *rates* of biochemical conversions (with unit [mmol/(g dryWeight * h)]) are the central focus of many computations relying on Metabolite Balancing. Each reaction is referenced by a text box which can be arranged by the user on its associated pathway in the network map. (Fig. 2: each of the 8 reactions (R1-R7, µ) has its own text box). Of course, an element of type *reaction* can also be a sequence of more than one biochemical reaction (cumulative stoichiometry).

A special “reaction” is biomass synthesis, which in CNA is always referenced with the identifier *mue* (its rate is the growth rate; unit: 1/h). In contrast to all other reactions, its stoichiometry is not constant because this depends on the defined biomass composition (see *macromolecules*). Therefore, the stoichiometry of *mue* is determined before every computation on the basis of the currently defined biomass composition. In Fig. 2, the text box for *mue* (referencing the growth rate) is located in the network map next to the symbol μ.

Essentially, each (non-empty) mass-flow project comprises a certain number of reactions and metabolites. The following two network elements are optional. They facilitate the consideration of a flexible composition of the biomass and thus a flexible stoichiometry of the biomass synthesis reaction *mue*. In general, if biomass synthesis is considered in a mass-flow network, the user may either define a single biomass synthesis reaction with a fixed stoichiometry (in this case, do not name it *mue*). However, if he wants to investigate e.g. the consequences of changing the biomass composition on certain flux distributions, CNA provides a convenient way to that:

- **Biomass constituents (macromolecules)**: Usually, this element type represents substantial components of the biomass like protein, DNA, RNA, lipids.... etc. In CNA, every biomass constituent (abbreviated as BC) has its own stoichiometry that defines which metabolites are consumed for its synthesis (where the user has to know, for example, the amino acid composition of proteins and so on). Additionally, every BC is referenced by its own text box as it was introduced for the reactions. Therein, the relative concentration of the biomass component can be declared by the user [g/(g dryWeight)]. As explained above, these values are used for calculating the stoichiometry of the biomass synthesis (reaction) *mue*. In Fig. 2, two biomass compositions were defined (BC1 and BC2; their associated text boxes are brightly red). The assumed stoichiometry for their synthesis is as follows:

Biomass component 1:	BC1[g]	=	2 [mmol] A + 1 [mmol] C
Biomass component 2:	BC2[g]	=	1 [mmol] C + 3 [mmol] D

  There are some applications (as often given in the literature) where the user doesn’t have or doesn’t know the biomass composition but the overall demand of precursors for synthesizing the biomass. In such cases, the user can easily define only one BC (the biomass “itself”) and then define the overall efflux of precursors (energy, reduction equivalents, ...) into the biomass. Clearly, the relative concentration of the only BC must then be set to 1. Of course, alternatively he may also define a single reaction comprising the stoichiometry of precursors needed for building the complete biomass. Then, a BC needs not to be declared. However, in this case, assembly routes (below) can also not be defined and *mue* should not be used as reaction identifier.

- **Assembly routes (assembly rates)**: Assembly route is an auxiliary element type and can be used for displaying the efflux of *one* metabolite into biosynthesis of *one* BC. For example, the amino acid serine is used for synthesis of lipids and for synthesis of proteins. For an explicit output of both fluxes in the interactive network map the element type *assembly route* can be assigned. The flux (i.e. the rate) along an assembly route from metabolite M into biomass component B is the product of the stoichiometric coefficient of M in the synthesis reaction of B with the concentration of B and the growth rate (the rate of the biosynthesis “reaction” *mue*). The product of the first two values is implicitly (cumulative with all effluxes of M into all BCs) contained in the biomass synthesis “reaction” *mue*. It is not mandatory to output all possible assembly routes (doing this for ATP would be very tedious, for example). In Fig. 2 one can see four assembly routes (dashed arrows) with red numbers displaying the fluxes along these routes. Bear in mind: an assembly rate is not the same as a reaction rate in this context, because only the latter one is used for the balancing equations of the metabolites.

Each element type has a set of attributes which must be defined if a new element (an instance) of this type is declared (Table 1):

Attributes of *metabolites*:

1) **identifier** of this substance in CNA (e.g.: G6P) without blanks
2) **the full name** of the metabolite (e.g.: glucose-6-phosphate; no blanks allowed!)
3) **flag "external":** if this flag is set, then the metabolite is considered to be external, i.e. the metabolite is considered to be in excess and therefore not balanced via eq. (1). This flag also allows to "remove" a metabolite temporarily from the network without removing it explicitly from the reaction equations
4) **notes/comments:** for documentation purposes. Can contain any strings including line breaks.

Attributes of *reactions*:

1) **identifier** of this reaction without any blank
2) **symbolic reaction equation**: the metabolite identifiers can be used for compiling the reaction equation. The equal sign "=" denotes the reaction arrow. Identifiers, coefficients and equal sign must be separated by at least one blank. The stoichiometry can be defined in the usual way. An example (where NAD and ADP are not modeled explicitly):

   G3P = PG + ATP + NADH

   If a metabolite (e.g. Pyruvate (identifier Pyr)) is taken up or excreted the equation reads:

   = Pyr   or   Pyr =
3) **default value:** this is the default value for the reaction rate (example: a reaction or pathway that is considered only in special cases could be assigned a default value of 0). When CNA is started all default rates are set in the associated text boxes. Of course, during a session, the rate values can be edited in the text boxes whenever the user wants to. Keep in mind that reaction rates have always the unit [mmol/(g dryWeight * h)], except the rate of *mue* ([1/h]). The default value "#" denotes that no default rate for this reaction is defined.
4) **minimum:** is the minimum of the reaction rate, e.g. 0 if the reaction is irreversible. This value is needed for consistency checks, optimization and calculation of elementary modes. If this value is unknown use a high (negative) value (e.g. –100) or even Infinity ("–Inf"). Note that the minimum value must be smaller or equal to 0.
5) **maximum:** is the maximum of the reaction rate. If unknown set a large value (e.g. 100) or even Infinity (value "Inf"). Note: The maximum value must be greater than zero. Thus, a reaction can not be defined in such a way that it can be driven only in reverse direction (e.g. minimum < 0 and maximum <= 0).
6) **Coefficient for objective function:** is the (default) coefficient of the linear objective function for optimization procedures (note: the objective function minimizes). Usually, most coefficients are zero, only some reactions have a coefficient unequal to zero. For maximizing the growth rate set this coefficient to –1 for *mue*.
7) **Variance**: variance of measuremnts of this reaction rate is given to be used in metabolic flux analyses. Whether the given value is a relative or absolute variance can be defined prior the analysis of redundant systems (section 3.3: „Procedures for redundant systems ...").

Each reaction has an associated text box which must be arranged on one of the available network maps. Four parameters define (i) the position of the text box, (ii) the map number and (iii) whether the text box is editable or not. Size and color of the text box are basic properties of the complete network project and can be defined by editing the general project properties (see section 2).

8) **x-position** of the associated text box within the network map
9) **y-position** of the associated text box within the network map
10) **number of the network map** in which the text box has to be arranged
11) **text box style**: is “editable” or “non-editable”. The latter one could be useful if one rate depends only on a certain other rate and should therefore not be simultaneously editable by the user. However, calculated values will be displayed also for non-editable text. For instance, anabolic fluxes depend mostly only on biomass composition and growth rate. Because their rates cannot be measured directly it makes sense to define these fluxes as “non-editable”.
12) **notes/comments:** for documentation purposes. Can contain any strings including line breaks.

Usually, each reaction text box should be placed on the associated pathway (reaction arrow) in the network map. The position is related to the pixel position (in pixel-oriented graphics) or to the relative position (in MATLAB-graphics, *.fig). How the x/y-coordinates can be easily obtained is explained in section 3.2.

Note: If you don’t want to create/display a text box for a certain reaction you can define large negative coordinates for this text box. MATLAB places the text box then virtually outside the map.

Note: For the biosynthesis “reaction” the user should assign the identifier *mue* (as explained above). A reaction equation for *mue* is not directly formulated, instead, the cumulative stoichiometry for biomass synthesis is calculated *automatically* (prior a computation) by multiplying the current concentration of the biomass constituents and the synthesis equation for the biomass constituents. Therefore it is not necessary to define a reaction equation for *mue*.
As mentioned above, it is not mandatory to declare the reaction *mue*, one may also use another reaction identifier (e.g. *biosyn*) and then assign a cumulative stoichiometry for biomass synthesis. However, in this case, the convenient definition of the biomass composition (and the resulting stoichiometry for biomass synthesis) by defining biomass constituents cannot be used.

<u>Attributes of *biomass constituents*:</u>

1) **identifier:** for an unique identification of this macromolecule (no blanks)
2) **full name** of the macromolecule (no blanks)
3) **default value** for the relative concentration in the biomass ([g/g dry weight]). This value is shown by default in the associated text box when the network application is started.
4) **synthesis equation**: contains the stoichiometries of the consumed metabolites for synthesis of 1 gram of this biomass component (unit: [mmol Metabolite / g biomass component]). With a small deviation (see section 3.2) the equation is formulated by using the metabolite identifiers in the same way as for reaction equations.

To specify the associated text box (containing the value of the relative concentration) for a biomass constituent the same four parameters as explained for the reactions are needed:

5) **x-position** of the associated text box in the network map
6) **y-position** of the associated text box in the network map
7) **number of the network map** in which the text box has to be arranged
8) **text box style** (“editable” or “non-editable”)

<u>Attributes of *assembly routes*</u>

1) **identifier of the metabolite** (consumed for synthesis of the biomass constituent (2))
2) **identifier for the biomass constituent** (in whose synthesis the metabolite (1) is consumed)
3) **x-position,** where the rate has to be displayed in the network map
4) **y-position** where the rate has to be displayed in the network map
5) **number of the network map** in which the rate has to be displayed

Metabolite (1) must be contained in the synthesis equation of the biomass component (2).
Each time a computation has been finished, CNA checks, whether the rate of *mue* (the growth rate) is known. If so, the efflux of metabolite (1) in biomass component (2) is displayed on position (3)/(4) in the network map (5). These text boxes always have the style “non-editable” because any user input would not make sense. As mentioned above, every (possible) assembly route is facultative.

So far we have learned how a metabolic or stoichiometric network is modeled in CNA. The external (chapter 5) and internal storage (chapter 7) of a mass flow network is explained at another place. The next paragraph explains how we can compose a network in CNA.

3.2 Composing and editing the structure of stoichiometric networks

A convenient way to construct a network in CNA is the *Network composer* that can be opened from the “CellNetAnalyzer”-menu (Figure 5). The identifiers of the currently defined instances of the four element types are shown as alphabetically ordered lists. The properties of a selected element can be displayed and edited (“Edit/Info”) in a mask. For the declaration a new network element click on the “New” button of the desired element type. A mask comes up where all attributes of the new element can be defined as described in section 3.1. Figs. 6-9 show the masks of the element types; elements of the network shown in Fig. 1 were used as examples. The masks for existing reactions and biomass constituents can also be opened by right-clicking on the associated text boxes on the network map. It is not possible to open multiple editing masks at the same time. Therefore if you open a new editing mask any previously opened mask will be deleted and possible changes discarded.

Using the description of the element properties given in section 3.1 it should be no problem to fill out the masks. Here is some additional advice:

Metabolites: The participation of a certain metabolite in reactions, synthesis of biomass constituents and assembly routes can be seen by clicking on the corresponding buttons in the metabolite mask (appearing after clicking on “Edit/Info”, Fig. 6).

Reactions: A reaction equation must be defined for each reaction except for the biomass

synthesis *mue* (cf. section 3.1). Again, it is important that the rate maximum value is always greater than zero. Thus, a reaction should not be defined in such a way that it can be driven only in reverse direction (e.g. when minimum < 0 and maximum <= 0).
The x/y-position of the associated text box for the reaction rate can be conveniently defined by clicking on the button "Get x/y-Pos" (Fig. 7) and positioning subsequently the mouse in the window of that network map where the text box is to be arranged. A crosshair appears which can be used to click on the place which is reserved for the text box. The x/y-coordinates (as pixel for pixel-based graphics or as relative coordinates for MATLAB-figures) are automatically inserted in the text boxes "X-Position" and "Y-Position". After closing the mask ("OK"), the text box will be placed on the defined coordinates.
See below how you can change the coordinates for all text boxes conveniently in one step (button: "Move Text Boxes ..." in the Network Composer window).

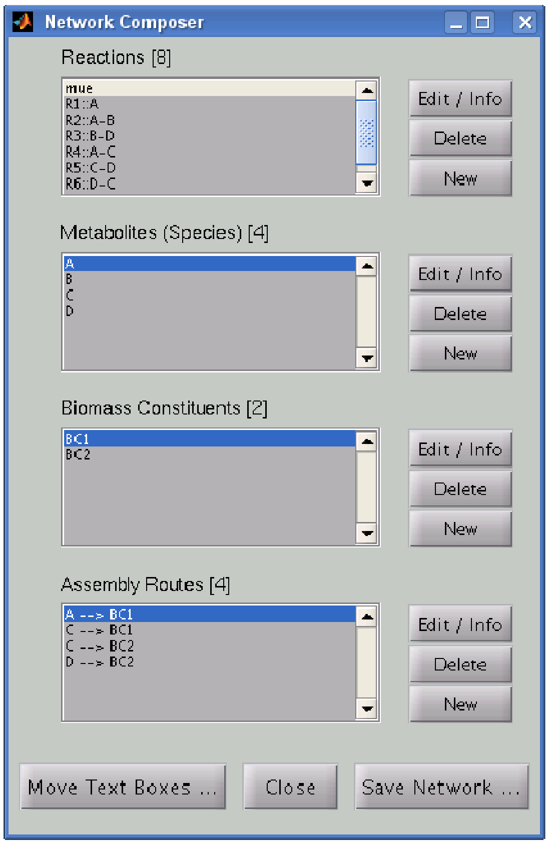
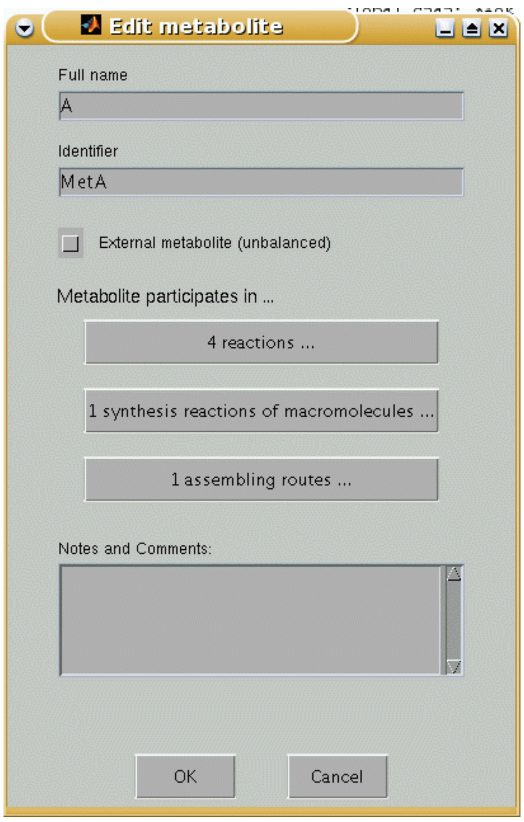


Fig. 5: Network Composer for editing the network structure (here: for example network Fig. 2)

Biomass constituents: A (cumulative) synthesis equation must be defined. The syntax is as follows: <coeff1> <MetID1> + <coeff2> MetID2 + ... (without equal sign). For determining the x/y-position of the associated text box (displaying the relative concentration) use the button "Get x/y-Pos" as described above for the reactions.

Assembly rates: For a selected biomass constituent (right pull-down menu in Fig. 9), CNA configures automatically the (left) metabolite pull-down menu, which lists all metabolites that are contained in the synthesis equation of this biomass constituent. Select one to determine a new assembly route. The text position for displaying the assembly rate is specified in the same way as for reaction text boxes.


Edit metabolite

Full name: A

Identifier: MetA

☐ External metabolite (unbalanced)

Metabolite participates in ...

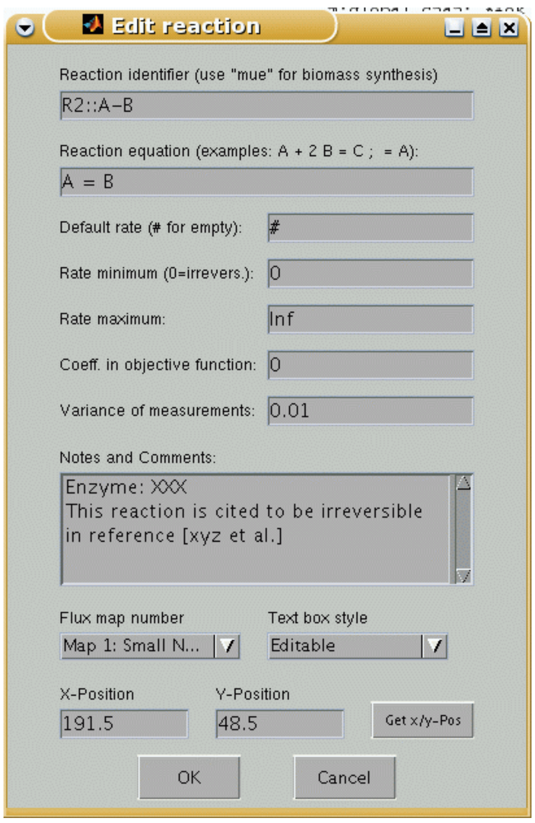
4 reactions ...

1 synthesis reactions of macromolecules ...

1 assembling routes ...

Notes and Comments:

OK | Cancel


Edit reaction

Reaction identifier (use "mue" for biomass synthesis): R2::A–B

Reaction equation (examples: A + 2 B = C ; = A): A = B

Default rate (# for empty): #

Rate minimum (0=irrevers.): 0

Rate maximum: Inf

Coeff. in objective function: 0

Variance of measurements: 0.01

Notes and Comments:

```
Enzyme: XXX
This reaction is cited to be irreversible
in reference [xyz et al.]
```

Flux map number: Map 1: Small N...

Text box style: Editable

X-Position: 191.5

Y-Position: 48.5

Get x/y-Pos

OK | Cancel

Fig. 6 (left): Mask for editing/declaring metabolites.
Fig. 7 (right): Mask for editing/declaring reactions.

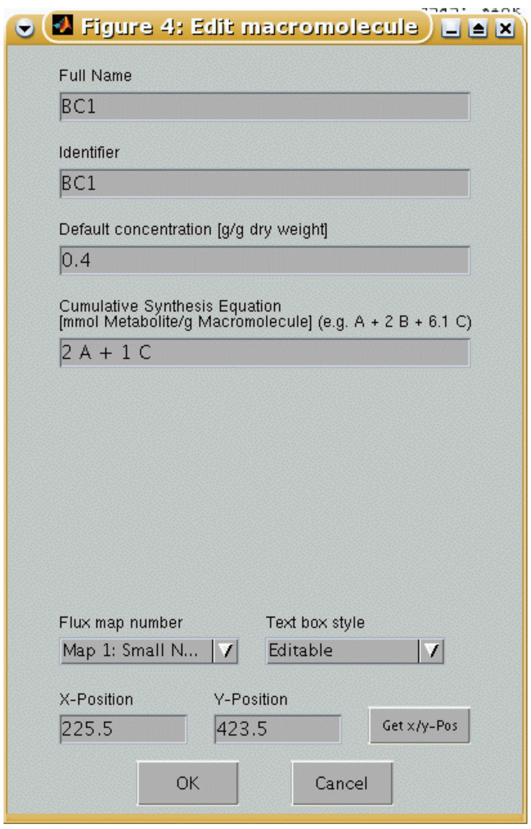


Figure 4: Edit macromolecule

Full Name: BC1

Identifier: BC1

Default concentration [g/g dry weight]: 0.4

Cumulative Synthesis Equation
[mmol Metabolite/g Macromolecule] (e.g. A + 2 B + 6.1 C): 2 A + 1 C

Flux map number: Map 1: Small N...

Text box style: Editable

X-Position: 225.5

Y-Position: 423.5

Get x/y-Pos

OK | Cancel

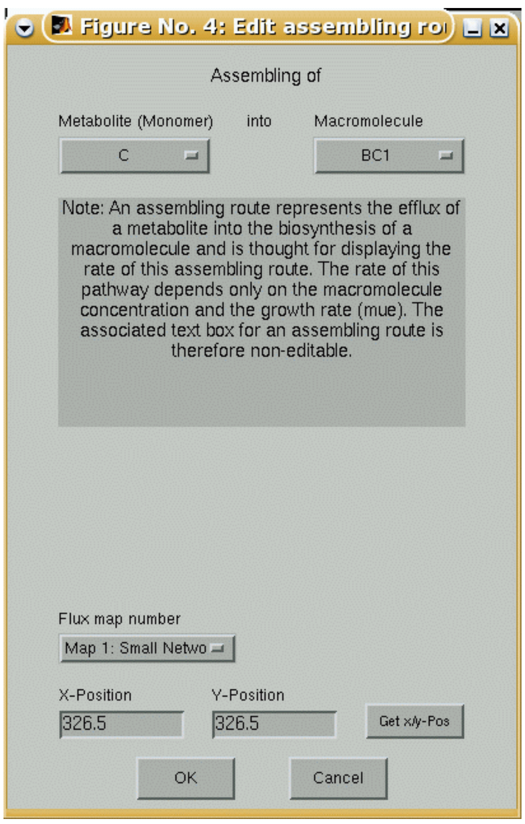


Figure No. 4: Edit assembling ro

Assembling of

Metabolite (Monomer): C into Macromolecule: BC1

Note: An assembling route represents the efflux of a metabolite into the biosynthesis of a macromolecule and is thought for displaying the rate of this assembling route. The rate of this pathway depends only on the macromolecule concentration and the growth rate (mue). The associated text box for an assembling route is therefore non-editable.

Flux map number: Map 1: Small Netwo

X-Position: 326.5

Y-Position: 326.5

Get x/y-Pos

OK | Cancel

Fig. 8 (left): Mask for editing/declaring biomass constituents
Fig. 9 (right): Mask for editing/declaring assembly routes.

If all network elements have been defined or edited the network structure can be finally saved ("Save"). In case you press "Close", the modifications are not lost but remain valid only for the current session with this network project. It is also possible to save the network later during the same session by opening the network composer again and then pressing "Save".

Convenient positioning of all text boxes

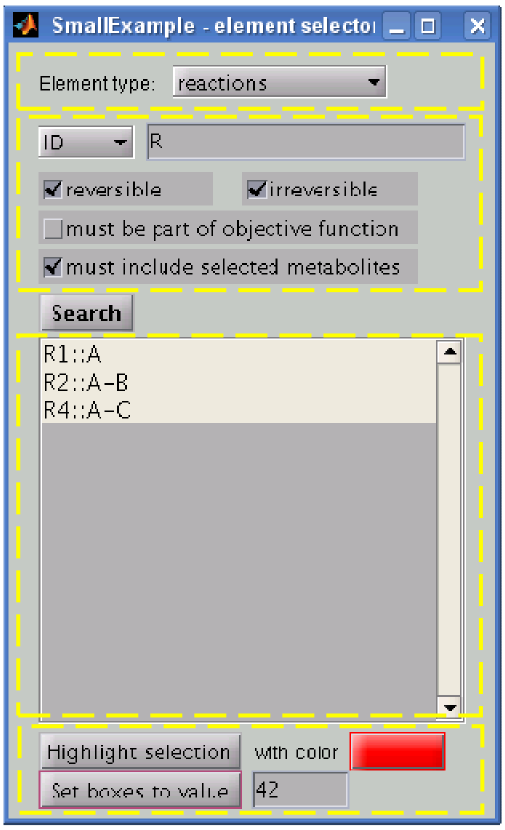
The button *"Move Text Boxes ..."* in the network composer window allows convenient positioning and movement of text boxes directly on the network maps. Clicking on this button opens a small window with two buttons: "Accept changes" and "Discard changes". The user will need this window when the rearrangement of the text boxes has been finished. Each text box (of an arbitrary map) can now be selected and then be moved (drag and drop) to the desired place on the map. It is important to use the LEFT mouse button since by the right button the selected text box will be copied. For selecting a text box of type reaction or biomass constituent one needs to click in the five pixel border of the box which becomes then highlighted. Selecting the text of assembly routes is a bit easier as you can click in the center of the text. When the user has finished the rearrangements he can accept or discard the changes by choosing the respective button in the above mentioned window. After this, the text boxes are "fixed" again and can not be moved anymore. Don't forget to save the network.

When using this function the user should consider the following points:

1. If a text box has been selected, it can also be resized. However, this will have no effect in CNA. After finishing the rearrangement (Accept/Discard changes), the text boxes will be resized to the default size (that can be defined in the project properties, see section 2).
2. If a text box is moved from one network map into another one (or into any other MATLAB window) the text box will be COPIED (the original text box will remain in the original map). However, the new text box won't have any connection to *CellNet-Analyzer* and will thus not be used. If the user wants to move a text box from one network map into another he must change the network map number of the respective network element (see Figs. 7-9).
3. The toolbar of MATLAB should not be used as long as the text boxes are in "movable" state. The CNA menu is also in an inactive state as long as the rearrangement has not been finished. Finally, the user should also avoid changing the zoom factor (via CNA's zoom tools) while the text boxes are movable.

Searching for elements in the network (element selector)

Calling "Element selector" from the CNA menu launches a new window for searching elements in the network, highlighting their text boxes and setting values. Although the appearance of this window varies with the network type and the element type you are currently searching it consists of four main parts which are described in Fig. 10. The search string is taken to be a case-insensitive regular expression which follows MATLAB's regexpi syntax. This means that when you are searching for non-alphanumerical characters (most importantly .*[]()|) they have to be quoted with a preceding backslash (e.g. use \. to search for a dot). The regular expression functionality is in particular useful for making an or-combination of two search strings using the pipe symbol (string1|string2). Leading and trailing white-space of the search string is ignored, but white-space within the search string is significant for the regular expression. If the search string is empty it is interpreted as matching all elements. For more advanced queries using regular expressions please refer to MATLAB's help.



Element type: reactions

ID: R

[✓] reversible
[✓] irreversible
[ ] must be part of objective function
[✓] must include selected metabolites

R1::A
R2::A-B
R4::A-C

with color 🟥

42

Element type to search, influences the options below.

String search in the selected attribute of the element (can be, depending on the element type, the ID, name or notes); right-click for search history
the other search options can be used to restrict the results

Search results; by default, all results are selected, but the selection can be modified by the user
- if IDs is searched they are displayed here
- if names or notes are searched they are displayed with the associated ID prepended to them

Subsets of the found search results can be (de-)selected.
Double-clicking on an element launches a window for editing the element properties (cf. Figs. 6-8).

Text boxes of selected elements from the search results can be highlighted or set to a specified value

Fig. 10: Searching for those reactions that contain "R" in their ID, are either reversible or irreversible and that include previously selected metabolites (in this case this is metabolite A).

The search results of a metabolites/species search can also be used in a subsequent search: When searching for reactions it is possible to restrict the results to those reactions in which all previously selected metabolites participate (the same can be done with species/interactions). For instance, in Fig. 10 this option is used to only select those reactions that involve metabolite A (which was selected in a previous search).

3.3 Toolbox (menu functions) for analyzing mass-flow networks

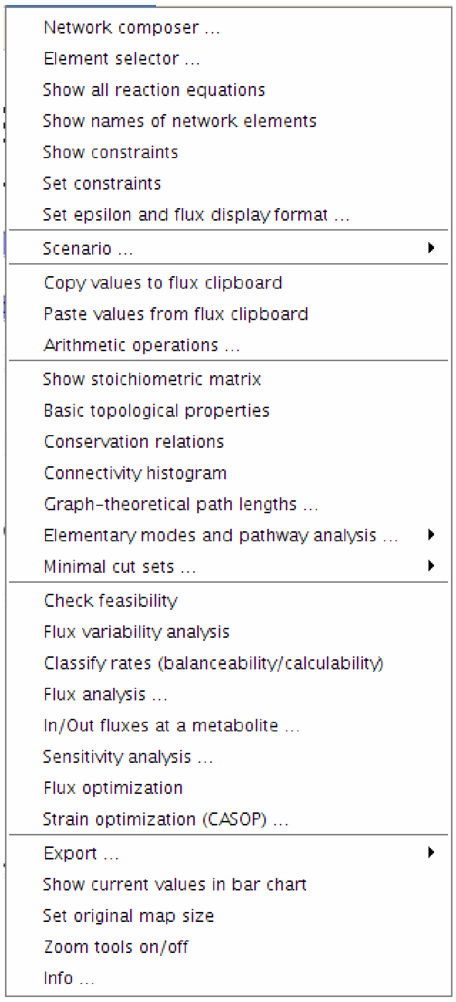
Fig. 11 depicts the menu of *CellNetAnalyzer* which you find in the menu-bar of all interactive network maps after loading a mass-flow network project.

If a menu function has been started it is recommended to pay attention to both the results displayed directly in the text boxes on the maps and to the output in the command window.
The menu is partitioned into several groups of functions (cf. Fig. 11):

- Editing/Viewing network structure, network elements and parameters
- Saving/Setting/Loading values (reaction rates and biomass composition belonging to a

certain scenario)

- Flux clipboard and arithmetic operations
- Analysis of network structure: basic and graph-theoretical properties; conservation relations, pathway analysis; cut sets
- Metabolic flux analysis, flux balance analysis (optimization of fluxes)
- Miscellaneous


Network composer ...
Element selector ...
Show all reaction equations
Show names of network elements
Show constraints
Set constraints
Set epsilon and flux display format ...
Scenario ...
Copy values to flux clipboard
Paste values from flux clipboard
Arithmetic operations ...
Show stoichiometric matrix
Basic topological properties
Conservation relations
Connectivity histogram
Graph-theoretical path lengths ...
Elementary modes and pathway analysis ...
Minimal cut sets ...
Check feasibility
Flux variability analysis
Classify rates (balanceability/calculability)
Flux analysis ...
In/Out fluxes at a metabolite ...
Sensitivity analysis ...
Flux optimization
Strain optimization (CASOP) ...
Export ...
Show current values in bar chart
Set original map size
Zoom tools on/off
Info ...

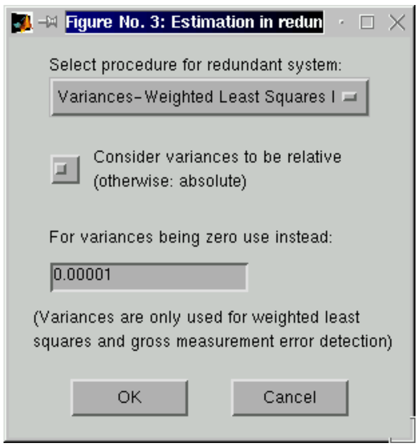


Figure No. 3: Estimation in redun

Select procedure for redundant system:

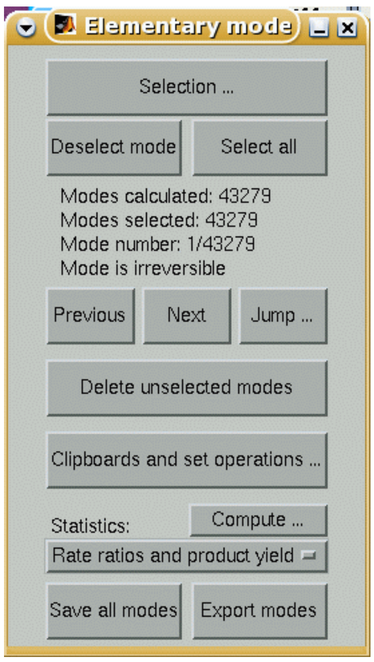
Variances-Weighted Least Squares I

Consider variances to be relative (otherwise: absolute)

For variances being zero use instead:

0.00001

(Variances are only used for weighted least squares and gross measurement error detection)


Elementary mode

Modes calculated: 43279
Modes selected: 43279
Mode number: 1/43279
Mode is irreversible

Statistics:

Rate ratios and product yield

Fig. 11 (left): "CellNetAnalyzer" menu in mass-flow networks
Fig. 12 (center): Window for selecting „Procedures for redundant systems ..."
Fig. 13 (right): Control panel for displaying/analyzing computed elementary modes

In the following all menu functions are described in detail (ordered by menu groups):

3.3.1 Editing/Viewing network structure, network elements and parameters

Network composer ...: This opens the main window for editing and saving network elements and network structure as it was described in detail in sections 3.1. and 3.2.

Element Selector ...: Tool for searching network elements fulfilling a given set of properties. Explained in detail in section 3.2 (Fig. 10).

Show all reactions: Displays all symbolic reaction equations in the command window (including the biomass synthesis equation based on the currently defined biomass composition).

Show names of network elements: In each text box the name of the associated network element (reaction or biomass constituent or assembly route) is shown. This function is especially useful when moving text boxes (“Move Text Boxes ...” in network composer window; section 3.2).
Click on „Reset last values“ for reloading the data set which was contained in the text boxes prior to the last calculation.
The names of reactions and biomass constituents are also shown as tooltip strings when moving the mouse over the respective text box.

The following two functions allow a fast editing of parameters of reactions (coefficient in the objective function, minimum/maximum rate) directly in the associated text box. Sometimes, this is more convenient than editing the values for each reaction separately (in the masks).

Show constraints: Shows three parameters (separated by a blank) for each reaction rate:
1) Coefficient for the reaction for the linear objective function
2) minimal reaction rate (e.g. zero for irreversible reaction)
3) maximal reaction rate

These three values are properties of the element type “reaction” and can also be defined by using the masks (section 3.2). All three parameters are used for the optimization routine and the latter two additionally for checking consistency of calculated reaction rates, for computation of elementary flux modes (reversibility condition) and for checking feasibility. The values can be changed and stored as new values by using the following command.

Set constraints: Takes the currently defined constraints in the text boxes (having the order as described under “Show constraints”) and stores these values as the new constraints for each reaction. It is recommended to load the constraints in the text boxes (by “Show constraints”), then to edit these values and finally to save them by the use of this function.
(Hint: The changed values are only valid for the current session unless the network is saved).

Set epsilon and flux display format ...: The user may change the *epsilon* value (smallest number greater zero) and the display format, i.e. the number of digits displayed after the decimal point in the text boxes. The epsilon value is required for numerical calculations, e.g. for computation of elementary modes. Note that the epsilon will only be changed for the current session, not permanently. To change it also for later network session you must change it by editing general project properties; see chapter 2.

3.3.2 Saving/Setting/Loading scenarios

Scenario ... : is a sub-menu with several functions for saving, setting and loading scenarios. A scenario is the set of known numeric values of reactions (usually the reaction rates) and biomass constituents (usually the biomass proportion) that is displayed in the respective text boxes. This sub-menu contains the following items:

Scenario.../Clear all values: In each text box referencing a reaction or assembly rate the string “#” (meaning “undefined”) is displayed. The values for macromolecule concentrations remain unchanged (because they must have a value prior starting a calculation from the menu).

Scenario.../Reset last scenario: The values defined prior the last calculation are displayed in the text boxes.

Scenario.../Set default scenario: The default values for the reaction rates and the relative concentration of biomass components are displayed in the text boxes. If no default value exists for a certain network element the string “#” is displayed.

Scenario.../Save scenario: Opens a dialog box for saving the currently shown text boxes values (reaction rates, biomass proportions) in a file. By default, the file is saved in the directory of the current network project and should have the file extension “val” (“values”).

Scenario.../Load scenario: For loading a scenario (formerly saved by “Save scenario”). A window opens for selecting a scenario file (by default with extension “val”). The loaded values are displayed in the text boxes and are highlighted in the same way as done by Scenario.../Highlight values (see below).
Attention has to be paid if the network structure has been modified after the scenario was saved. Then, problems may occur if names of network elements (reactions or biomass constituents) have been changed. Network elements which have been added do not cause problems (they will get value “#”, i.e. “not defined”).

Scenario.../Highlight values (on/off): This function colors the reaction text boxes according to the numerical values displayed in them. Text boxes with a zero value are indicated by color “special” and those with non-zero values by color “computed”. Boxes with non-numerical values remain in standard color. This function is useful, for instance, to quickly identify zero fluxes in the network.

Scenario.../Highlight values (heatmap): This function is similar to the previous one but uses the *heatmap* style to highlight the reaction text boxes. The user has to define a threshold value which defines the ‘zero’ or ‘neutral’ value. For all text boxes with a larger value, the text box color becomes green; the saturation of the green hue is determined by the ratio

(text box value)/(max_value_of_all text_boxes – threshold value).

Similarly, for all text boxes having a smaller value than the threshold value, the text box color becomes red; the saturation of the red hue is determined by the ratio:

(text box value)/(threshold value – min_value_of_all text_boxes).

(Boxes with non-numerical values have standard color.) One may use this function, for instance, to display and highlight gene expression data of metabolic genes within a network context.

3.3.3 Flux Clipboard and Arithmetic Operations

Values to flux clipboard: Works like a usual clipboard: The currently defined values in the text boxes of reaction rates are copied to a clipboard (macromolecule concentrations are not copied!). The following two functions can be applied to the values of the flux clipboard.

Paste values from flux clipboard: Writes the values from the flux clipboard into the text boxes.

Arithmetic operations: Useful for arithmetic linkage of two flux distributions, for example for computing the difference between two flux distributions. The four basic calculation operations are available for linking the values currently displayed in the text boxes with the values from the flux clipboard. Alternatively, arithmetic operations can also be performed between a number and a flux distribution (e.g.: normalization of a flux distribution on the substrate uptake rate). The result supersedes the old values in the text boxes.

3.3.4 Analysis of network structure: basic and graph-theoretical properties; pathway analysis

Show stoichiometric matrix: The stoichiometric matrix is displayed graphically for a quick overview on the stoichiometry of the network. The rows of the displayed matrix correspond to the metabolites (names given on the left boundary of the window) and the columns to the reactions (names given on the lower boundary of the window). A red matrix element e_{ij} corresponds to a metabolite i which is consumed in an *irreversible* reaction j. In the same way a green matrix element e_{ij} corresponds to a metabolite i which is produced in a *irreversible* reaction j. A blue matrix element e_{ij} corresponds to a metabolite i which participates in a *reversible* reaction j. A black matrix element e_{ij} indicates that metabolite i is neither consumed nor produced in reaction j. The connectivity number of each metabolite is given on the right boundary of the window (in brackets: number of reactions where the metabolite is consumed[irreversible reaction]/produced[irreversible reaction]/involved[reversible reaction]). The names of reversible reactions are shown in blue. If the network is very large (more than 120 reactions or/and metabolites) the display of this matrix might fail.

Basic topological properties: Calculates global topological properties of the network and displays them in the command window. This function is also useful to detect errors in the network structure after the network has been composed. Note that external metabolites are not considered when determining these properties.

1) Internal metabolites connected to no reaction and reactions without any participating internal metabolite are detected.
2) *Sinks and Sources*: Metabolites which can only be produced or only be consumed (dead-ends) are displayed.
3) *Simple uptake/excretion of one metabolite:* useful to detect transport (uptake/excretion) pseudo-reactions.
4) *Blocked (strictly detailed balanced) reactions*: The rates of such reactions are always zero in steady-state and are thus of no importance for many stochiometric studies (for example, blocked reactions are never involved in any elementary mode). An example: the rate of a reaction in which a dead-end metabolite participates is always strictly detailed balanced (and must have the value zero to maintain the metabolite in steady state). Note that the identification of blocked reactions is done here by null-space analysis.
5) Reactions with *empty stoichiometry* (where none of the internal metabolites participates) are displayed.
6) *Enzyme subsets*: In each flux distribution obeying the steady state condition all reactions of one enzyme subset have always either zero flux or non-zero fluxes with fixed proportions.
7) All groups of *parallel reactions* (reactions having the same stoichiometry (often from isozymes) are displayed. Two reactions with reverse stoichiometry (reverse direction) are also considered to be isozymes if at least one of these reactions is reversible.
8) The linear objective function (required for the “Flux optimization” menu function) – as currently defined – is displayed.
9) *Stoichiometric matrix*: Number of internal metabolites (rows) and reactions (columns) is given. The rank of the stoichiometric matrix is computed. If the rank is lower than the number of (internal) metabolites then conservation relations exist (linearly dependent rows) usually

caused by „conserved moieties“. That means that the balance of (at least) one metabolite is implicitly contained in the balance of the other ones and can therefore be expressed as a linear combination of other balances. For instance, a simultaneous balancing of NAD and NADH usually leads to such a priori redundancy because the balance of NADH is the same as the balance of NAD multiplied with -1. Conservation relations can be eliminated by the removal of one (or more) metabolite(s) contained in the elementary conservation equations. The latter ones can be calculated and displayed by a separate menu function (mentioned below).
Next, *CellNetAnalyzer* computes a compressed version of the current network. This compressed network would have the same basic properties with respect to steady-state flux distributions (e.g. an equivalent set of elementary modes). In the compressed network, for example, all identified strictly detailed balanced reactions would be removed. The dimension of the compressed network tells you how complex your network really is. The number of reactions in the compressed network could be further reduced if parallel reactions (emerging, for example, by isozymes) are considered only once. Note that the network compression procedure is also employed before computing elementary modes.

Conservation relations: This procedure calculates the *elementary* conservation relations (ECR) and displays them in the command window. Conservation relations occur in networks where the rank of the stoichiometric matrix is lower than the number of metabolites (see notes above: menu item "Basic topological ...") and may be seen as redundant balance equations. Calculating the *elementary* conservation relations one obtains all non-decomposable conservation relations similarly to *elementary flux modes* (see reference [8]). The ECRs are useful for detecting conserved moieties but also to find those metabolites, which are involved in conservation relations. The user is requested whether he wants to compute all ECRs or only the non-negative ECRs (which represent conserved moieties, see [8]), and he has also the option to use *CellNetAnalyzer*'s and/or METATOOL's MEX files for computing the ECR's (accelerates the computation). Regarding MEX files see also section 0.4 and under menu item "Elementary modes and pathway analysis ..."

Connectivity histogram: The distribution of the connectivities (connectivity = number of reactions in which a metabolite participates) is displayed in a histogram.

Graph-theoretical path lengths ...: This enables the calculation of *graph-theoretical* (shortest) path lengths between metabolites (plus the average path length and the network diameter). Note that reaction networks are usually *hypergraphs*. Therefore, for applying graph-theoretical methods, this hypergraph must be transformed into a bipartite or substrate graph. Note also that the results from graph-theoretical paths analysis can differ considerably from pathway analysis using elementary modes (see below). For example, a graph-theoretical path from metabolite A to B can exist but, nevertheless, there might be no way to produce B from A. However, graph-theoretical path lengths are – compared with elementary modes – quickly computable and might be seen as a very rough characterization of routes in the network.
Prior to the computation, the user can choose several options in the appearing window:

- *Exclude external metabolites*: If this has been chosen, then connections via external metabolites are not considered, i.e. these nodes are considered as not existing in the network. (This also means that the path lengths between external and other metabolites are not calculated). This will usually increase the paths lengths.
- *Use directed graph*: If this option is selected the reversibility of the reactions is taken into account by using the rate minimum (see "Set constraints" and section 3.1), hence the directionality of the edges is then considered. In such a case, the shortest path

length from A to B is, in general, not the same as that from B to A.

- *Display distance matrix*: Displays the shortest path lengths (distances) between each metabolite pair as a colored matrix. The larger the path lengths is the brighter the matrix cell. A blue color marks metabolite pairs, between which no path exists. In an undirected graph, the distance matrix is symmetric. In directed graphs, this is not necessarily the case. Then, the matrix element e_{AB} represents the shortest path length from A to B. In very large networks, the creation of this graphic representation of the matrix might fail in MATLAB. Use then:
- *Export distance matrix*: This allows for an export of the calculated shortest path lengths into an ASCII file. The user can then further study the path lengths matrix by its own algorithms and tools.
- *Exclude reactions whose rate is given*: All reactions having an arbitrary number (e.g. 0 or 1) in their associated text box are considered as being not existent during the calculation. This enables to hide certain reactions when calculating the path lengths.
- *Compute path lengths only for these metabolites*: This enables the user to select only some metabolites (using the corresponding identifiers) for which then the path lengths to all other metabolites are calculated. If, for example, metabolite ‘A’ is defined in this row, then only the path lengths from A to all others are computed. Several metabolites may be defined here (separated by a blank). If no identifier is given, the shortest path lengths between all metabolite pairs are calculated.

After computation, depending on the chosen options, the distance matrix is shown and/or saved. Besides, the average path lengths, the network diameter (longest shortest path length) and the number of connected components in the graph are displayed in the command window.

Elementary modes and pathway analysis ...: This sub-menu opens the door for pathway analysis: elementary modes or a convex basis (a minimal generating set of the flux cone) can be calculated and analyzed (the latter allows also for computing extreme pathways). Recently, this functionality has been extended to treat also inhomogeneous (equality/inequality) constraints on reaction rates – in this case, CNA may compute elementary/extreme rays and points of the resulting flux polyhedron (seealso [23]).
Basically, two different scenarios can be considered: (i) In the *homogeneous* case, the solution space defined by the steady state assumption + reversibility constraints forms a polyhedral (flux) cone. Elementary modes correspond to particular (elementary) rays with an irreducible set of non-zero elements and include all extreme rays of the flux cone (but possibly more). In contrast, the convex basis (minimal generating set) of a flux cone is the lineality space plus the set of *extreme* rays of that cone. The convex basis is not necessarily unique. (ii) In the *inhomogeneous* case, inhomogeneous constraints (e.g. fixing a reaction rate to a non-zero value or introducing upper and/or lower boundaries for the rates) form a flux polyhedron. This function will then compute either the elementary vectors (with irreducible number of non-zero entires) of the flux polyhedron (including all extreme rays and extreme points) OR, again, only a minimal set of unbounded (lineality space + extreme rays) and bounded (extreme points) generators spanning the resulting flux polyhedron (Minkowsi sum). Note that the zero point will not be delivered separately, even if it is an extreme point of the solution space. Most applications focus on elementary modes in the homogeneous setting (and we will therefore alsways speak about (elementary) “modes”) but there are also applications for inhomogeneous specifications. For more information see reference [23].

The sub-menu has three entries:

Load modes ...: Allows one to load modes saved in earlier sessions can. Select a proper file (CNA saves modes in MATLAB files with extensions “mat”). Warning: modes should be newly calculated if they have been saved before the network structure has been modified. Otherwise, an error message could appear.
In case that a set of modes has already been loaded or computed, the user is asked whether the set of modes to be loaded has to be combined with current set of modes (or whether the current set of modes has to be discarded). This enables to compute certain sets of modes independently and to merge them later.

Show modes ...: elementary modes which are still in the memory (loaded or calculated earlier in the current session) will be displayed again.

Compute modes ...: A dialog box opens for computing elementary modes / convex basis (=minimal generating sets of the flux cone). By the following checkboxes the user can specify several options with respect to the computation:

- *Consider constraints given in text boxes*: This option allows one to *exclude* reactions or/and to *enforce* the involvement of certain reactions. In addition, inhomogeneous constraints on reaction rates can be specified (see constraints (3)-(5) below). If this checkbox is activated, the following specifications in the reaction text boxes can be made will be considered prior computation:
  (1) All reactions having a zero in their associated text box are considered as reactions which *must not* be contained in the elementary flux modes to be computed. That means that these reactions are always (temporarily) excluded and, hence, will have rate 0 in all resulting elementary modes.
  (2) In contrast, each reaction which has a numerical value unequal to zero in its corresponding text box is enforced to be involved in the modes to be computed. [Note that enforcement of reactions is not possible for computing convex bases (but exclusion is); see below.] Thus, this option can be used to compute only a subset of all elementary modes of the network. The computation time may considerably be reduced by excluding reactions. Mostly, enforcing reactions does also have a positive effect on running time, but sometimes it may even take longer than computing the complete set of modes (without enforcing reactions) although this is a superset of those modes which involve all enforced reactions.
  (3) One may enforce equalities on reaction rates. For example, if a reaction rate should equal 3 then write “= 3” (with space between “=” and “3) in the corresponding text box. Note the difference with constraints of type (2) – in the latter case, a non–zero value (without “=”) only demands that the reaction must be active in the modes to be computed, but no specific value is enforced.
  (4) One may set lower boundaries for reaction rates, e.g. by writing “>= 5” (again with space between “>=” and “5”) in the text box.
  (5) One may set upper boundaries for reaction rates, e.g. by writing “<= 7” (again with space between “<=” and “7”) in the text box.
  Note that constraints (4) and (5) must explicitely be set in the text boxes, the standard min/max reaction rate values (defined e.g. via network composer) will not be considered here, except for marking irreversible reactions (minimum reaction rate is non-negative). One may also combine constraints (4) and (5) (but no others) within one text box, e.g. “>= 1 <= 4” will constrain the respective reaction rate within the

range [1,4]. Consistency should be ensured (e.g. “<= 1 >= 4” makes no sense and “<= -2” is not allowed if was rate minimum attribute of the reaction is larger than -2). Furthermore, one may set constraints (1)-(5) for arbitrary many reaction rates. One should not use constraints (3) for a value of “0”, as this can be better achieved by constraint (1).

Any constraints of type (3), (4) or (5) render the problem to be inhomogeneous and the solution space to be a (flux) polyhedron (which can generally be described by a set of bounded (extreme points) and unbounded (extreme rays and lineality space) generators (Minkowski-Weyl theorem) in contrast to the unbounded (flux) cone requiring only extreme rays for its inner description. In the inhomogenous case, this function will compute all extreme points and rays as well as all elementary vectors of this flux polyhedron (convex basis flag is off; see below) OR (if the convex basis flag is on; see below) a minimal set of bounded (extreme points) and unbounded generators (extreme rays + lineality space) Note that, in the inhomogenous case, the returned vectors will be an unsorted mixture of bounded and unbounded generators which is (currently) not (yet) indicated when displaying the mode in the maps (see below). If one needs these flags, one may use the API function CNAcomputeEFM instead (see chapter 7).

If the option (checkbox) *“Consider constraints given in text boxes”* is not selected then all elementary modes (irrespective of constraints given in the text boxes) will be calculated.

- *Check reversibility:* If this option is selected the reversibility of the reactions is taken in account by using the rate minimum (see section 3.1 and 3.2). If this option is set to “off” there will be no restrictions on the elementary modes with respect to feasibility (will increase the number of modes considerably). Useful for some particular applications (in smaller networks).

- *Only a convex basis:* If this check-box is selected CNA computes only a “convex basis” (also: “minimal set of generating vectors”) which is, in the (homogeneous) case of a pointed flux cone, a proper or non-proper subset of the elementary modes (the latter being elementary vectors in the sense that they contain an irreducible number of non-zero elements). If the flux cone is not pointed, the the convex basis will also contain a linelity space and is then non-unique. A convex basis suffices to represent each feasible flux distribution in the network (lying in the so-called “flux cone”) by a nonnegative linear combination of the convex basis vectors, however, for many applications one needs the full set of elementary modes. If some reactions are configured to be reversible, then the convex basis can be non-unique (whereas the set of elementary modes is). In large networks with many reversible reactions the number of elementary modes can be several-fold greater than the number of vectors in the convex basis. Note that the “convex basis” option can also be used to compute Extreme Pathways – provided that the network has been configured properly (see also ref. [4]).

  Analogously, in case of inhomogeneous constraints (see above) only a minimal generating set of bounded (points) and unbounded (lineality space + extreme rays) generators will be calculated if this checkbox is selected – otherwise, all elementary vectors (including extreme points and rays) will be computed). If you want to compute the elementary modes in the system only, do not select this checkbox and do not set any constraints of type (3)-(5).

- *Consider isozymes only once*: Selecting this option means that from each isozyme group (group of reactions having the same stoichiometry) only one representative is considered for calculating the modes. The other reactions of the isozyme group will then not occur in any of the determined modes but their “equivalent modes” can easily be identified by the modes where the representative reaction occurs (one only has to substitute the representative by another reaction from the isozyme group). Two reactions with reverse stoichiometry (reverse direction) are also considered to be isozymes if at least one of these reactions is reversible. In this case, the reversible reaction is retained. For example, in Fig.1, R5 and R6 would be one isozyme group, R6 would be retained as representative and R5 would be dropped. In this example, the only “true” information lost is the mode (2-cycle) built up by R5 and R6.
  This option is especially useful in larger (reconstructed) networks. Therein, many isozymes may occur, which then increase drastically the number of modes. Many of these modes would then carry redundant (equivalent) information due to the parallel isozyme reactions.

- *Calculation method*: Currently four different calculation methods are supported:
  1. METATOOL + CNA MEX files (default): When this method is selected, *CellNetAnalyzer* tries to use METATOOLs algorithm for computing elementary modes which proved to be faster than *CellNetAnalyzer*’s routines (one reason is that the main computation is done *completely* externally). The required MEX-file (for WINDOWS: DLL-file) is distributed together with CNA (cf. sect. 0.4). In fact, CNA and METATOOL MEX files are used together with this option.
  2. *EFM Tool*: Currently the most efficient method for large networks [15]. However, calculation of the convex basis is not supported. In addition, EFM Tool can fail when too many reactions are enforced (c.f. bugs section on /www.csb.ethz.ch/tools/efmtool). When choosing EFM Tool, an additional checkbox “*Use rational numbers*” appears. If this is selected EFM Tool uses arbitrary length rational numbers during calculation. This yields exact results (at least on the EFM Tool side) but increases memory usage and computation time. For additional information see README.txt in CellNetAnalyzer/code/ext/efmtool/.
  3. *CNA MEX files:* CNA’s original algorithm for computing elementary modes also uses MEX files accelerating the computation of elementary modes. If the META-TOOL module does not work on your system (for reasons described above), you may try to use CNA’s MEX version (cf. sect. 0.4). If no executable CNA MEX file is found the pure MATLAB version will be used which, however, is slow in large networks.
  4. *CNA functions:* Provided as fall-back option in case the MEX-files do not work.

  In general, which version (EFM Tool, METATOOL MEX, CNA MEX or CNA MATLAB) is actually used is displayed in the command window after starting the computation.

When the computation has been started the progress of the calculation can be seen in the command window. The algorithm starts with preprocessing steps mapping the original network to a smaller one with the same final number of modes (e.g. by combining all reactions from an enzyme subset into one overall reaction). Calculating elementary modes / comvex basis needs

information on the reversibility of each reaction which is taken from the rate minimum (if rate minimum < 0 then the reaction is considered to be reversible, otherwise not). For calculating elementary modes it is important that the rate maximum value is always greater than zero. Thus, a reaction should not be defined in such a way that it can be driven only in reverse direction (minimum < 0 and maximum <= 0). If METATOOL is used, METATOOL will try to convert the stoichiometric matrix into an integer representation. If this is successful (if you have not too many real numbers in your matrix with many decimal places) METATOOL will employ integer arithmetic (and thus produce exact intermediate results).
The overall calculation time depends on the network structure, on the selected options described above and on your hardware. It can take a few seconds or several days (or even longer ...). Besides, it can occur that the algorithm stops with an error message: “Out of memory”. In this case you have not enough RAM or your network is simply too large to be treated with elementary-modes analysis.

If the calculation has been finished (or if a set of flux modes has been loaded) the first (flux) mode is shown in the network map and a control panel comes up (Fig. 13). Again, it depends on the chosen constraints and options (convex basis) whether the shown mode is really an elementary mode or, e.g., an extreme point. In the following we will always refer to the standard case of “elementary modes” but most of the applications and tools described below are also meaningful in conjunction with e.g. extreme points in flux polyhedra.

All participating reactions of a flux mode (having a rate unequal to zero in this mode) get the color “calculated” and reactions having a zero get the color “standard” (see section 2). A reaction which does not occur in any mode (for instance when it was excluded by using the option “Exclude reactions ……” as described above) is designated by the color of type “defined”. Each elementary mode is normalized in such a way that the smallest rate has an absolute value of 1, except if biomass synthesis occurs (*mue* unequal to zero). In the latter case the flux mode is normalized to a growth rate of 0.1. The number of the current mode as well as its reversibility is displayed in the elementary mode control panel (Fig. 12).

The flux mode control panel provides the following functions that are useful for changing the currently displayed flux mode, for selecting a certain subset of all modes, for calculation of statistical properties and for saving the computed set of modes:

- *Selection*: This tool allows the selection of a certain subset of all flux modes. First, a panel opens where the following 8 specifications for selecting a subset can be made:
    - first row: reactions which *must not* be contained in any mode (i.e. those reactions must have a rate of zero in the selected modes)
    - second row: reactions which *must* be contained in all modes (i.e. those reactions must have a rate unequal to zero in the selected modes)
    - third row: reactions which may only proceed in backward direction (i.e. those reactions *must* have a rate being negative *or zero* in the selected modes)
    - fourth row: reactions which may only proceed in forward direction (i.e. those reactions *must* have a rate being positive *or zero* in the selected modes)
    - metabolites which must not participate in any mode
    - metabolites which must participate in all modes
    - minimal number of participating reactions
    - maximal number of participating reactions

For each row no, one or several reaction or metabolite name(s) can be defined, respectively. If nothing is specified the complete set of modes is selected (see also button "*Select all*" below). After pressing the button OK the user is asked whether the selection should be performed on the current selection (if a subset is chosen currently) or on the complete set of elementary flux modes. In the first case the user specifies a subset of the current subset thereby refining the selection to incorporate the new constraints.
For example, if in the first row *mue* is entered the user will get all flux modes without growth. Another example: First row: uptake of oxygen; second row: *mue* - all flux modes which allow anaerobic growth will be selected.
Except button"*Save all modes*, all functions described in the following operate only on the current selection of modes.
Important: an additional option to select modes (namely with respect to their product yields) is provided when computing "*(Product) Yields and rate ratios*" (see below).

- *Deselect mode*: Deselects the currently displayed mode. Thus, a certain flux mode can be deselected separately without using the *Selection* tool.

- *Select all*: Selects the complete set of elementary modes.

- *Next mode / Previous mode*: Switches to the display of the next or previous mode of the set of currently selected modes.

- *Jump ...*: Here, a certain mode can be reached by its number. This mode becomes the currently displayed mode. (Note: the mode number must be contained in the set of currently selected modes).

- *Delete unselected modes*: Removes all unselected modes irreversibly from the memory (in contrast to the *Selection* tool). This tool could be useful if some modes are not of interest and can be dropped.

- *Clipboard and set operations*: Useful for obtaining subsets of elementary modes which can not be generated by the *Selection* tool alone. The user can copy the current selection to clipboard 1 or 2 as well as restore a subset from clipboard 1 or 2 to the current selection. Furthermore an additional window can be opened by "*Set operations*" where set operations (union, difference, intersection) for combining the subsets of the current selection, clipboard 1 and clipboard 2 are provided. After performing a set operation the obtained subset becomes the current selection. Example (cf. Fig. 1): Flux modes for growth on substrate A and D have been calculated. For selecting all flux modes, where substrate A or D is taken up (reaction *R1::A* or *R7::D* have rate unequal to zero) select first all modes, where *R1::A* is unequal to zero (Selection tool) and copy this selection (*Clipboard* tool) to clipboard 1. Then select all modes, where *R7::D* is unequal to zero (*Selection* tool) and combine then (*Clipboard* tool) the current selection with clipboard 1 by set operation "union". The selection tool would only allow one to get a subset of modes in which *R1::A* and *R7::D* participate but not those where only one of these both is unequal to zero.

- *Statistics*: A number of useful statistical features of *the current selection* of modes may be determined by selecting the respective feature from the pull-down menu (Fig. 12) and clicking on button "Compute":

- *(Product) Yields and rate ratios*: Searches for the flux mode(s) with the maximal ratio P/S between a reaction rate P (e.g. “product excretion”) and a reaction rate S (e.g. “substrate uptake”). The names of the reactions P and S have to be entered, whereby for both more than one reaction name can be specified separated by blanks (then the maximal ratio (P1+P2+P3...)/(S1+S2+S3...) will be searched). Only those flux modes of the current selection are considered where the sum S1+S2+... of the substrate uptake reaction rates is positive and where the sum P1+P2+... of product rates is not negative. The found optimal flux mode becomes the current flux mode and is displayed. If there are two or more flux modes with the same maximal ratio then the first of these is displayed and the numbers of the others are shown in the command window. The flux mode with the maximal P/S is the optimal flux distribution with respect to product yield (see reference [4]). Thus, this method is an alternative to the optimization procedure (described below). An example: P=*mue*, S=*Glucose_uptake* determines the maximal growth yield on substrate glucose. Besides calculating the maximal rate ratio, a figure comes up displaying the distribution of this rate ratio over all selected elementary modes (again: only modes where at least one of the S-reactions participates are considered).
  Optionally, in addition to the calculations above, modes can be selected whose yield is greater than a definable minimum yield and smaller than a defineable maximum yield (these boundaries can be entered in the same window where the names of P and S rates must be specified). According to these specifications, all those modes of the *current* selection which are in the valid range of yields will be selected (modes with negative P rates or/and zero or negative S rates become unselected). The control panel for displaying the elementary modes will be updated when the selection has been made. If neither a minimum nor a maximum yield is specified (empty lines) then the current selection of modes remains unaltered (and the first optimal mode will be slected as described above).
- *2D Yield space plot*: This function enables one to compute a two-dimensional yield space plot from the current set of selected modes and is particularly useful for exploring the production capabilities of a metabolic network, e.g. for coupled product and biomass synthesis. Similar as above, for each of the two yields X and Y (X=Px/Sx and Y=Py/Sy) one has to define the names of the Px and Py reaction(s) and of the Sx and Sy reaction(s), respectively. As for the previous function, Px and Py will usually present “product excretion reactions” and Sx and Sy “substrate uptake reactions” and several reaction names can be specified separated by blanks (the yields are then defined as X=(Px1+Px2+Px3...)/(Sx1+Sx2+Sx3...) and Y==(Py1+Py2+Py3...)/(Sy1+Sy2+Sy3...)). Only those flux modes of the current selection are considered where the sum Sx1+Sx2+... ands Sy1+Sy2+... of the substrate uptake reaction rates are positive and where the sum Px1+Px2+... and Py1+Py2+... of the product rates are not negative. After specifying the reaction names, the corresponding 2D yield space plot will appear. A typical example would be to choose Px=“mue” and Sx=”substrate uptake” and Py=”product excretion” and Sy=”substrate uptake”. The resulting yield space will display the network’s capability for coupled product and biomass synthesis.
- *Control-effective fluxes*: The concept of control-effective fluxes (CEFs) was introduced and explained in [2]. It tells you something about the “importance” of each reaction in the current set of modes and involves a weighting procedure for each mode with respect to (user-)defined objectives. It has even been applied to predict transcript ratios [2]. Clicking on this button opens a dialog box where you have to define (the last parameter is optional):
    - *Objective reactions:* Here one defines the names of reactions which are supposed to be objectives of cellular optimization (e.g. growth rate ‘*mue*’ or

ATP production).

    - *Substrate uptake reactions:* Here you can define (arbitrary many) reaction names which represent uptake of ‘sources’ or ‘input’, hence, usually substrate uptake reaction(s).
    - *Reference substrate uptake reactions:* This is similar to the previous parameter but selects a reference ‘source’ or ‘reference input’ in order to calculate the ratios of the CEF.

  The CEFs for each reaction (or CEF ratios when “reference substrate uptake reactions” is defined) are calculated and displayed in the text boxes.
  (Hint: a CEF ratio could alternatively be computed by calculating the CEFs for Substrate uptake reaction(s) A, copying the values to the flux clipboard, then calculating the CEFs for Substrate uptake reaction(s) B and dividing the CEF values of A by the CEF values of B by using the menu function ‘Arithmetic operations ...’).
  - *Reaction participation*: Determines the absolute (displayed in the command window) and relative (in %; displayed in the network maps) frequency of the appearance of each reaction (rate unequal to zero) in the selected modes.
  - *Histogram of pathway length*: Displays a histogram representing the frequency distribution of the number of participating reactions in the elementary modes. Quite useful to assess whether there are many/few small/large pathways.
  - *Structural couplings of reactions*: Determines enzyme subsets [= groups of pathways which operate in the current selection of flux modes always together (if one reaction of an enzyme subset has a rate unequal to zero in a certain flux mode then all other member of this enzyme subset have a rate unequal to zero, too)] and mutually excluding reaction pairs [= two reactions which never occur together (with rate unequal to zero) in any mode]. The results are displayed in the command window.
  - *Minimal cut sets*: Starts minimal cut set calculation, display and analysis. Pressing this button is a short-cut because it is equivalent to starting the menu item “Minimal cut sets ...” from the pull-down menu (for a description see below).
  - *Normalization...*: This feature allows the normalization of the (currently selected) elementary modes to a certain reaction or number. Accordingly, a dialog box opens allowing one to enter
    - a reaction name: All modes involving this reaction are normalized to the rate of this reaction. Modes in which this reaction is not involved remain unchanged. If more than one reaction is defined the modes are normalized to the sum of the defined reactions.
    - AND/OR a number: All rates in each elementary mode are divided by this number. Note, if reaction(s) have been defined in the first row of the dialog box, then only modes involving this (or at least one of these) reaction(s) are normalized as described above and additionally divided by the given number.

  An example: If you want to normalize all growth-related modes (with rate of ‘mue’ > 0) to a growth rate of 0.5 then write in the first row of the “Normalization ...” dialog box ‘mue’ and in the second row the value 2.

- *Save all modes*: All modes (not only the selected subset) including the current values of the relative concentrations of biomass components are saved in a file whose name has to be provided. The extension of this file must be “mat” (MATLAB-format).

- *Export modes*: All SELECTED modes can be exported to an ASCII file: either as list of modes (using the reaction names) or as a matrix (rows: elementary modes, columns:

reactions). In the latter case, the reaction names are given in the first row.

Minimal cut sets ...: Calculation, display and analysis of minimal cut sets (MCS). Functions and actions described in the following are similar to those for elementary modes.

Load ...: Allows one to load cut sets that were saved in earlier sessions. Select a proper file (usually, the modes are saved in MATLAB files with extensions “mat”). Warning: cut sets should be newly calculated if they have been computed and saved before the network structure has been modified. Otherwise, an error message could appear.

Show ...: cut sets which are still in the memory (loaded or calculated earlier in the current session) will be displayed again.

Compute ...: A dialog box opens in which several checkboxes allowing one to specify certain parameters for the computation of minimal cut sets. First, the calculation of a new set of MCSs is *always related to the current set of elementary modes*. This means, that you cannot compute MCSs before you have not calculated or loaded a set of modes. Furthermore, at least one mode must be selected. The currently selected set of modes will be considered as all possible realizations of a certain function (like synthesis of a product) which are to be “destroyed”, i.e. the current selection of modes will be interpreted as the set of “target modes”. (In case all modes are selected, the complete network will become inoperable by the cut sets to be computed). The MCSs calculated are those minimal sets of reactions, whose removal from the network will guarantee that none of the selected modes will “survive”, hence, that these pathways are destroyed and non-functional anymore. The “minimal” property means, that no subset of the MCSs would be (structurally) sufficient for this goal. For example, in Fig. 1, {R1,R6} would be an MCS for the complete network. Each essential reaction is an MCS having a size of one. (For further details see [9].) As a new feature, CNA now also allows the computation of *constrained MCSs* [21]: one can define up to two sets of elementary modes from which a minimum number (to be specified) must not be hit by the MCSs to be computed. In this way, one may compute MCSs that, for instance, disable the synthesis of a certain product while preserving the ability to produce other components (including biomass)[21]. In principle, this constrained set of MCSs would be contained in the complete set of MCSs without the side constraint (and one could select them after the computation) but it can be highly beneficial to consider these constraints already during the algorithm since it can drastically reduce the number of admissable MCSs. The “desired modes” (modes that should not be hit by the MCSs) are specified with the help of the two clipboards available for elementary modes (see above).
Before starting the calculation of MCSs, the user has the several options to specify the MCSs to be computed:

- *Exclude reactions with 0 value*: All reactions having a “0” in their associated text box are considered as reactions which *must not* be contained in the MCSs to be calculated. This option can be used to compute only a subset of all MCSs which do not involve the respective reactions.
- *Store cut sets compact:* Enzyme subsets (ES) in the selected sets of modes (which are sets of reactions which occur always together) can be seen as “’equivalent sets of reactions’ for calculating the MCSs: no MCS will contain simultaneously two reactions of an ES, but each MCS in which a reaction of an ES occurs has “equivalent sets of MCSs” in

which this reaction is substituted for another reaction from this ES. If many (large) ESs occur in the set of modes, the number of MCSs can become quite large with many equivalent sets. The user can therefore choose this option to save only the MCSs of one “representative” for each ES. All other reactions from each ES will, nevertheless, be stored. When an MCS is displayed later on, the equivalent reactions will be shown in the command window. Furthermore, the complete set of MCSs with all equivalent MCSs can easily be extracted later on if desired (see “Expand cut sets” below).

- *Request multifunctional enzymes:* If the network contains reactions which are catalyzed by the same enzyme (e.g. transketolase) then this can be considered properly during the calculation of MCSs (since both reactions will be eliminated simultaneously in mutants unable to synthesize the respective enzyme). If this checkbox is selected then – after starting the calculation procedure – the user will be requested to define (arbitrary many) sets of reactions each related to a particular multifunctional enzyme. Each reaction set of a multifunctional enzyme can be defined by entering the respective names of the reactions in a provided window. After choosing ‘OK’ a new set can be defined. One can exit this loop by letting the input field empty and clicking on ‘OK’.
- *Save at least ... modes from clipboard 1*: As described above, using this option one can impose constraints to the MCSs to be computed. In an initial step, one selects a set of modes to be preserved (they should not be hit by the MCSs) and copies them to the elementary-modes clipboard 1 (see clipboard tools for elementary modes). Then, by selecting this checkbox and by specifying a minimum number of modes to be preserved, CNA will compute only those MCSs that hit all target modes and satisyfy the additional constraint of saving the required minimum number of modes (the “desired modes” being specified by clipboard 1).
- *Save at least ... modes from clipboard 2*: The same tool as the previous one but in combination with clipboard 2. One may also combine both clipboards allowing one to define more complex constraints: one may specify up two sets of “desired modes” (one is clipboard 1, the other in clipboard 2) and for each of these sets a minimum number of modes that must be preserved. A complex example: using this facility we may compute MCSs that will destroy all modes but save (i) at least one mode that will enable biomass synthesis (out of a set of biomass-producing modes specified in clipboard 1) and (ii) at least one mode out of a set of modes (in clipboard 2) that produce synthesis of a product of interest.
- *Late check (during postprocessing):* An algorithmic flag meaningful only in combination with the two previos options: The fulfillment of the side constraints (minimum number of modes to be saved) is normally checked during the algorithm which is especially advantageous if the set of “desired modes” is relatively small. However, if the set of desired modes specified in the clipboard is relatively high (compared to the set of target modes) it might by better to discard non-admissable MCSs in a post-processing step which can be enforced by selecting this checkbox.
- *Maximal cut set size:* Here, the user can specify the maximal size of the MCSs to be calculated (= number of reactions an MCS contains). Especially in large networks, the number of MCSs can grow rapidly. Therefore, by limiting the size of the MCSs, the user may focus only on the subset of smaller MCS, which are usually the most important ones.

When the computation has been started the progress can be seen in the command window. The overall computation time depends on your computer, the selected set of modes, the number of involved reactions and the algorithm used. As in the case of elementary modes it can take a few seconds or several days (or even longer ...).

When the calculation has been finished (or if a file with stored MCSs has been loaded) the first MCS is shown in the network map and a control panel comes up. All reactions which are elements of the current MCS are indicated by color “special” (and value “0”). Reactions not involved get the “standard” color (see section 2). A reaction which does not occur in any MCS (for instance when it was excluded by using the option “Exclude reactions ...” as described above) is designated by the color of type “defined”. In case the MCSs have been stored in compact format (see above), the equivalent reactions of the reactions contained in the currently displayed MCS are shown in the command window.
The number of the current MCS as well as the overall number of MCSs is shown in the cut sets control panel which is analogous to that in Fig. 12. Similar as for elementary modes, the following functions in the cut sets control panel are useful for stepping through the set of MCSs, for selecting a certain subset of MCSs, for assessing statistical features or for saving/exporting the MCSs:

- *Selection*: This tool facilitates the definition of a selection of a certain subset of all MCSs. For this purpose a panel opens where the following six specifications can be made:
  - reactions which *must not* be contained in the MCSs to be selected
  - reactions which *must* be contained in the MCSs
  - metabolites which must not participate in any reaction involved in the MCSs
  - metabolites which must participate in all reactions involved in the MCSs
  - minimal size of the MCSs (min. number of contained reactions)
  - maximal size of the MCSs (max. number of contained reactions)

  For each row no, one or several reaction or metabolite name(s) can be defined, respectively (separated by a blank). If nothing is specified the complete set of MCSs is selected (equivalent to “*Select all*”). After entering these specifications the user is requested whether the selection should be performed on the current selection (when a selection is currently already active) or on the complete set of MCSs. In the first case the user specifies a subset of the current subset of MCSs thereby refining the selection to incorporate the new constraints.

  All functions described as follows operate only on the current selection of MCSs, except *“Save cut sets”* and *“Expand cut sets”*.

- *Deselect set*: Deselects the currently displayed MCS. Thus, a certain MCS can be deselected separately without using the *Selection* tool.

- *Select all*: Selects the complete set of MCSs.

- *Next / Previous*: Switches the display to the next or previous MCS of the currently selected MCS.

- *Jump ...*: Here, the number of an MCS to be displayed can be entered. This MCS becomes the currently displayed MCS. (Note: the MCS number must be contained in the set of currently selected MCSs).

- *Expand cut sets*: If the MCSs have been stored in compact format (see above), one can use this function to extract all MCSs.

- *Statistics*: A number of useful statistical features of the current selection of MCSs can be computed by selecting the respective feature from the pull-down menu (similar as in Fig. 12) and clicking on button “Compute”:

  - *Select non-affected modes*: Clicking on this button all those elementary modes (currently in memory) that are not affected by the current MCS (i.e. all modes remaining operable after removing the MCS from the network) will be selected. This enables the user to check which network functions are not affected by the current MCS. Of course, this function makes only sense if the current sets of MCSs and elementary modes are corresponding, i.e. if the set of MCSs has been calculated on the basis of the current set of modes.
  - *Average cut set size*: Determines and displays for each reaction the average size of all selected MCSs in which it is a member of. This gives you a clue how robust the network (or a function) with respect to a failure of that reaction is. The average MCS size of all selected MCSs and the average *average MCSs size of each reaction* as well as the averaged fragility coefficient (see [9]) are displayed in the command window.
  - *Reaction participation*: Determines the absolute (displayed in the command window) and relative (in %, displayed in the network maps) frequency of the appearance of each reaction in the selected MCSs.
  - *Cut set size histogram*: Displays a histogram representing the frequency distribution of the MCSs sizes. Quite useful to assess the fragility of the system.
  - *Show equivalent reactions*: If the MCSs have been stored in compact format (see above), the sets of equivalent reactions are displayed.
  - *Cut set with lowest side-effects*: As for the feature “*Select non-affected modes*” this functionality computes a relationship between the currently selected sets of elementary modes and MCSs, respectively: from all selected MCSs the one which hits the lowest number of selected modes is determined. This MCS (or several MCSs in case that several minima exist) becomes the new selection. This feature is useful to find an optimal intervention strategy (which hits all undesired modes but spares most of the others).

- *Save cut sets*: All MCSs (not only the selection) will be saved in a file whose name has to be given. The extension of this file must be “mat” (MATLAB-format).

- *Export cut sets*: All SELECTED MCSs can be exported as a (binary) matrix to an ASCII file (columns: reactions, rows: MCSs). The first row contains the names of the reactions.

3.3.5 Flux (balance) analysis

(For mathematical details see references [1,5,6]).

Check feasibility: This menu function checks whether there is at least one (feasible) flux distribution in the network, which is consistent with the defined rate constraints (min/max rate) AND with the currently defined rates given in the text boxes. (Note: if a defined network scenario is redundant and not consistent, it is definitely not feasible. But even if a system is consistent, it may happen, that a system is not feasible due to capacity or/and sign constraints of the reaction rates). If a feasible solution can be found then it will be displayed in the interactive maps. NOTE: This function **needs the Optimization Toolbox of MATLAB** or the **GLPK solver**! When starting this routine, the user may choose from two LP solvers: MATLAB’s linprog function (needs the optimization toolbox of MATLAB) or GLPKMEX. Make sure that at least one of the two LP package is working on your PC. (see section 0.4).

Flux variability analysis: Useful for checking the *feasible* upper and lower boundary of each reaction rate consistent with a given set of predefined fluxes and consistent with the min/max reaction values defined for each reaction. Obviously, the computed upper and lower boundaries will lie within the min/max range defined by the reaction properties. Given values in the reaction text boxes are considered as predefined reaction rates (the macromolecule concentrations are also read; if applicable). Linear optimization is then used to determine for each reaction the feasible lower and upper boundary and the result is displayed in the command window. Additionally, all fluxes that are uniquely determined (upper and lower boundary are identical) are also displayed in the corresponding reaction text boxes in the interactive network map (together with the predefined values). This function can also be used as a substitute for "Flux Analysis ..." (see below) as long as the system is feasible and non-redundant. It even has the advantage that the min/max constraints of the reactions (and thus the reversibility of the reactions) is explicitely taken into account; in contrast to classical flux analysis. It may therefore result in a larger number of uniquely determined rates. However, it takes longer since several linear optimization problems must be solved. Moreover, if the defined scenario is overly stringent and thus not feasible (function 'Check feasibility' can be used to test this) an error message will be shown and one should then try "Flux Analysis ..." to treat redundant systems.
NOTE: This function **needs the Optimization Toolbox of MATLAB** or the **GLPK solver**! When starting this routine, the user may choose from two LP solvers: MATLAB's linprog function (needs the optimization toolbox of MATLAB) or GLPKMEX. Make sure that at least one of the two LP package is working on your PC. (see section 0.4)

Classify rates (balanceability/calculability): The currently specified scenario (characterized by the currently defined and non-defined values of reaction rates) is investigated to classify non-defined rates to be (uniquely) calculable (C) or non-calculable (###) and to classify defined ("measured") rates to be balanceable (B) or non-balanceable (M). The classification of rates also allows for a classification of the complete system: If balanceable rates occur then a redundant system was defined (otherwise a non-redundant one). Furthermore, if any unknown rate is non-calculable then the system is underdetermined (otherwise: determined).
The classification result is shown in the network maps (using the above introduced symbols #,B,C,M and the respective box colors as defined in section 2) as well as in the command window of MATLAB. This function is also useful for planning experiments: the user can test whether a certain unknown rate is calculable by the use of a certain set of measured rates. For this purpose the user must set the value of all reaction rates which are thought to be measured on an arbitrary value and all other rates on undefined ("###") and then start this function. (Note: The real values of the defined rates have no influence on the classification! Note also that this classification does not consider reaction reversibilites)

Flux analysis ...: This menu function provides methods for metabolic flux analysis, i.e. for calculating unknown fluxes from measured or known rates (given by the user in the text boxes). Two basic types of scenarios can occur, depending on the redundancy of the defined scenario (see also menu function "Classify rates" (above) and ref. [1]):

Case 1) *The scenario is non-redundant*: In this case, no redundancies in the given rates occur.

(Redundancies induced by conservation relations are an exception. These a-priori redundancies will - independently from the set of given rates - always be consistent and are therefore treated here also as case 1). In a non-redundant scenario, on the basis of the currently defined values, all (uniquely) calculable rates are determined. The results are shown in the network maps whereby the color of boxes associated with a reaction is adapted depending on whether the respective reaction rate was defined or calculated or non-calculable (for colors see section 2). Some further information is displayed in the command window. There, warnings will appear when a reaction rate has left its admissible range (by checking the rate minimum and maximum).
Note that Case 1 can also be treated by “Flux variability analysis” (see above) with the additional advantage that reaction reversibilities are explicitely accounted for.

<u>Case 2)</u> *The scenario is redundant*: In this case, some defined rates possess redundant information. The scenario can now be either inconsistent or consistent (non-redundant scenarios are always consistent). In redundant systems, the *CellNetAnalyzer* allows the calculation (or better estimation) of unknown rates as well as the balancing of so-called balanceable rates (which possess the redundant information). An example for a redundant scenario is shown in Fig. 1 where the growth rate (µ) and the rate for uptake of A (R1) as well as the following conversion to B (R2) and then to D (R3) have been defined. It is obvious that R2 and R3 must have the same value and defining both rates results in a redundant system where R2 and R3 are balanceable. If in one of the following procedures the balanceable rates have been balanced (i.e. the system is made consistent) then these rates get the color of type “special” (e.g. in Fig. 1: red; for defining the colors see section 2). Whether a system is redundant and (if so) which defined rates are the balanceable ones can be easily checked by the function “Classify rates …” as described above. If one has a redundant scenario a window opens where one of the following estimation procedures must be chosen:

- *Simple least squares*: This function determines the least squares solution of the unknown rates by using the pseudo inverse. Only for those unknown rates, for which the solution is unique the values are displayed (with color “calculated”). It is noteworthy that by using this procedure no balancing of the balanceable rates is carried out, i.e. the balanceable rates remain, in general, inconsistent and the solution does then not fully satisfy the steady state condition of the metabolites. This means that residuals occur which are displayed in the command window. Balanceable rates can not be distinguished from the other given rates (by color “special”) because they have not been balanced.

- *Variances-weighted least squares*: This is the standard (and recommended) procedure for handling redundant systems as often described in the literature (e.g. [5,6]): Firstly, the balanceable rates are identified and adapted (by least squares) to obtain a consistent system. This estimation uses the variances of reaction rates (which can easily be defined in the masks for reactions). Then, based on the estimated values, the unknown rates are computed whereby only the (uniquely) calculable rates are displayed (with color “calculable”). The (in general adapted) values of the balanceable rates are also displayed (with color “special”). Given but not balanceable rates are displayed as usual in color “defined” and non-calculable rates remain in color “standard”. Further information about the estimation are displayed in the command window including a consistency check based on the residual *h* which is determined during the estimation of the measurements. If the (variances-weighted) residual *h* is higher than the chi-squares value for confidence interval 0.95, then modeling or measurement errors are expected with a probability of at least 95 %. Additionally, the variances of the calculated (actually: estimated) rates are also displayed in the command window (determined as given in [6]).

- *Gross measurement error detection*: This function can be applied only in redundant systems with at least 2 degrees of redundancies. The goal is the identification of that balanceable rate which causes the largest deviation from the steady state assumption. For this purpose for each balanceable rate a separate estimation based on the above described procedure "*Variances-weighted ...*" is performed where the rate is considered as non-defined. Thus, for each balanceable rate one gets the variances-weighted residual *h* if the rate is not involved (these *h*'s are shown in the command window). The reaction, for which the *h* decreases at most, causes the largest error. This result is also displayed in the command window. No results are displayed in the network maps – every value remains unchanged. (Note: if one would like to calculate the flux distribution with the smallest error one has to set the rate with the largest error to "###" (undefined) and to select subsequently the estimation method "*Variances-weighted …*").

The last three procedures use the variances of (measured) reaction rates. *Variance* is a parameter of each reaction element (see section 3.1 and 3.2). Whether the variances have to be considered as absolute or relative can be defined by the checkbox (if set on "off" means absolute variances). Relative variances are determined by multiplying each given reaction rate with its parameter *variance*. Furthermore, each resulting variance must have a value unequal to zero. This would mean that the relative variance of a given rate whose value is zero (e.g. oxygen uptake under anaerobic conditions) would also be zero. For such cases (or if the defined parameter *variance* of a reaction rate is zero) the "zero-variance" is used which can be entered by the user. This value should be small because zero-rates are mostly well-known and have therefore high certainty.

Again: note that "classical flux analysis" does not consider reaction reversibilites and min/max reaction rates. As long as the system is not redundant, one may also use "Flux variability analysis ..." which has the advantage that it takes min/max reaction rates into account and it may result in a larger number of uniquely determined fluxes. However, this function takes a bit longer and requires the MATLAB optimization toolbox or the GLPK solver.

In/Out fluxes at a metabolite …: Shows a (stacked) bar chart with all influxes and effluxes around a specified metabolite. The fluxes are computed by multiplying the reaction rates (given in the text boxes) with the stoichiometric coefficient of the selected metabolite in these reactions. This feature is useful, for example, to identify the reactions with the highest production / consumption of a particular metabolite. You may apply this function also in the case where only a subset of all reaction rates (producing/consuming the respective metabolite) is given in the textboxes.

Sensitivity analysis...: This function calculates the sensitivities for all currently unknown but calculable rates with respect to a certain currently defined rate. For this purpose, one of the known (entered) rate (here denoted by r_m) can be chosen from a list box. After clicking on the OK-button the linear dependency $\partial r_c / \partial r_m$ of each unknown but calculable rate r_m with respect to the chosen rate r_m is computed and displayed in the text boxes of r_m. Calculated sensitivities get the background color "calculated". Defined rates get the color "defined", except the rate for which the sensitivities have been calculated (gets color "special"). Non-calculable rates remain in color "standard", because their sensitivity is non-calculable, too. In redundant systems the sensitivity is computed with respect to a simple (non-weighted) least squares estimation (see "*Procedures for redundant systems ...*").

Flux optimization (flux balance analysis): This function starts the optimization procedure (only possible in underdetermined systems) where a linear objective function is minimized. The following parameters are involved in this procedure:

- the currently defined rates in text boxes of reactions are considered as constraints
- the valid range (constraints) of the reaction rates (min/max) as well as the coefficient in the linear objective function are basic properties of each reaction and can be defined by using the menu item „Network composer“ or „Show / Set constraints...“.

The user may choose from two LP solvers: MATLAB’s linprog function (needs the optimization toolbox of MATLAB) or GLPKMEX. Make sure that the respective LP package is working on your PC. I suggest taking GLPKMEX, however, due to the licensing conditions you have to install this solver separately (cf. sect. 0.4).
It is useful to have a look at messages in the command window. If you have chosen the MATLAB solver, two different procedures are used for “checking“ the uniqueness of the calculated solution: if these two solutions supply different solutions than it is likely that the calculated solution is not unique (this is mathematically not fully correct and not very elegant, but at least fast and “almost” certain).
An optimization example would be „Maximize growth rate“ for an uptake of one mmol of a certain substrate. For this purpose, set the value 1 in the text box associated with the uptake rate and set the coefficients for the linear objective function for each reaction on 0, except for „mue“ which gets the coefficient -1 (keep in mind: the function minimizes!).

Strain optimization (CASOP): This function provides a computational framework for strain optimization (overproduction of a desired product) based on reaction importance measures derived from weighted elementary modes (the method has been called CASOP = Computational Approach for Strain Optimization aiming at high Productivity; for details see reference [18]). Reaction importances for different proportions (gamma) of the target metabolite at an artificial external metabolite composed of biomass and desired product are calculated (by means of parameter k one can put a high emphasis on yield-optimality (high k, e.g. k=10) or on network capacity (low k, e.g. k=1)). The importances can be plotted for different k’s and gamma’s suggesting knock-out and over-expression candidates (see reference [18]). Similar as the corresponding API function “CNAapplyCASOP” (see chapter 7) the function also returns (as a variable) a reaction rating useful for ranking knock-out and over-expression candidates

A new window is launched and all parameters needed for the CASOP optimization procedure can be specified (Fig. 14; see reference [18]). The involved parameters are:

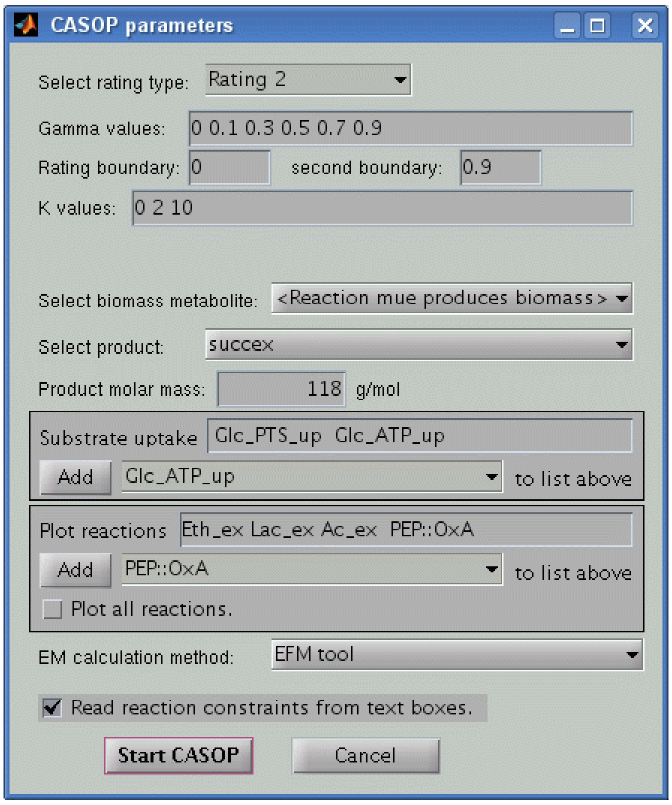
- Rating type: Two different rating types can be chosen.
  If rating 1 is selected, the rating measure is the importance at the specified rating boundary in the corresponding text field.
  If rating 2 is selected, an additional text field for a second boundary occurs and the rating measure is the difference of the importances between the upper and lower bound specified in the corresponding text fields.
- Gamma values: vector with increasing discrete values in the range of [0 1]
- Rating boundaries: at these points the reaction importances are evaluated for the rating measure. Only values that are given in the gamma values are feasible.
- k values: a list of values for the exponent k ($k \geq 0$) (with increasing k a higher emphasis is on yield optimality)

- Biomass metabolite: defines the biomass metabolite (and thus the "cellular objective"). If CNA's standard biomass synthesis reaction *mue* is defined one may use the "default" biomass (composed by the biomass constituents with composition as currently defined in the maps) that is virtually produced by *mue*. Alternatively one may declare an external metabolite (which participates in exactly one producing reaction) as biomass metabolite. In this case, the stoichiometry of the reaction producing this metabolite must be defined in such a way that 1 gram biomass (metabolite) is produced by this reaction.
- Product: name of the (external) target metabolite. Can be selected from a list of appropriate external metabolites (only those are shown that participate in exactly one producing reaction).
- Molar mass: molar mass of the product in unit gram per mol
- Substrate uptake: names of the substrate uptake reactions (can be several reactions; they will be summed up when computing the product (or biomass) yield)
- Plot reactions: to specify names of the reactions whose importances will be plotted (if the checkbox is marked importances of all reactions will be plotted)
- EM calculation method: method / procedure to compute the elementary modes
- Read reaction constraints: if this checkbox is marked, the elementary modes are calculated with respect to the values given in the reaction boxes (for example, an reaction is inactivated if the value in the corresponding box is set to 0; see also menu entry Elementary flux modes and pathway analysis ...).
- Start CASOP: start computation

After computation the importances of the reactions (selected for plotting) will be plotted in separate figures. For further analysis, the reaction importances and rating values for each gamma and k are stored in two new network variable fields:

- cnap.local.ReacWeightMatrix: a cell array where cnap.ReacWeightMatrix{i,j} holds the vector with reaction importances for the i-th gamma-value and the j-th k-value.
- cnap.local.RatingMatrix: a matrix where cnap.RatingMatrix(i,j) holds the rating value for the i-th reaction in cnap.reacID and the j-th k-value.

Again, note that this function can also be started from command line using the CNAapplyCASOP API function (chapter 6).



CASOP parameters

Select rating type: Rating 2

Gamma values: 0 0.1 0.3 0.5 0.7 0.9

Rating boundary: 0 second boundary: 0.9

K values: 0 2 10

Select biomass metabolite: <Reaction mue produces biomass>

Select product: succex

Product molar mass: 118 g/mol

Substrate uptake Glc_PTS_up Glc_ATP_up

Add Glc_ATP_up to list above

Plot reactions Eth_ex Lac_ex Ac_ex PEP::OxA

Add PEP::OxA to list above

☐ Plot all reactions.

EM calculation method: EFM tool

☑ Read reaction constraints from text boxes.

Start CASOP Cancel

Fig. 14: Window for specifying parameters for strain optimization with CASOP. In the example shown, reaction importances for synthesizing Succinate from Glucose (which can be taken up via two uptake reactions) will be computed for 4 reactions. "mue" is chosen as the (standard) reaction producing biomass.

3.3.6 Miscellaneous

Export ... : this sub-menu comprises functions for exporting the stoichiometric network in three different formats:

Export stoichiometric matrix ...: Allows one to export the stoichiometric matrix (together with the identifiers of the reactions and metabolites) for further analysis outside of the *CellNetAnalyzer*. The user can choose whether the matrix is to be exported as ASCII or MATLAB file and whether the external metabolites should be included or not. In ASCII format the matrix is saved as follows:

first row: <reaction names>

second row: reaction reversibilities (0 = reaction irreversible, 1 = reaction reversible)

third row <name of first metabolite> <first row of stoichiometric matrix>

fourth row: <name of second metabolite> <second row of stoichiometric matrix>

....

(m+2)-th row: <name of the m-th metabolite> <m-th row of stoichiometric matrix>

If the matrix is stored in MATLAB format then four variables are saved which can be reloaded in MATLAB later on:

metabolite_names: names of the m metabolites (char array)

reaction_names: names of the q reactions (char array)

reaction_reversibilities: reversibility of each reaction

stoichmat: stoichiometric matrix (m x q) where the m rows are the metabolites in *metabolite_names* and the q columns are the reactions in *reaction_names*

Export in SBML format: Exports the network (basically the species and the reactions including their stoichiometry and reversibility) in SBML (Level 2) format. The user may choose name and location of the file. Note that all internal metabolites (without "external flag") are put into a compartment called "Internal species" and all external metabolites into compartment "External species". Note that the reaction and metabolite ID's are converted to valid SBML ID's when saving. This means all illegal characters are replaced with a '_' and all metabolites/reactions are prefixed with the letter 'S'/'R' respectively followed by a consecutive number.

Export in METATOOL format: Exports the network (basically the species and the reactions including their stoichiometry and reversibility) into a file which can be read by METATOOL. Note that in the output file reactions can occur which have no substrates or no products. This can happen when there are undeclared external metabolites in the network. In such cases, the METATOOL parser may show a warning about incorrect syntax, but this can be ignored because external metabolites don't enter the stoichiometric matrix anyway and therefore don't affect the calculations.

(See also chapter 6: Importing/Exporting from SBML and METATOOL).

Show current values in bar chart: Displays the values currently defined in the reaction text boxes in a bar chart. This is useful for comparing the fluxes of a given scenario and to identify the greatest absolute values. The function is also valuable for evaluating the results of arithmetic operations.

Set original map size: Each network map is resized to its original resolution (pixel-based graphics) or the original size (MATLAB figures). The maps should then appear well-displayed. For this reason, all graphics should not have a resolution larger than the monitor resolution.

Zoom tools on/off: Displays the zoom tools in the network map in which this menu item has been selected (or the zoom tools disappear if they are currently visible). The zoom tools are always mounted in the upper right corner of the map. The scrollbars allow one to scroll the map left/right and up/down, respectively (note: sometimes, the scrollbars are not correctly displayed by MATLAB; you should then click somewhere in the map or just minimize and then maximize the window). The '+' button enables to zoom in. Press the '+' button and then select two points on the map using the cross-hair. The rectangle spanned by these two points will be the new region displayed in the window. The text boxes are also zoomed with the respective factor. By pressing the '-' button one can zoom out (factor of 2). Finally, by pressing the 'R' (reset) button the entire map is displayed again. Note that the assignment of text box positions can also be done in the zoomed state.
Pan (from toolbar, only in MATLAB version 7.3 and higher): Can be used to activate the pan mode of the figure. When the pan mode is switched on you can click and drag the network map around inside the window. This will only have an effect if a part of the map is currently displayed (e.g. after zooming in). Due to limitations of the MATLAB GUI the textboxes are hidden while dragging the maps. Pan can be used in conjunction with zoom but then zoom must be activated first.
Important: Do not use the MATLAB zoom/pan tools when working with CNA.

Info ...: Shows info about *CellNetAnalyzer*.

4. Constructing and analyzing signal-flow (signaling, regulatory) networks

4.1 How signal-flow networks are represented in CellNetAnalyzer

In CNA, the construction and analysis of models of signal-flow networks is done in a similar way as for mass-flow (stoichiometric) networks. However, whereas stoichiometric networks describe the flow of material in a (stoichiometrically) exact manner, signal-flow networks may capture signal (information) and, to a certain extent, also material flows in a more abstract way by describing causal and logical (Boolean) dependencies. This leads to a different network representation. In CNA, a signal-flow network is a Boolean network represented as a signed directed hypergraph. The latter can easily be converted to an interaction graph (see our paper [11]). Before confusing you too much, let’s see how signaling networks can be set-up in CNA in practice.

Signal-flow networks are composed of elements of two different types only, namely of *species* and of *reactions,* here also referred to as *interactions*. The species are the network nodes, equivalent to the metabolites in stoichiometric networks, and interactions are directed hyperarcs in the network similar but not fully equivalent to reactions in stoichiometric models. In signaling networks, each reaction represents one possibility how (i.e. under which conditions) the end node of this reaction can be activated (or produced; see below). The element types *biomass constituents* and *assembly routes* are not available in interaction networks.

Attributes of species and reactions in signal-flow networks are listed in Table 2.

Element Type	**Species** network nodes	**Reaction (Interaction)** interactions (causal relationships, e.g. activations, inhibitions etc.)
Text Box	logical state	signal flow
Attributes	• *full name* • *identifier* • *default value* • *text box parameters* • *comments/notes*	• *identifier* • *reaction (interaction) equation* • *default value* • *time scale* • *flag 1: whether this interaction is to be excluded in logical computations* • *flag 2: whether the logical interaction is treated with incomplete truth table* • *flag 3: whether the interaction is monotone* • *text box parameters* • *comments/notes*

Table 2: Network elements in signal-flow networks and their attributes.

As indicated in Table 2 and in contrast to metabolites in stoichiometric networks, all species have an associated text box usually displaying information about the current or computed (logical or discrete) state of that species. Consequently, as for reactions, a default value can be defined for species as well.

Interactions have some similar and but also different attributes compared to reactions in stoichiometric networks. First, the associated text box usually displays the *signal flow* along this interaction (hyper-)arc (instead of a reaction rate). The reaction or "interaction equation" can be defined symbolically by using the species identifiers. It defines *which* species participate (with positive or negative influence) in this interaction and *which* species can be activated along this interaction. Syntax and semantics of interaction equations are similar as in stoichiometric reaction equations but there are some fundamental differences. In a two-level (binary) network an interaction equation looks as follows

[!]<Spec1> + [!]<Spec2> + [!]<Spec2> ... = <SpecR>

<Spec1>, <Spec2>, ... and <SpecR> are species identifiers. On the right-hand side of the equation only one species identifier (<SpecR>) can appear. Each equation represents one possibility to activate the "target species" SpecR and the "+" sign on the left hand-side denotes a logical AND while the (optional) exclamation marks "!" indicate logical NOT operations. (Note that the meaning of the "+" in CNA differs thus from the symbolic representation of Boolean functions where "+" represents normally an OR operation - but in CNA it is an AND).
An example: the interaction equation

A + !B + C = D

states that D can be activated if A AND C are active AND B NOT. Note that no space between the NOT operator „!" and B is allowed whereas space is mandatory between species identifiers and „+" and „=". Note that all the AND activation rules (i.e. all reaction equations) by which a certain species (here D) can be activated are implicitly OR-connected. Thus, if one would define additionally to the interaction above a second interaction such as

A + !C = D

the total Boolean function for D resulting from these two interactions would read: D is active if (A AND NOT(B) AND C) OR (A AND NOT(C)). Thus, all the interaction equations pointing in one and the same species define implicitly the so-called (minimal) disjunctive normal form (DNF) of the Boolean function describing how the species (here D) can be activated.

Metabolic (stoichiometric) reactions can be represented in this framework up to a certain (qualitative) degree. For example, A + B = C might represent a metabolic reaction where A and B are required (are converted) to produce C. However, stoichiometric coefficients (except "1") cannot be captured because coefficients in interaction equations have a different meaning as explained in the folowing: In binary logical networks, only on / off (0 and 1) states are considered for each species. CNA also allows the construction of multi-valued logical networks in which a species may have arbitrary many discrete states. In the interaction equation, this is indicated by coefficients in front of the species identifiers (separated with blanks from the identifiers):

[Coeff1] [!]<Spec1> + [Coeff2] [!]<Spec2> + [Coeff3] [!]<Spec2> ... = [Coeff3] <SpecR>

For example, the equation

2 A + !C + 2 !G = 2 F

expresses: If A is has at least level 2 AND C is NOT active (level is 0) AND G is NOT at level 2 (or higher) then F will reach level 2. As indicated by “at least” and “not higher”, CNA assumes monotone behavior. That means, in the example above, F gets level 2 if C<1 (i.e. C=0), G<2 A>=2. If you want to have an interaction to be non-monotone, i.e. to be active only if the species states obey exactly the coefficients, you may deactivate flag 3 (Table 2). So if the above example is non-monotone, then F gets level 2 if and only if A=2, C=0, and G<>2 (<> denotes “unequal”). Hence, if A=3, C=0, G=1, then F gets level 2, only if the interaction is monotone. On the other hand, if A=2, C=0, G=3 then F gets level 2 only in the non-monotone case (because of G=3).

As shown in this example, the level (or coefficient) „1“ needs not to be written explicitly (as done here for !C). Therefore, the general interaction equation as given above for binary networks, is just the special case where each species in the equation is associated with a coefficient of “1”.
Note that the exclamation mark is not allowed on the right-hand side and that every species may occur only once in each equation.

Regarding this general principle of constructing multi-level networks there is one important rule: If a species is activated by several different interactions, always the highest possible activation level will be set. For example, having the following two interaction equations

2 A = 3 G
!B + 2 C = 2 G

and assuming that the left-hand side of both equations is fulfilled (A is at level 2, B is zero and C is 2), then G will be assigned level 3 (the highest one). The signal flow along the first interaction will be “3” and along the second one “2”.

Furthermore, input and output arcs may be defined, such as
= A (if the signal flow along this interaction is set on “1”, then A will be activated)
B = (if B has state “1” then this interaction will have an (output) signal flow of “1”)

As already mentioned above, the logical network representation as used herein can be transformed into an interaction graph (the hyperarcs are split into arcs and all species entering an interaction equation with NOT (“!”) are the starting points of a negative (inhibiting) arc in the graph model). However, in certain case one knows that a species, let’s say A, has a positive influence on B but not in such a strong way that it can activate B alone. Then, A would not appear in the logical model. To indicate, that there is nevertheless an influence, one can declare an interaction

A = B

and set the *flag* (1) that this interaction is to be *excluded in logical computations* (see Fig. 17).

In realistic networks, one is sometimes not sure whether the influence of some species on another species has to be combined with a logical AND or a logical OR. An example: assume A activates C and B activates C but you don’t know whether both are required for (full) activation of C. Here one might define an interaction

A + B = C

and set another (the second) *flag*, indicating that this interaction has to be treated with an *incomplete truth table* (see Fig. 17). This means that C becomes zero if A=B=0 and becomes active if A=B=1. These are the identical entries for the AND and OR truth tables. For the other two possible cases (A=1/B=0 and vice versa), the activation level remains undefined. Note that the interaction

A + !B = C ,

where this flag has been set, has a defined value for C when A=1/B=0 (C=1) or A=0/B=1 (C=0) and undefined otherwise.

Finally, another parameter of interactions is the *time scale*. Sometimes, one knows that a certain interaction becomes active significantly later than others (e.g. due to the involvement of gene expression). Then it makes sense to assign a larger time scale value (e.g. “1”) to this particular interaction, whereas all others get a time scale of 0. In principle, arbitrary time scale values can be used; a lower value indicates that an interaction is earlier available. This parameter is used for setting quickly time scale scenarios (see section 4.3).

Fig. 15 depicts the interactive network map of a simple signal-flow network in CNA. Notice that the default colour (grey) and the size of the reaction text boxes have been configured differently compared to species (yellow text box; see section 2) ensuring that species and reactions can easily be distinguished. Connections of species entering an interaction with a negative (NOT) operation were indicated in the network map by a red colour in the map. Hence, the interaction equation of arc 7 is !D = A and for 2&3 it reads !I1 + I2 = E.

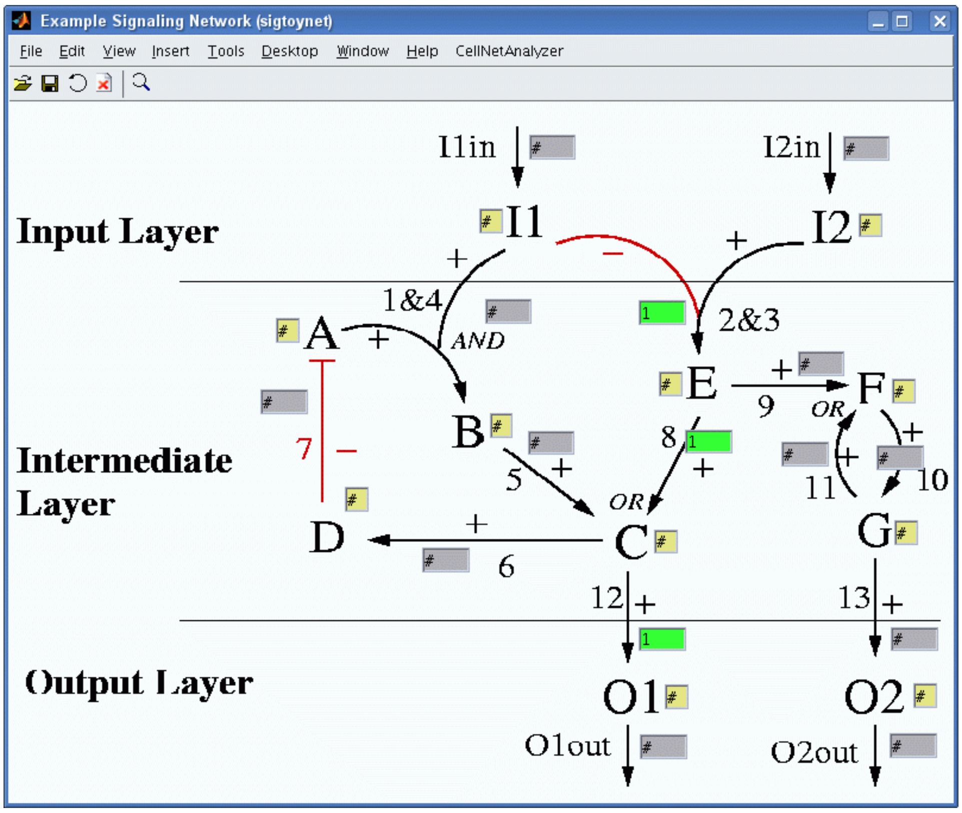
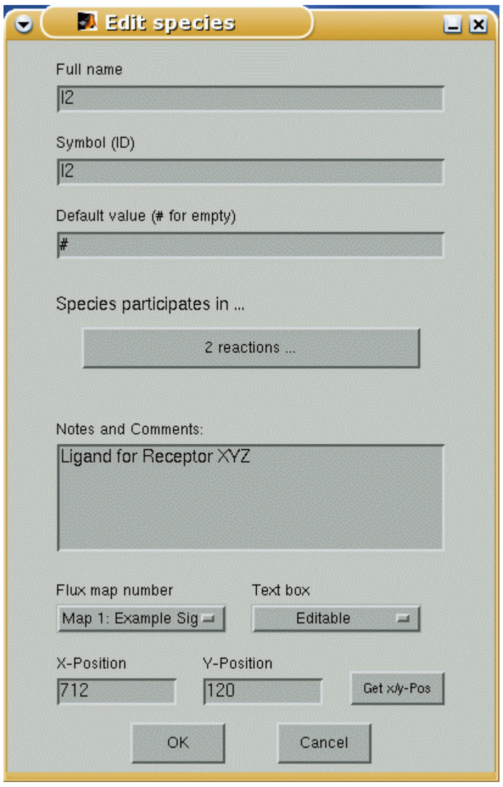

Example Signaling Network (sigtoynet)
File Edit View Insert Tools Desktop Window Help CellNetAnalyzer
Input Layer
Intermediate Layer
Output Layer
I1in
I2in
I1
I2
A
B
C
D
E
F
G
O1
O2
O1out
O2out
1&4
AND
2&3
5
6
7
8
9
10
11
12
13
OR
1

Fig. 15: Interactive network map of a simple signal-flow network that could represent a signaling or regulatory network. The green text boxes indicate a signaling path from I1 to O1 (or from I2 to O1; regarding this ambiguity see remarks in section 4.3.).

4.2 Composing and editing the structure of signal-flow networks

Bearing the technical differences and the new attributes of elements in mind, composing signal-flow networks with the masks provided by CNA is analogous to mass-flow networks (section 3.2). The network composer appears similar as in Fig. 5 (but only with species and reactions as possible network elements) and can be used in the same way as described in section 3.2. Again, the masks for existing species and interactions can be opened by right-clicking on the associated text boxes on the network map. It is not possible to open multiple editing masks at the same time. Therefore if you open a new editing mask any previously opened mask will be deleted and possible changes discarded.

The masks for declaring/editing species and reactions/interactions are depicted in Figs. 15 and 16. The text box parameters can be defined in the same way as described in section 3.2 (and all text boxes can be moved simultaneously using the respective button in the network composer window). There is only one difference: Text boxes for species and reactions can be of type „Editable" and „Non-editable" (as in mass-flow networks) and additionally of type „Non-visible". The latter is helpful in large networks for hiding text boxes of certain (not that important) network elements. This helps to clean up crowded interactive network maps. Note that you can, irrespective of the set default visibility, always hide and display all species /reactions independently during a network session (see menu functions). Furthermore, be aware of that the values currently contained in hidden text boxes are considered as usual when performing computations.


Edit species

Full name: I2

Symbol (ID): I2

Default value (# for empty): #

Species participates in ...

2 reactions ...

Notes and Comments: Ligand for Receptor XYZ

Flux map number: Map 1: Example Sig

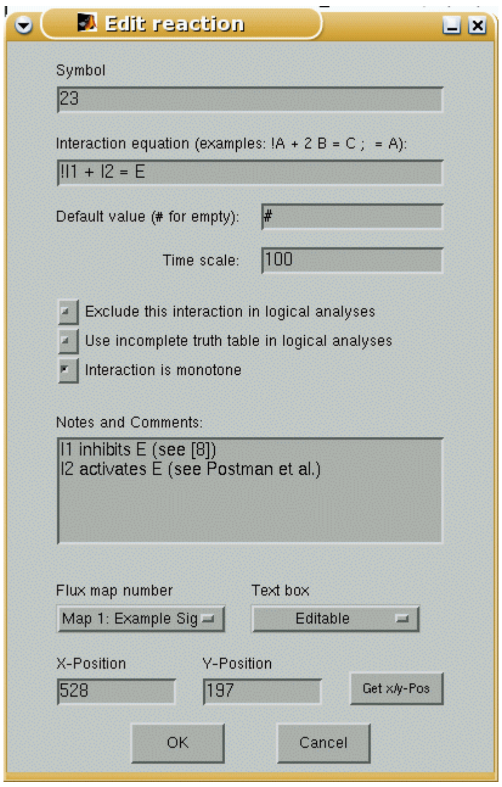
Text box: Editable

X-Position: 712

Y-Position: 120

Get x/y-Pos

OK Cancel


Edit reaction

Symbol: 23

Interaction equation (examples: !A + 2 B = C ; = A): !I1 + I2 = E

Default value (# for empty): #

Time scale: 100

[ ] Exclude this interaction in logical analyses

[ ] Use incomplete truth table in logical analyses

[✓] Interaction is monotone

Notes and Comments:
I1 inhibits E (see [8])
I2 activates E (see Postman et al.)

Flux map number: Map 1: Example Sig

Text box: Editable

X-Position: 528

Y-Position: 197

Get x/y-Pos

OK Cancel

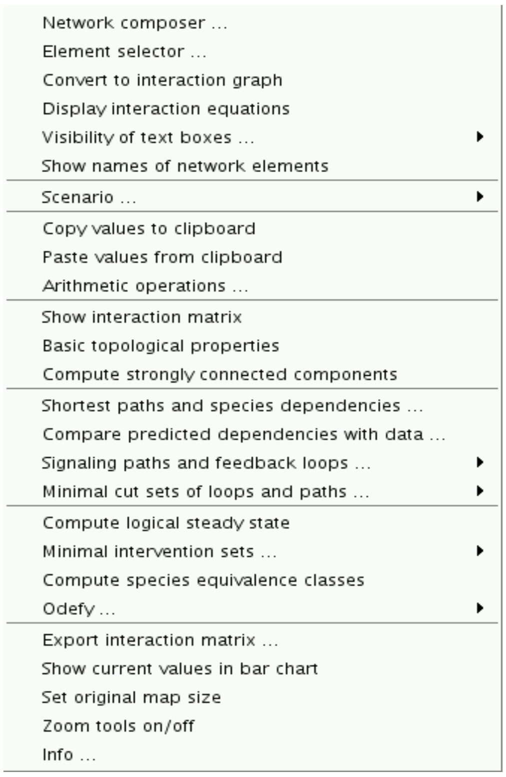
Fig. 16 (left): Mask for editing/declaring species in signal-flow networks.
Fig. 17 (right): Mask for editing/declaring reactions (interactions) in signal-flow networks.

Searching for elements in the network (element selector)

See corresponding section (3.2) for mass-flow networks.

4.3 Toolbox (menu functions) for analyzing signal-flow networks

CNA provides a number of functions for studying signaling networks and the menu (Fig. 18) is similar as the menu for mass-flow networks (Fig. 11): it has a similar set-up, it is partitioned into several groups of functions and can be called via the menu item “CellNetAnalyzer” within the interactive maps (Fig. 18).


Network composer ...
Element selector ...
Convert to interaction graph
Display interaction equations
Visibility of text boxes ...
Show names of network elements
Scenario ...
Copy values to clipboard
Paste values from clipboard
Arithmetic operations ...
Show interaction matrix
Basic topological properties
Compute strongly connected components
Shortest paths and species dependencies ...
Compare predicted dependencies with data ...
Signaling paths and feedback loops ...
Minimal cut sets of loops and paths ...
Compute logical steady state
Minimal intervention sets ...
Compute species equivalence classes
Odefy ...
Export interaction matrix ...
Show current values in bar chart
Set original map size
Zoom tools on/off
Info ...
Fig. 18: Menu with functions for studying signal-flow networks.

The menu is partitioned into the following groups, with (*) and (**) indicating the core functions for exploring the functional properties of the interaction network:

- Editing/Viewing the network structure and properties
- Saving/Setting/Loading scenarios
- Clipboard and arithmetic operations
- Basic properties of the network structure
- Tools for analyzing the (underlying) interaction graph (*)
- Logical analysis (**)
- Miscellaneous

Many menu items are analogous to their counterparts in mass-flow networks (cf. section 3.3).

<u>4.3.1 Editing/Viewing the network structure and properties</u>

Network composer ... : This opens the main window for editing and saving the network structure as it was described in detail in the previous sections.

Element Selector ...: Tool for searching network elements fulfilling a given set of properties. Explained in detail in section 3.2 (Fig. 10).

Convert to interaction graph: Each interaction with AND connections (i.e. hyperarcs with at least two species on the left-hand side of the interaction equation) are split into single graph-like arcs. For example, an interaction equation named ‘ieq1’ that reads

A + !B + 2 C = D

would be split into the three arcs

A = D;   !B = D;   2 C = D

The original interaction (‘ieq1’) is removed whereas the three new arcs represent now new interactions. Each of them gets a new name (composed of the original name of the split interaction and the name of the source species of the arc; in the example above: ‘ieq1_A’, ‘ieq1_B’, ‘ieq1_C’) and an associated text box, which is positioned closely to the original text box. For certain computations, this explicit graph representation is more appropriate. For example, the computation of signaling or influence paths (see below) is internally always performed within the underlying graph model and then mapped back to the hypergraph model which might sometimes lead to non-unique mappings. For example, Fig. 15 shows a signaling path computed with CNA (see below) in which the interaction (hyperarc) 2&3 participates – it is actually not clear whether the path starts at the left (I1) or the right (I2) branch of 2&3. After changing the network into an explicit graph representation, the display of this path would unambiguously indicate that either I1 or I2 is involved. Certain computations (e.g. minimal species cut sets of paths) even require the graph representation.
Before the network is converted to an interaction graph, the user is asked whether he really wants to perform the conversion since the original hypergraph model will be lost in this session (functions for logical analysis are usually not sensible anymore, e.g. splitting 2&3 in Fig. 15 would mean that either I1=1 OR I2=0 would suffice to activate E which is not equivalent to the original hypergraph model). Hence, changes made in the hypergraph model before should first be saved (whereas the new (graph-like) network structure may also be saved separately.). Furthermore the user is asked whether CNA should remove parallel (duplicate) arcs which may arise during splitting. For example, assume you have two interactions that may activate a species Z, namely A + B = Z and A + C = Z. Splitting these two hyperarcs would lead to four new arcs: A = Z, B= Z, A = Z, C = Z. Thus, the connection A = Z would be represented by two equivalent (parallel) arcs. This may lead to undesired effects. For the example above: computing signaling paths or feedback loops, all paths and loops that lead over A and then Z would be computed twice as there are two parallel arcs from A to Z. This can be avoided by letting CNA remove parallel arcs when converting the network to an interaction graph.

Display interaction equations: Displays all interaction equations in the command window.

Visibility of text boxes ...: This sub-menu allows one to hide/make visible the text boxes of reactions and/or species. This is often useful to make crowded network maps easier to read. The default visibility of all text boxes may be set back whenever required.

Show names of network elements: In each text box the name of the associated network element (reaction or species) is shown. This function is especially useful when moving text boxes (“Move Text Boxes ...” in network composer window; section 3.2).
The names of reactions and species, respectively, are also shown as tooltip strings when moving

the mouse over the text boxes.

4.3.2 Saving/Setting/Loading scenarios

Scenario ...: is a sub-menu with several functions for saving, setting, loading and displaying scenarios. A scenario is the set of numeric values of reactions and species that are currently displayed in the respective text boxes. The sub-menu contains several items:

Scenario.../Reset last scenario: The values defined prior the last calculation are displayed in the text boxes.

Scenario.../Set default scenario: The default values are displayed in the text boxes. For network elements without a default value, "#" is displayed indicating an unknown/undefined value.

Scenario.../Save scenario: Opens a dialog box for saving the current text boxes values in a file. By default, the file has extension "val" ("values").

Scenario.../Save species values for ProMoT: Opens a dialog box for saving the current species values in a text file that can be loaded by ProMoT (the values can then be displayed in the context of the network using ProMoT's Visual Explorer). The user also has to define an "attribute name" that will be required by ProMoT (see ProMoT manual).

Scenario.../Save reaction values for ProMoT: Opens a dialog box for saving the current reaction values in a text file that can be loaded by ProMoT (the values can then be displayed in the context of the network using ProMoT's Visual Explorer). The user also has to define an "attribute name" that will be required by ProMoT (see ProMoT manual).

Scenario.../Load scenario: For loading a scenario (formerly saved by "Save scenario"). A window opens for selecting a scenario file (by default with extension "val"). In case that some numeric values are already given in the text boxes, the user will be asked whether the given values are to be overwritten (current text box values of all those elements are retained for which no new value is assigned in the loaded scenario) or whether all text box values are set to be undefined ('#') before loading the scenario. After loading, the text box values are highlighted in the same way as done by Scenario.../Highlight values (see below).
Attention has to be paid if the scenario had been saved prior the network structure has been modified. Then, problems may occur if names of network element have been changed. Network elements that have been added will not cause problems (they get value "#").

Scenario.../Clear all values: The string "#" (meaning "undefined") is displayed in all text boxes.

Scenario.../Clear all reaction values: The string "#" (meaning "undefined") is displayed in all reaction text boxes, current text box values for species remain unchanged.

Scenario.../Clear all species values: The string "#" (meaning "undefined") is displayed in all species text boxes, current text box values for reactions remain unchanged.

Scenario.../Set all input/output gates to zero: A zero value is displayed in all text boxes

(i) of input and output species, i.e. of species without predecessor or without successor
(ii) and of all interactions representing an input or output arc.
Other text box values may be retained or not (the user is requested). This prevents (activating) signal input crossing the boundaries and is useful, for example, for computing feedback loops.

Scenario.../Set time scale scenario: The user is requested to enter a time scale. Then, all reactions having a time-scale larger than the entered value will be assigned a zero value (indicated also by color "special" assigned for the respective text boxes). The user is asked whether the values of all other text boxes should be retained or cleared (set to "#", i.e. undefined).

Scenario.../Highlight values: This function highlights all text boxes with a given (known) value; zero values are indicated by color "special" and non-zero values by color "computed" (see section 2). Undefined values get standard color. This is especially useful in large networks.

Scenario.../Highlight values (on/off): This function colors the reaction and species text boxes according to their displayed value. Text boxes with a zero value are indicated by color "special" and those with non-zero values by color "computed". Boxes with non-numerical values remain in standard color. This function is useful to quickly identify on/off values of the current scenario.

Scenario.../Highlight values (heatmap): This function is similar as the previous one but uses the heatmap style for highlighting the text boxes. The user has to define two threshold values; one for reaction text boxes and one for species text boxes. These thresholds define the 'zero' or 'neutral' value. All text boxes having a larger value than the threshold (separately for species and reactions) will become green; the saturation of the green hue is determined by the ratio

(text box value)/(max_value_of_all text_boxes – threshold value).

Analogously, for all text boxes having a smaller value than the threshold value, the text box color becomes red; the saturation of the red hue is determined by the ratio:

(text box value)/(threshold value – min_value_of_all text_boxes).

(Boxes with non-numerical values have standard color.) One may use this function, for instance, to display and highlight measured data of gene expression or protein phosphorylation (activity) within a network context.

4.3.3 Clipboard and Arithmetic Operations

Values to clipboard: Works like a usual clipboard: The currently given numeric values in the text boxes are copied to a clipboard. The following two functions can be applied to the clipboard.

Paste values from clipboard: Takes the values from the clipboard and pastes them back into the text boxes.

Arithmetic operations: Useful for arithmetic operations on reaction and species values of two different scenarios (e.g. computing and highlighting differences). The four basic calculation operations are available for linking the values currently displayed in the text boxes with the values from the clipboard. Alternatively, arithmetic operations can also be performed with a user-defined numerical value. The results are displayed in the text boxes.

4.3.4 Basic properties of the network structure

Show interaction matrix: The interaction matrix is displayed graphically giving a quick overview of the network structure. The rows of the displayed matrix correspond to the species (names given on the left boundary of the window) and the columns to the reactions / interactions (names given on the lower boundary of the window). A red matrix element e_{ij} corresponds to a species i which has an inhibiting influence (NOT operation) in the j-th interaction. Analogously, a green matrix element e_{ij} corresponds to a species i which has an activating influence in interaction j. A blue matrix element e_{ij} corresponds to the species i which is activated in reaction j. A black matrix element e_{ij} indicates that species i does not participate in reaction j. The connectivity number of each species (number of interactions where the species is involved) is given on the right boundary of the window (number in brackets: number of interactions where the species activates/inhibits/is activated). If the network is very large (more than 120 reactions or/and species) the display of this matrix might fail or become unreadable.

Basic topological properties: Displays some basic topological properties in the command window. This function is particularly useful to detect errors in the network structure that might have been inserted during network construction.

- *Species participating in no interaction* are detected
- *Sources and Sinks*: Species that are not influenced by others (=sources: no predecessor) or that do not influence others (=sinks: no successor) are detected.
- *Input Arcs and Output Arcs*: Interactions with an empty left-hand side (INPUT) or with an empty right-hand side (OUTPUT) in their interaction equations are identified. These arcs typically represent signal flows across system boundaries (e.g. *I1in* and *O1out* in Fig. 15).
- *All non*-monotone interactions and those with incomplete truth table or to be excluded in logical computations (see Table 2) are displayed.
- *Parallel Reactions*: Reactions with the same participating species are detected.
- *Number of different time scales.*
- *Basic Properties* of the interaction matrix.

Compute strongly connected components: A strongly connected component (SCC) is a maximal subgraph of the interaction graph in which paths between all pairs of nodes exist. Consequently, when searching for cycles that involve a certain node it is only necessary to search in the SCC that contains this node. After calculation, SCCs that comprise more than two nodes are visualized on the network map: The species that belong to the same SCC are labeled with the same number and in addition their text boxes are colored with the same color. Because colors (currently four) are used repetitively, species belonging to different SCCs can have the same color but their labels are unique to the SCC.

4.3.5 Tools for analyzing the underlying interaction graph

Shortest paths and species dependencies ...: Similar function as “Graph-theoretical path lengths ...” in mass-flow networks but even more powerful. Allows for the calculation of *graph-theoretical* (shortest) path lengths between all species, separately for negative and positive paths from which then (by taking the minimum of both) the unsigned shortest path is also computed. Additionally, the average path length and the network diameter are computed (for the unsigned shortest path length). Furthermore, a dependency matrix is determined (see below) which is extremely helpful for detecting functional dependencies between all pairs of species.

Note that a signal-flow network in CNA is a Boolean network represented as a *hypergraph*. Therefore, for applying graph-theoretical methods, this hypergraph is temporarily transformed into its underlying interaction graph (see also “Convert to interaction graph”). Regarding the interpretation it is important to realize that the shortest paths which are computed with this function indicate whether there is a positive/negative/any influence of a species A on B, without a guarantee that the path is active under a given scenario in the logical model. Note that the shortest paths algorithms work well also in large networks.

Prior to the computation, the user can choose several options (Figure 19):

- *Exclude species with given zero value*: This option can be used to exclude certain species and, hence, certain possible paths in the network while computing the shortest paths (enables, for example, to simulate a certain failure or knock-out). Just set a zero value in the text boxes of species to be excluded and select this option.
- *Exclude reactions with given zero value:* This option allows one to exclude certain interactions and, hence, certain possible paths in the network while computing the shortest paths. Set a zero value in the respective reaction text box before computing the paths and select this option.
- *Display distance matrices*: Displays in three separate windows the shortest positive/negative/unsigned path lengths (distances) as colored arrays. The array element e_{AB} represents the shortest path length from A to B. The larger the path length is the brighter the array element. A blue cell marks species pairs between which no path exists.
- *Display dependency matrix*: Finally, a dependency matrix is determined which characterizes the functional dependency between each pair of species: The color of the matrix element e_{AB} indicates one of 6 possible types of dependency: (1) black: no influence of A on B, (2) yellow: A has activating and inhibiting effect on B, (3) light red: A is pure inhibitor of B, (4) light green: A is pure activator of B, (5) dark red: A is total inhibitor of B), (6) dark green: A is total activator of B. Dependencies with respect to one single species may also be displayed directly within the network maps after the computation has been finished (see below).

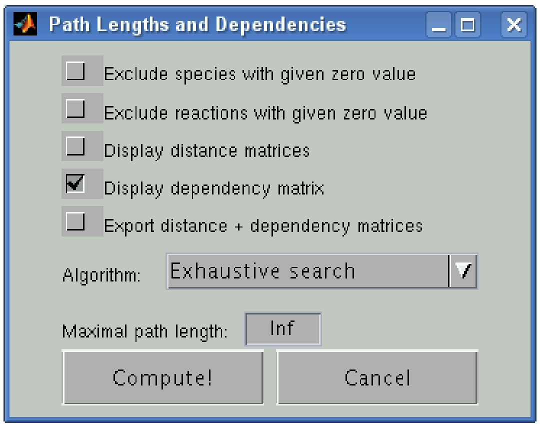
In very large networks, the creation of distance and dependency matrices might fail in MATLAB. Use then instead the option:

- *Export distance + dependency matrix*: This allows for an export of the calculated distance and dependency matrices into an ASCII file. The user may then further study the path lengths matrix by its own algorithms. Each distance matrix (total, negative, positive path lengths) and the dependency matrix will be exported in separate files (the user specifies the files upon finishing the computation). The dependency matrix contains in each cell one of the classification number 1-6 as described above.
- *Algorithm*: In contrast to standard shortest path algorithms in unsigned graphs, the computation of shortest positive / shortest negative paths and cycles is not trivial; the algorithm is not polynomial-time and computing the pos./neg. paths may thus take a long time (dependent on network size and structure). CNA provides three algorithms: (i) a simple exhaustive search algorithm which delivers exact results and which – although not polynomial – proved to be efficient enough also for larger networks with several hundreds of nodes (ii) for larger networks or those with very complicated structure (e.g. many negative feedback loops), CNA provides an approximative algorithm which is faster (polynomial complexity; seconds or minutes in large and complicated networks) and has, to our experience, a high probability to compute the correct distances [14]. The approximative algorithm is definitely correct if no

negative feedback circuit is contained in the network. However, for special configurations it may happen that negative (positive) paths between a pair of nodes A and B are overlooked if a positive (negative) path from A to B exists and the network contains negative feedback loops. (iii) Another exact algorithm which turned out to be favorable (compared to the first one) in certain classes of networks is the Two-Step-Algorithm [14]. In general, try first the exact algorithms (i) or (iii), if they are too slow take (ii).

- *Maximal Path Length*: You are allowed to restrict the paths to be considered by setting an upper bound for the path length (longer distances will not be considered decreasing also the computation time).

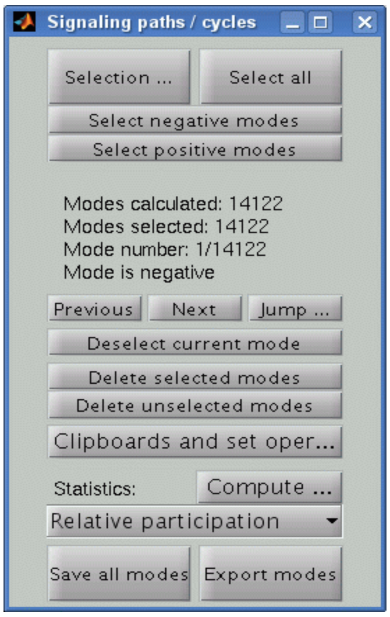
After computation, depending on the chosen options, the distance matrices and/or dependency matrix are shown and exported if desired. Besides, average path length, network diameter (longest shortest path length) and the number of connected components in the graph are displayed in the command window.
Finally, a small panel comes up allowing a detailed influence/dependency analysis (relying on the computed dependency matrix): Enter a species identifier in the respective input field, select (case 1) "Dependency on all species" or (case 2) "Influence on all species" and click then on "Compute". In the first case, each compound is classified into one of the following categories: (total) activator (only positive paths to the selected species exist), (total) inhibitor (only negative paths to the selected species exist), ambivalent factor (positive and negative paths to the selected species exist) or non-affecting factor (no path to the specified species exist at all). The classification is displayed in the species text boxes, also illustrated by different box colors. In the second case, the converse relationships is determined (for which compounds is the specified species an (total) activator, (total) inhibitor, ambivalent factor or non-influencing) and then displayed in the species text boxes. Regarding the classifications, in particular for *total* activators and inhibitors, see [11].


Path Lengths and Dependencies

☐ Exclude species with given zero value
☐ Exclude reactions with given zero value
☐ Display distance matrices
☑ Display dependency matrix
☐ Export distance + dependency matrices

Algorithm: Exhaustive search

Maximal path length: Inf


Signaling paths / cycles

Modes calculated: 14122
Modes selected: 14122
Mode number: 1/14122
Mode is negative

Statistics: Compute ...
Relative participation

Fig. 19 (left): Options for computing shortest paths and species dependencies
Fig. 20: Control panel for displaying and analyzing signaling paths and cycles.

Compare predicted dependencies with data ...: With the help of the dependency matrix, causal dependencies between each pair of species in the model can be predicted. Having data available where the activation level of certain species is measured in response to different stimuli (e.g. adding ligands and/or inhibitors), one can detect whether an increase or decrease in the activation levels is in accordance with the predicted causal dependencies (see [17]). For example (Fig. 15), assume species O2 shows a higher activation level when it is measured in absence of input I2 than it does in presence of I2. Since I2 is a (strong) activator for O2 in this network, this leads to a contradiction between the model and the experimental data: an increase in I2 cannot induce a decrease in O2, as there are only positive paths connecting I2 and O2. (Hint: this function assumes that the data reflects the initial responses of the species, i.e. weak activators and weak inhibitors contained in the dependency matrix are handled in the same way as strong activators and strong inhibitors, respectively.)

The information on the experimental data must be stored in a structure array named *CNADepProject* that contains the following fields:

CNADepProject.InputMatrix: Matrix that contains in the rows the different experimental scenarios (i.e. adding a ligand, inhibiting a certain species ...) and in the columns the species. It is not necessary that all metabolites of the current model have a column in the InputMatrix (however, the column order must fit with the MetaboliteNames field and the columns in OutputMatrix (see below)). A positive entry corresponds to adding a ligand or activating a certain species (a larger value indicates a higher concentration) whereas negative entries are used for inhibitors (a smaller value (i.e. larger absolute value) indicates a stronger inhibitor). Also important, a species having an undefined value is indicated with ‘NaN’.

CNADepProject.OutputMatrix: Matrix that contains observed species values (measured or from literature). The rows (scenarios) and columns (species) must fit with the InputMatrix.

CNADepProject.MetaboliteNames: Cell array of strings or character array that contains the metabolite (species) names (the first entry corresponding to the first column of the InputMatrix or OutputMatrix, second entry to second column and so on). The metabolite names have to fit with the species names in the model (although not all species of the model must be present here, e.g. if they have not been measured).

CNADepProject.NegativeStates (optional): Cell array of strings or character array that contains species that are assumed to be constitutive active at time 0 and are inactivated when the signal proceeds. If the field does not exist, an empty field is created.

CNADepProject.ScenariosToCompare (optional): Matrix that contains in each row one pair of scenarios the user wants to compare. For example to analyze the influence of a ligand, one compares a scenario x where the ligand was added with a scenario y where it was not added. The respective row in the matrix ScenariosToCompare would store the indices of these two scenarios.

If this field does not exist, all scenarios are compared (pairwise) to each other.

When starting this routine, similar to the computation of shortest paths and dependencies, the user can choose several options prior to the computation. For the comparison of the predicted dependencies with experimental data the model’s dependency matrix is computed. Therefore, the first three options are already known from the previous function (see figure 18):

- *Exclude species with given zero value*
- *Exclude reactions with given zero value*
- *Compute approximative dependency matrix*

Additional options are:

- *Display comparison results:* The result of the comparison is displayed as colored matrix, where the rows correspond to the compared scenarios and the columns to the measured species. Row labels indicate which ligands/inhibitors were added in both scenarios with the same concentration (green dot for added ligands, red dot for added inhibitors) and how the stimuli of the two scenarios differ (e.g. ligand A↑). For example, the label I1 •, A •, I2 ↑ refers to the comparison of two scenarios that were both stimulated with the same concentrations of ligand I1 and of an inhibitor for A, but with different concentrations of ligand I2 (possibly I2 was only added in one of both scenarios), i.e. the influence of an increased level of I2 is analyzed. The color of a matrix element indicates one of 10 possible cases that are explained in the legend displayed in another window (e.g. light green: treatment causes increase as expected; orange: no effect expected, but treatment causes decrease).
- *Export comparison results:* Saves the structure *depmat_compare_results* to file. The structure contains the following fields:

  depmat_compare_results.ExpectedDependencies: Matrix that describes for each comparison the expected up/down behavior (resulting from the model structure).

  depmat_compare_results.FoldChange: Matrix that contains the fold changes in the compared scenarios.

  depmat_compare_results.DiscChange: Matrix that contains discretized values that one obtains by comparing the FoldChange with a fixed threshold (see below). 1: significant increase (FoldChange > threshold); -1: significant decrease (FoldChange < 1/threshold); 0: no significant change; 5: states that are not compared, because an external intervention on the measured species was made.

  depmat_compare_results.ComparisonResult: Matrix that contains the result of the comparison, coded with 1, 2, ..., 10. 1: no significant change, positive or negative effect expected; 2: increase although no change expected; 3: decrease although no change expected; 4: increase although negative effect expected; 5: decrease although positive effect expected; 6: influence on a species is not considered if an external intervention on this species was made; 7: decrease as expected; 8: increase as expected; 9: no significant change as expected; 10: ambivalent in the model.

  depmat_compare_results.Label: Row labels, indicating which scenarios are compared and how the stimuli differ.

Finally, the fields *Ratio threshold* and *Data file* should be specified:

- *Ratio threshold*: threshold to define when the FoldChange is significant. FoldChange > threshold: significant increase; FoldChange < 1/threshold: significant decrease. Default value is 1.5.
- *Data file*: Enter the path of the MATLAB (mat) file where you saved the *CNADepProject* data structure. Alternatively, you can use the *Browse* button.

Start the computation by clicking the *Start!* button.
To test this function an example data file (containing a fictitious dataset) for the Signaling Toy model is provided (‘testdepmatcomp.mat’ in the ‘sigtoynet’ directory).

Signaling paths and feedback loops ...: This sub-menu allows one to compute, display and analyze feedback loops and signaling paths in the underlying interaction graph of the logical network model. In general, it is recommended to first convert the logical (hypergraph) model into an interaction graph (see above) when using this function, however, it can also directly be performed in the logical network. The sub-menu provides two main functions:

Feedback loops ...: Computes the internal feedback loops (= circuits or (elementary) cycles) in the underlying interaction graph of the network. For example, in Fig. 15, there are two internal circuits (loops, directed cycles). Although the elementary modes algorithm can be used for computing the cycles (see [11]), CNA uses a backtracking algorithm for enumerating the cycles. The following three options can be chosen in a dialog box appearing after selecting this menu item:

> (i) *Enforce/Exclude reactions by given values*: if selected, the numerical values given in the text boxes of the reactions (interactions) confine the set of cycles to be computed (similar as for elementary modes in metabolic networks): all interactions with a zero value in their text box are blocked during circuit computation and if an interaction has a non-zero value in its text box then CNA will compute only those cycles that involve this interaction. For example, to compute only the cycles involving interaction 6, one would set a "1" in the associated text box (one circuit would be found). Note that it is not possible to enforce a "hypergraphical" interaction where several species are AND-connected. Thus, interactions "1&4" und "2&3" can not be enforced in Fig. 15. For those cases, you should first convert the network into an interaction graph (see above). Another important issue is, that multi-level interactions (coefficients larger than 1 in an interaction equation) are treated as binary interactions (all non-zero coefficients are set to 1).
>
> (ii) *Enforce/Exclude species by given values*: as for reactions, when selecting this checkbox, the numerical values given in the text boxes of the reactions (interactions) confine the set of cycles to be computed: only those cycles are computed, that run over all the nodes having a non-zero value in their box and that do not visit nodes with a zero-value in their box. For example, to compute the (only) circuit where A is involved, write a "1" into the box of A, select this checkbox and start the calculation. As for reactions, multi-level interactions (coefficients larger than 1 in an interaction equation) are treated as binary interactions (all non-zero coefficients are set to 1).
>
> (iii) *Consider unsigned graph*: if selected, CNA ignores the direction of the arcs (i.e. an undirected graph model is evaluated). This can be useful e.g. to compute also feedforward loops (there are two such feedforward loops in Fig. 15 (additionally to the two feedback loops). (Note that option (ii) is similar to option "Check reversibility" for elementary modes in mass-flow networks).

Signaling paths...: This function works similar to the previous one: Instead of feedback loops it computes all paths connecting one species from a specified set of start nodes with another species from a specified set of end (target) notes. In the upcoming panel, the user may specify a set of start and a set of target nodes by entering the respective species identifiers. Additionally (as a short cut), he may optionally include all input nodes (= nodes that have no predecessor or are connected to an input arc) to the set of start nodes or/and all output nodes (= nodes that have no successor or are connected to an output arc) to the set of target nodes. As explained above for feedback loops, one may optionally exclude or enforce certain interactions and/or nodes by setting zero/non-zero values into the respective text boxes prior to computation. An example: five signaling paths would be found, if one selects the input nodes (I1, I2) as start nodes and the output nodes (O1, O2) as target nodes. Only two of them would remain if one sets a 1 in the text box of interaction 9 and a zero in the text box of reaction 8 (and enables the checkbox '*Enforce/Exclude reactions by given values*').

When the computation of cycles or signaling paths has finished, a control panel comes up (Fig. 20) which is very similar to the panel for managing the computed set of elementary modes in mass-flow networks (Fig. 13, paths and cycles are also referred to as modes in Fig. 20). The usage is completely analogous as explained for elementary modes in section 3.3. Paths and loops can be displayed subsequently in the network by using the "Next" and "Previous" button. Involved reactions on a path are indicated by value "1" and the text box color is "calculated". Note again that paths and cycles are computed in the interaction graph and then mapped back to the logical hypergraph (see remarks for menu item "Convert to interaction graph"). Specific selections of modes can be made by using the 'Selection tool'. The same statistical computations can be made as for elementary modes; exceptions are the features "(Product) Yields and rate ratios" and "Control-effective fluxes" which are for obvious reasons not available in signal-flow networks. Regarding Minimal Cut Sets see below.
There is one major difference regarding elementary modes in stoichiometric networks and paths and cycles in signaling networks: a reversibility property is not specified in signal-flow networks. On the other hand, paths and cycles can be positive (activating influence) or negative (inhibitory influence), depending on the parity of negative signs (NOTs) on the path. The sign of the currently displayed mode is therefore shown in the control panel. Furthermore, two additional buttons allow for a selection of all negative or all positive modes of the current selection. With the clipboard functions and the buttons 'Delete selected modes', 'Delete unselected modes' and 'Deselect current modes' one may assemble more complicated selections of the set of paths/loops (as explained for elementary modes in section 3.3.). Finally the signaling paths and feedback loops may be saved or exported to an ASCII file.

The other two items of the "*Signaling paths and feedback loops...*" sub-menu are *Load* ... and *Show* ... which allow one to load a set of paths/loops saved in an earlier session or to show a set of paths/loops computed or loaded earlier in the current session (and still being in memory, see also section 3.3. for the same menu items for elementary modes).

Minimal cut sets of loops and paths...: As the minimal cut sets in mass-flow networks attack a selected set of elementary modes (section 3.3), the minimal cut sets (MCSs) computed with this function attack (cut, interrupt) the currently selected set of paths or/and loops in a signal-flow network. Hence, before computing MCSs for paths/loops the user has to compute these modes by one of the two methods described above.
In signaling networks one may determine MCSs removing (only) *reactions* **or** MCSs removing (only) *species*. (From an experimental point of view, the removal of species is realistic in signaling networks, while practically infeasible in metabolic networks). Therefore, when choosing a function from this sub-menu, the user has to specify whether species MCS or reaction MCSs are to be computed/loaded/displayed.
Computing, analyzing and displaying reaction MCSs is completely analogous as for elementary modes (see section 3.3):

- (i) the user selects a (sub)set of modes (paths/cycles) that is to be attacked (interrupted) – the current selection defines this set of target paths/cycles.
- (ii) As for MCS for elementary modes, using the paths/cycles clipboards one may define up to two sets of paths/cycles for which a minimum number (to be specified) must not be hit by the MCSs to be computed.
- (iii) choose from menu "Minimal cut sets for paths .../Compute..." and choose *reaction MCSs*
- (iv) specify options for calculation (the same as for "metabolic MCSs" (described in section 3.3), except that multifunctional enzymes are not supported)

- (v) when the MCSs have been computed, a control panel comes up allowing to walk through the set of MCSs (while displaying them in the interactive maps), to select certain subsets of MCSs, to compute statistical properties and to save the MCSs (these functionalities are all described in detail in section 3.3)

Note that negative/positive effects of removing reactions are not considered when computing the MCSs (see also item (b) for species MCSs).

Computing MCSs for species, some particularities have to be taken into account:

(a) Species MCSs can only be computed in interaction *graphs* (this is also recommended for reaction MCSs), Therefore, if you want to compute species MCSs for paths, first apply “Convert to interaction graph” from the menu, compute the paths in the interaction graph and then the species MCSs for these paths.

(b) CNA assumes that you want to interrupt a signal flow along a path or within a feedback loop. Whereas this procedure is straightforward for feedback loops and in graphs with exclusively activating (+) arcs, it might have ambiguous effects for signaling paths with negative arcs. For example, assume in Fig. 21 the negative path starting in I1 and leading to O1 via E should be attacked. Each species (I1, E, C, O1) on this path would represent one Minimal Cut Set. However, whereas removing I1 would have a positive effect on O1, removing E would have the opposite effect: it would indeed correctly disrupt the path starting in I1 but the inhibition along this path would virtually still be active since E is an activator of O1 (removing E is structurally equivalent to activate I1 inhibiting E). These effects are not taken into account by the MCS approach – the computed MCSs will only ensure, that no signal can flow from the start to the end node in any of the target paths. Such effects can be much better taken into account when computing Minimal Interventions Sets (see below) in the logical model, possibly with incomplete truth tables.

The rest is analogous as for reaction MCSs. Options for the computational procedure may be specified (same options as offered for reaction MCSs except that compact storage is not supported). Then, a control panel comes up which is similar as the one for reaction MCSs and analogous in usage. Of course, the main difference is, that for a displayed MCSs not reactions but species are marked in the network maps. Inactivated (removed) species are indicated by a zero value (the box color is of type “special”) in the interactive maps. Specific selections of MCSs can be made, some statistical features can be computed (again analogous as for MCSs in metabolic networks) and all MCSs can be saved or exported.

Again, with the *Load ...* item in the minimal cut sets sub-menu one may load a saved set of (reactions or species) MCSs.

4.3.6 Logical network analysis

Compute logical steady state: Takes the currently defined reaction values (=fixed signal flows) and species values (= fixed states) and computes from this information – as far as uniquely possible – the resulting logical steady state of the species as it follows from the Boolean network structure. This function is in a certain sense similar to metabolic flux analysis (section 3.3).
First, the user may fix some states of species (which is particularly relevant for INPUT species and for indicating knock-outs or knock-ins in the network). Additionally, the user may enter values for signal flows along selected interactions. This is particularly relevant for INPUT arcs (see Figure 20), but it might also be useful to deactivate certain interactions by setting the signal

flow along this interaction to zero. Note that only INPUT arcs are allowed to have a zero or non-zero value whereas all other interactions may only be assigned a zero (non-zero values will be ignored in the latter case). The order of evaluation of given species/reaction values is as follows:

- if a species value has been defined, then the logical state of this species will be fixed to that value; any given value for a reaction that points into that species will be ignored
- a reaction will be considered to be off (i.e. temporarily removed) if a signal flow of zero has been defined for that reaction (note: if all reactions pointing into a species have been set to zero, this species will be assigned a zero in steady state, i.e. this is analogous to setting the respective species value to zero)
- if a non-zero value has been defined for an INPUT arc, then the state of the node connected to this INPUT arc will be fixed to that value. If non-zero values for several INPUT arcs pointing into one and the same species have been defined than the one with the highest activation level will be taken to fix the state of the species. If a non-zero value has been defined for at least one INPUT arc all other signal flows pointing into that reaction will have no effect on the logical steady state of this species. Thus, if you have two interactions "= A" and "B = 2 A" then setting a "1" for the first interaction and letting the second undefined means that A will be fixed to 1 irrespective whether B will be assigned a value of 1 or not. Furthermore, for an INPUT arc of type " = 2 A", setting any value different to "2" will be automatically converted to "2" in order to be consistent with the level of the signal level flowing along this reaction. Generally, a simpler way than using INPUT arcs is to set the respective node value directly in the associated species box.
- setting a non-zero value for reactions that are not INPUT arcs will be ignored

After defining fixed species and reaction values and starting this function, the logical steady states resulting from the given scenario are computed. The algorithm tries iteratively to derive logical steady states for species until no further update is possible (see reference [11]). Fig. 21 shows an example: Setting the signal flows for the input arc *I1in* to zero and for *I2in* to 1 (given signal flows of input arcs are colored as "defined" – here in blue) and letting all other species and reaction values undefined, the logical steady state as displayed in Fig. 21 would result. For example, since I1 is inactive and I2 active, E becomes activated by hyperarc 2&3 and will remain in this active state (so the logical (qualitative) steady state of E will be 1). Then E will activate C, O1, D, F, G and O2. B becomes inactive because I1 is inactive and, since D is active, A becomes inactive. Resulting zero values for states (species) and signal flows (reactions) are indicated by color "special" (here red), non-zero values (the activation level) are indicated in color "calculated (here green) and unknown values in standard color. The signal flow values indicate whether an activation of the connected target species occurs (non-zero value: the level of activation reached) or not (zero value) along this interaction.

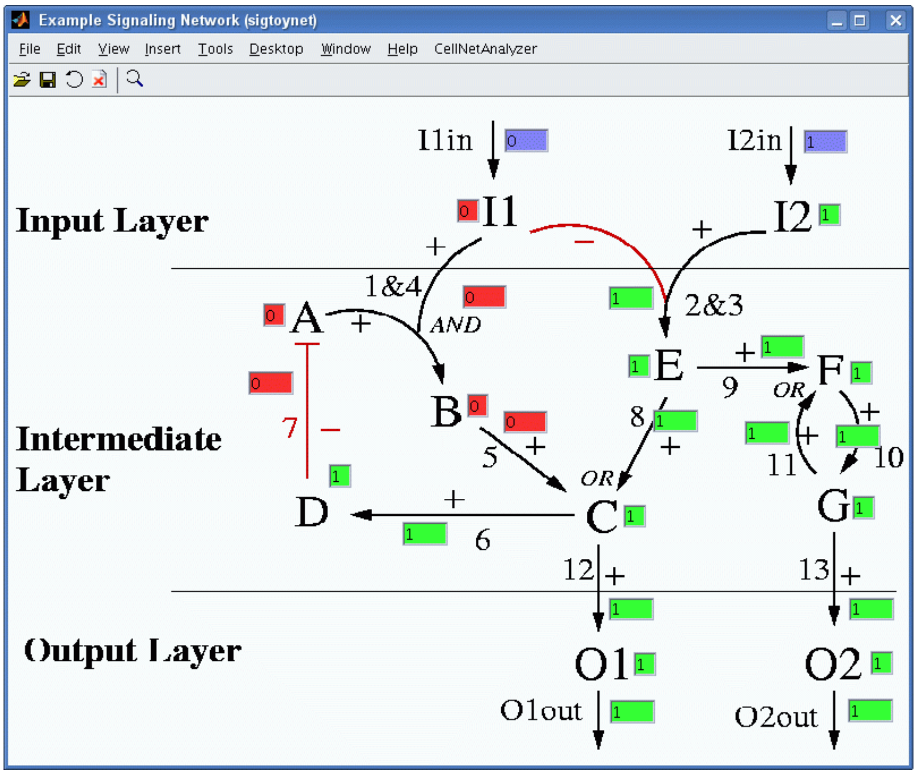


Fig. 21: A scenario of logical steady state analysis (small boxes: species; large boxes: reactions). Predefined values are in blue

Important remarks:

- The logical steady state resulting from a scenario depends heavily on the fixed signal flows and species states. It is based on an algorithm for three-valued logic (see ref. [20]).
- Note that a (global) logical steady state might not exist at all for a given scenario or that it cannot be resolved completely. For example, setting only the signal flow of input arc at I2 to 1 and everything else as unknown, nothing could be concluded. Setting instead the incoming signal flow at I1 to 1, we could derive at least a (partial) logical steady state, namely E = 0.
- There are a few cases where CNA does not identify existing logical steady state values. CNA does not (yet) check whether positive feedback loops are self-sustaining by initial state values of participating species (for example, in a dynamic Boolean network model, assuming initial values of "1" for F and G would result in partial logical steady states F=G=1 due to the positive feedback loop between F and G). Positive feedback loops can, however, be identified by the algorithms described above and therefore be checked by the user himself. There might be some other, for cellular networks very unlikely situations, where existing logical steady states are not identified. Certainly, all steady state values that have been found by CNA are definitely correct.
- As described in section 4.1, if the "Exclude in logical computations" – flag is set for a reaction, then it will not be considered as a possible activation route for the logical steady state identification procedure.
- Note also the particularity of interactions with incomplete truth table (section 4.1)

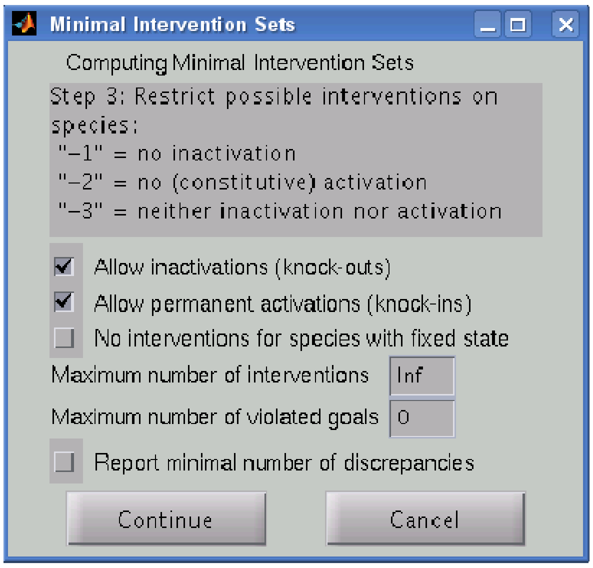
Minimal intervention sets ...: As explained above, minimal cut sets (MCSs) computed for a set of paths/cycles refer to the underlying interaction graph of the logical signal-flow network. In contrast, this function computes *minimal intervention sets* (of species) directly from the logical network, i.e. the more constrained logical behavior of the network is taken into account (see [11,12]). Minimal intervention sets (MISs) are a set of interventions that fulfill a user-defined intervention goal. An intervention can be a permanent deactivation (knock-out) or a permanent activation (e.g. mimicking knock-ins that occur during certain diseases like cancer) of a species. The computation of logical MISs requires three steps guided by CNA:

(i) The user defines an intervention goal. This can be done by setting the *desired levels* of reactions or/and species in the respective text boxes. For example, in Fig. 21 we might set a “0” for species O1 and a “1” for the output arc O2out (which is equivalent to setting a “1” for O2). This intervention goal demands that O2 has to be active in logical steady state and O1 inactive.

(ii) The user may a priori fix certain states (for species) and signal flows (for reactions). [Regarding the conventions for fixing species and reaction values, see the explanations given under “Compute logical steady state”]. Setting such a particular scenario is useful to compute MISs valid only under certain conditions (of interest). In our example, we might assume that I2 is active, i.e. we set a zero for the input arc I2in. The MISs to be computed will then have to ensure that O2 is active and O1 inactive (in logical steady state) under the condition that I1 is inactive.

(iii) First of all, certain species may be marked as non-removable/non-inactivatable (set a “-1” in the respective text box) or as non-activatable (set a “-2”) or as neither inactivatable nor activatable (set a “-3”). Since the computation of logical MCSs is intricate, this may reduce the computation time considerably. In addition, two options can be used to globally restrict the activation/inactivation (cf. Fig 22): (a) Species may be knocked-out (removed/inactivated). (b) Permanent activation of species. Deselecting this option and enabling knock-outs (option (a) is on) is equivalent to set a “–2” in all species text boxes. In contrast, selecting this option and disabling knock-outs (option (a) is off) is equivalent to set a “–1” in all species text boxes. A third option (c) allows to exclude species that have a fixed value (due to the values set in second step)from interventions. This often makes sense because e.g. the value of boundary species is fixed due to external constraints. In our example, we assume that all species are removable and activatable. If we would select option (c), then I2 would be excluded no MIS containing I2 would be computed.

A second set of options can be used to define the maximum number of interventions, the maximum number of violated goals and whether or not to report the minimal number of discrepancies. As for MCSs of elementary modes and paths/cycles, setting an upper boundary for the cardinality of MISs may drastically reduce the computational demand. From a practical point of view, the small MISs are the most relevant ones. If you want to have all leave the default “Inf” which we use also in our example. If the maximum number of violated goals is greater than zero then also intervention sets are accepted that do not fulfill all goals as long as they do not contain more errors than the specified value. When the option to report the minimal number of discrepancies is activated then during calculation the minimum number of errors of all tested intervention sets is tracked. In case intervention sets were found this value simply corresponds to the minimum number of errors of these sets. When no intervention sets were found this value is the minimum number of errors that can be obtained by any intervention set under the specified conditions (in particular the maximum number of interventions). The actual value is printed to the MATLAB console. Note that the determination of the minimum number of discrepancies may increase the computation time compared to the case where this flag is off (in particular when multiple scenarios are used which is currently only possible via the corresponding API function; cf. sec. 7.6).


Minimal Intervention Sets

Computing Minimal Intervention Sets

Step 3: Restrict possible interventions on species:
"-1" = no inactivation
"-2" = no (constitutive) activation
"-3" = neither inactivation nor activation

[✓] Allow inactivations (knock-outs)
[✓] Allow permanent activations (knock-ins)
[ ] No interventions for species with fixed state

Maximum number of interventions: Inf
Maximum number of violated goals: 0

[ ] Report minimal number of discrepancies

Fig. 22: Dialog for step three during the set-up for the computation of minimal intervention sets.

Then, CNA computes the logical MISs which is, as already mentioned, another complex (combinatorial) problem. Upon finishing the computation, a similar control panel as for species MCSs of paths/cycles comes up by which the user can step through the set of MISs and display each MIS in the interactive maps. Inactivated (removed) species are indicated by a zero value (the box color is of type "special") and activated species by a "1" (and box color "calculable"). Specific selections of MISs can be made. Two statistical features can be calculated for the current selection: a histogram of the size of the MIS can be computed or a histogram with the relative participation of each species (separately for permanent activation/deactivation) can be displayed. Finally, the set of computed MISs can be saved or exported.
For our accompanying example, we would get 43 (!!) MISs. Thereof two comprise only inactivations representing "logical cut sets" (as only cuts are contained): {!I1, !C} and {!I1, O1}. Three MISs are built up exclusively by activations: {I1, D, F}, {I1, D, G}, {I1, D, O2}. The remaining 38 MISs comprise at least one inactivation and at least one activation. One example is {!C, E}. (The exclamation mark indicates here a cut i.e. knock-out or inactivation).

Compute species equivalence classes: This function computes equivalence classes of species useful to identify synchronized network regions. All species of an equivalence classes are strictly correlated (positively or negatively) in every logical steady state that can be reached by the network, i.e. there synchronization holds for all possible inputs. For example, A and B are in an equivalence class if it follows from A being on (off) that B is also on (off) and vice versa. Species A and B are also in an equivalence class if they are negatively correlated, i.e. if it follows from A being off (on) that B is on (off) and vice versa. In the latter case the equivalence class would be written as {A, !B} or {!A, B}. You may also clamp values for species/arcs prior computation; in this case the equivalence is computed with respect to this scenario (for example, A and B might be in an equivalence class only if an input, say C, is set to be off). You should also keep in mind that the equivalences are only valid if the network reaches a logical steady state.
The computed equivalence classes are displayed in the text boxes (class number; with alternating text box colors for each class) and in the command window and may also be saved in a text file. Importantly, this function can only be used in binary-level networks.

Odefy ...: *Odefy* is a plugin for CNA developed (with the help of CNA's API) by Jan Krumsiek

and Dominik Wittmann in the group of Fabian Theis at the Helmholtz Zentrum Munich. The *Odefy* homepage can be found here: http://www.helmholtz-muenchen.de/cmb/odefy. *Odefy* uses a new technique for transforming a discrete Boolean network into a continuous ODE (ordinary differential equations) model using multivariate polynomial interpolation ([16]). The ODE model can then be exported (e.g. as MATLAB or SBML file) or be simulated on-the-fly in CNA. *Odefy* also allows the simulation of a Boolean network (with synchronous or asynchronous update rules) and it can also run independently of CNA from MATLAB command line (see *Odefy* documentation). Importantly, so far *Odefy* works only with Boolean, not with multi-valued logical models.
It follows a brief explanation of the two main features (copied from the *Odefy* documentation which can be found in *CellNetAnalyzer/code/ext/odefy/doc/index.html).*

Odefy.../Export ODE model: Odefy can export the generated ODE model as MATLAB scripts, in SB Toolbox format, GNA or SQUAD format. In the dialog which pops up you select a filename where the model will be stored. For MATLAB script files or SB Toolbox format you need to specify whether to create a HillCube ODE model, a normalized HillCube ODE model or a BooleCube ODE model (see [16]).

Odefy.../ Simulate Boolean / ODE model: Simulates Boolean models (only binary logic allowed) – either with synch/asynch. Boolean simulation or using the ODEs generated from the Boolean model. In the Simulation dialog you can specify all settings required for the continuous/Boolean simulation of the logical model:

- *Time units to simulate* - The number of time units to be calculated. For continuous simulations, *Odefy* uses the MATLAB `ode15s` ODE solver function with adaptive approximation intervals.
- *Edit parameters* - Here you can edit the Hill kinetics parameters for each interaction. Only required when simulating with HillCube / normalized HillCube (see [16]).
- *Edit initial values:* Odefy reads the initial values of the species directly from the species text boxes; so initial values cannot be edited when using *Odefy* with CNA. Reactions with a zero in their textbox are temporarily removed from the network before starting the simulation (see also remark below).
- *Simulation type* - determines whether a HillCube, a normalized HillCube (reaches y=1 at x=1) or a BooleCube transformation to an ODE model is chosen (see [16]). Alternatively, discrete Boolean simulations (with synchronous, asynchronous (with fixed order) or randomly asynchronous update rules) will be performed. Note that the BooleCube simulation ignores the *n* and *k* Hill parameters whereas the Boolean simulation ignores all parameters. For Boolean simulations with synchronous (all nodes fire at once) or randomly asynchronous (one randomly selected node fires) update rules, no parameters are required. For (non-random) asynchronous update (in each time step the state of only one node is updated and after n time steps each of the n nodes has been updated once), you must specify an order in which the species are updated. In *Odefy* you have three options: you can (i) load a permutation from the current workspace, (ii) update the species in sequential order or (iii) generate a random permutation.
  Note: When performing asynchronous Boolean simulation, one time step corresponds to an update step of <u>one</u> species!
- *Plot type*: You can choose “Regular line diagram” or “Heatmap style”.
- *Save results into workspace* - If this box is checked, the simulation time vector and the corresponding concentration matrix will be stored in the MATLAB workspace as `simt`

and `simy`, respectively. If you are performing a Boolean simulation, no time vector is created.

By clicking on *Simulate*, the simulation is started. Results will be displayed in a plot (of the chosen style).
Important: before the actual simulation starts, *Odefy* reads the values from the species and reaction text boxes. Values set in species text boxes are used *as initial values* for the respective species (should be between 0 and 1). Species for which no value is defined will be assigned an initial value of 0. Regarding the reaction text box values: all reactions with a zero value in their text box are (temporarily) removed from the network (non-zero values are ignored). In this way, a knock-out of a node can be simulated by setting all reactions pointing into that node to zero.

4.3.7 Miscellaneous

Export interaction matrix ...: Exports interaction matrix, NOT flags (how interaction matrix and NOT flags are stored, see chapter 7), reactions and species identifiers for further analysis outside of *CellNetAnalyzer*. (Note that the representation of logical networks is currently not supported by SBML and SBML export is thus not possible.)
The user can choose whether the matrix is to be exported as ASCII or MATLAB file. In ASCII format the network is saved as follows:

first row: <reaction names>
second row <name of first species> <first row of interaction matrix>
third row: <name of second species> <second row of interaction matrix>
....<and so on>

If the matrix is stored in MATLAB format, 4 variables are saved which can be reloaded in MATLAB later on:

species_names: names of the m metabolites (char array)
reaction_names: names of the q reactions (char array)
NOTflags: a zero indicates a NOT operation in the respective interaction equation
intermat: interaction matrix (m x q), where the m rows correspond to the species in *species_names* and the q columns to the reactions in *reaction_names*

Show current values in bar chart: Displays currently given reaction and species values in (two) bar charts. Useful for comparisons, e.g. participation levels of reactions in signaling paths

Set original map size: Each network map is resized to its original resolution (pixel-based graphics) or the original size (MATLAB figures). The maps should then appear well-displayed. (Again, for this reason, all graphics should not have a resolution larger than the monitor resolution.)

Zoom tools on/off: Displays the zoom tools in the network map in which this menu item has been selected (or the zoom tools disappear if they are currently visible). The zoom tools are always mounted in the upper right corner of the map. The scrollbars allows one to scroll the map left/right and up/down, respectively (note: sometimes, the scrollbars are not correctly displayed by MATLAB; you should then click somewhere in the map or just minimize and then maximize

the window). The ‘+’ button enables to zoom in. Press the ‘+’ button and then select two points on the map using the cross-hair. The rectangle spanned by these two points will be the new region displayed in the window. The text boxes are also zoomed with the respective factor. By pressing the ‘-’ button one can zoom out (factor of 2). Finally, by pressing the ‘R’ (reset) button the entire map is displayed again. Note that the assignment of text box positions can also be done in the zoomed state.
Pan (from toolbar, only for MATLAB version 7.3 and higher): Can be used to activate the pan mode of the figure. When the pan mode is switched on you can click and drag the network map around inside the window. This will only have an effect if a part of the map is currently displayed (e.g. after zooming in). Due to limitations of the MATLAB GUI the textboxes are hidden while dragging the maps. Pan can be used in conjunction with zoom but then zoom must be activated first. Important: Do not use the MATLAB zoom/pan tools when working with CNA.

Info ...: Shows info about *CellNetAnalyzer*.

5. Editing ASCII files of network projects

In *CellNetAnalyzer* there are four possible ways to define/import the structure of a network (project):

- Use the GUI masks to define your network (as described in section 3.2 and 4.2).
- One may manually create or edit the network-specific files which are read when loading a project in CNA. In some cases, this might be more convenient than using the masks, especially when constructing large networks. This chapter describes how a network project is saved in CNA and which network-specific files are required.
- Mass-flow (stoichiometric) models can be imported and exported in SBML and METATOOL format (detailed in section 3.3 and chapter 6).
- One may use the CNA's Application Programming Interface (API) to construct a new network project (via MATLAB's command line). Existing (possibly partial) information of the network structure (such as the stoichiometric matrix) can be used in the API-based generation of a new project. This solution is described in detail in chapter 7 and is recommended if some parts of the network structure are available/can be loaded into in the MATLAB workspace.

As mentioned, this chapter describes how CNA network projects are stored on disk.

5.1 Declaring a network project

Project-specific files are stored in the associated project directory. All network projects and their associated directories are registered in the file "networks" in the CNA main directory. A new project can be manually registered or an existing one can be modified by editing this file. Each row in this file registers a network project as follows:

```
<Name of the network project>        <directory>    <network type>
```

The directory should be a sub-directory of the CNA main directory and the relative path instead of the absolute one should be given. Network type can be "1" (mass-flow) or "2" (signal-flow). An example, defining a project of a mass-flow network named "Escherichia coli" with project directory "coli":

```
Escherichia;coli          coli      1
```

Semicolons can be used to mark blanks, hence, the String "Escherichia;coli" will be translated to "Escherichia coli".

5.2 Network project files

First, for both mass-flow and signal-flow networks, the project directory should contain the graphics serving as CNA network maps (see chapter 1 and 2). If no network map is available you may use the provided dummy picture "dummy.pcx" located in the CNA main directory.

Several ASCII files stored in the project directory are used by CNA for representing mass-flow (MF) and signal flow (SF) networks. These files are in a special format interpretable by CNA. The first three files (*app_para.m, metabolites, reactions*) are required for both MF and SF:

app_para.m
This file contains general information on the network project, including text box colors, text box size, in which graphic files the network maps are stored etc. This file should always be created by using the Project Manager (e.g. when initializing and registering a new project).

reactions
This file contains the reactions. Each row defines a reaction and its properties (see section 3.1 and 4.1) as follows:

<identifier> <reaction equation> | <default value> <Par1> <Par2> <Par3> <x-Pos> <y-Pos> <map number> <text box type> <Par4>

The (symbolic) reaction (in SF: interaction) equation can be defined as described in section 3.1 and 4.1, respectively. All equations must be finished by the pipe symbol as indicated above. For all reactions, except the biomass synthesis reaction (whose identifier is "mue"), a reaction equation must be given.
<default value>: If the default value is unknown write "#" or "NaN".
Par1: MF: rate minimum; SF: flag for incomplete truth table (0 = off, 1 = on)
Par2: MF: rate maximum; SF: time scale
Par3: MF: coefficient in linear objective function (for optimization problems); SF: flag for "Exclude in logical computations" (1: exclude; 0: include)
<x-Pos> and <y-Pos> are the pixel coordinates of the lower left corner of the text box associated with this reaction. <map number> is the network map where the box has to be appear. <text box type> is "1" for editable, "2" for non-editable and "3" for non-visible. The non-visible type is not allowed in mass-flow networks.
Par4: MF: Variance of measurements; SF: flag for monotony (0 = non-monotone)

Example: reaction R1::A in Fig. 1 (stoichiometric network):
```
R1::A    = 1 A   |     #       -Inf        Inf     0    73.5       97.5          1          1        0.01
```
Example: biomass synthesis ("mue") in Fig. 1 (stoichiometric network):
```
mue     |        #        0       10     0        394.5        448.5          1          1      0.01
```
Example: reaction 2&3 in Fig. 21 (signal-flow network):
```
2&3   !I1 + I2  = E   |  #       0       1      0     653      148          1          1         1
```

metabolites
Stores the metabolites (MF) or species (SF), respectively, including their attributes. Each row defines a metabolite/species with its parameters as described in section 3.1 and 4.1, respectively:

<identifier> <full name> <Par1> <Par2> <x-Pos> <y-Pos> <map number> <textbox type>

Par1: MF: not used; SF: default value (if no default value is known write "#" or "NaN").
Par2: MF: External flag (is "0" if the metabolite is internal and "1" if external); SF: not used
<x-Pos>, <y-Pos>,<map number> and <text box type> define the position and type of the associated text box as explained for reactions. Note that this information is only relevant and used for SF (since only there species have an associated text box).

Example: species A in Fig. 21:

```
A       SpeciesA      NaN   0       269     155     1       1
```

The two files *reactions_notes* and *metabolites_notes* are optional for both MF and SF. They contain comments and notes for the reactions / species defined in the files *reactions* and *metabolites,* respectively. If the respective file does not exist, the comments are set to be empty. The set-up of both files is simple:

```
<identifier1>   <Notes1>
<identifier2>   <Notes2>
.......
```

<identifier> is a identifier of a species/reaction as defined in the *reactions* and *metabolite* files
<Notes> is a string containing the notes/comments for the respective network element. Spaces are allowed in the String but line breaks are not. You may encode a line break by “;:;”. This string will be replaced when displaying the notes in *CellNetAnalyzer*. Note that the IDs need not be in the same order as in the *reactions/metabolites* files and not for all species and reactions a comment has to be included.

The following three files are additionally required (at least as empty files) for mass-flow networks:

macromolecules
Defines the biomass components including their attributes and parameters:

<identifier> <full name> <default value> <x-Pos> <y-Pos> <map number> <text box type>

<x-Pos>, <y-Pos>, <text box type> and <map number> describe the coordinates and style of the associated text box as for reactions. The (cumulative) synthesis equation for each biomass constituent is defined in file *macromolecule_synthesis* (see below).

Example: biomass constituent BC1 in Fig.1:

```
BC1   Biomass_Component_1      0.4     225.5   423.5     1       1
```

macromolecule_synthesis
Each row describes the cumulative synthesis equation for one biomass constituent (see section 3.1). It expresses the amount of metabolites (unit: [mmol Metabolite / g biomass component]) required for the synthesis of 1 gram of this biomass component. The equation can be formulated by using the metabolite identifiers, similar as for reaction equations:

<BiomassConst.> = <coeff1> <Met1> + <coeff2> <Met2> + ... + <coeffx> <Metx>

Example: Synthesis equation for BC1 in Fig.1:
BC1 = 2 A + 1 C

assembly
Each row describes an assembly route and its properties (see section 3.1) as follows:

```
<Metabolite>   <Biomass Constituent>   <x-Pos>  <y-Pos>   <map number>
```

<x-Pos>, <y-Pos> and <map number> define the location of the associated text box as for reactions. <Metabolite> is the identifier of the metabolite consumed for the biomass constituent given in the second identifier.

Example: assembly of A into BC1

```
A        BC1    181.5        331.5            1
```

6. Import of models in SBML and METATOOL format

CellNetAnalyzer supports the import and export of stoichiometric models given in SBML (Level 2) or METATOOL format. From version 9.7, these conversions are now supported via specific API functions which are described in section 7.4. The export of a CNA model to a METATOOL or SBML file can also be done via the respective menu entries as described in section 3.3.6.

7. Application Programming Interface (API)

The API (Application Programming Interface) of CNA allows interested users and developers

- to read/write or import/export the network structure of a project
- to read values from text boxes (GUIs), then to perform own calculations and finally to display the results in the interactive network maps
- to call selected functions of CNA (such as the computation of elementary modes or signaling paths) independently of the CNA GUI
- to integrate own functions as a new menu-entry in CNA's menu – you can thus construct and integrate plugins for CNA
- to change network/project attributes directly from MATLABs command line (only for advanced users; not recommended for beginners)

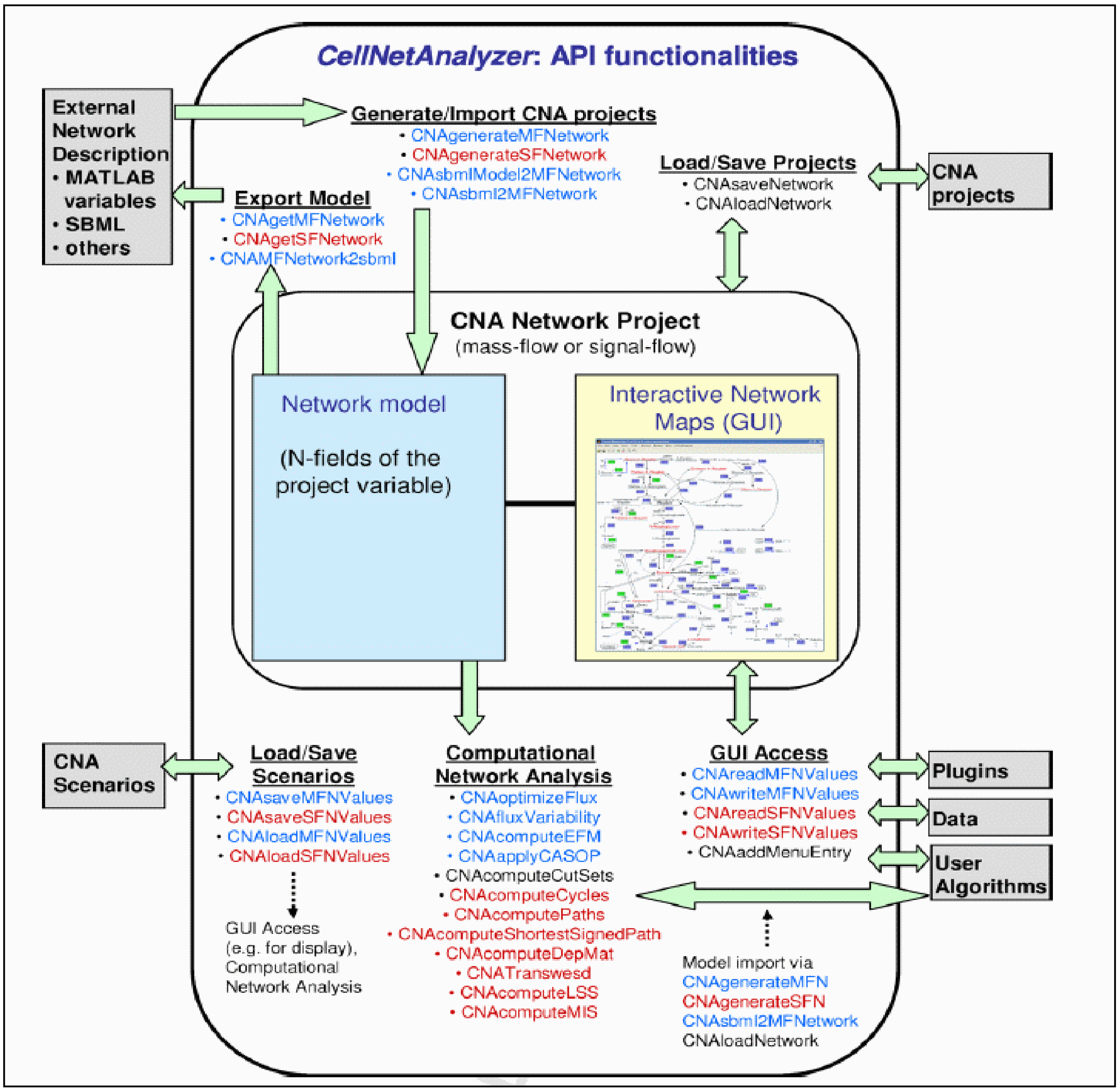


Fig. 23: Overview of API functionalities in CellNetAnalyzer.

CNA comprises several functions that have specific API functionalities (the name of these functions starts with prefix 'CNA') which will be explained below. Furthermore, we will also document the internal variable structure of CNA, which should make interfacing CNA with other applications easier.

Before using any of the functionality presented in this chapter make sure that CNA is initialized. This can be done by calling the **startcna** function (if this function is not available go into the *CellNetAnalyzer* directory first). Calling **startcna** adds all relevant CNA directories to the MATLAB path, initializes the global variable **cnan** and launches the project manager window. You can suppress the project manager (e.g. if you want to operate in batch-mode or if you are using octave) by calling **startcna(1)** (see also below).

7.1. The internal variable structure of CellNetAnalyzer

In previous versions of the CNA (< 9.0) it was only possible to work with one project at a time in a given MATLAB session because all variables belonging to this project were stored in the base workspace. Furthermore, a lot of functionality was realized as scripts that changed or added variables in the base workspace. In order to encapsulate the data belonging to one project and to allow working with multiple projects, (network) project variables were introduced in CNA version 9.0.
Project variables are MATLAB structure arrays (type struct) which contain several fields for the different types of data associated with a project. Moreover, some of the fields are structs themselves to allow grouping of variables that belong together. Basically this data model follows the idea of object orientation as it was introduced in MATLAB in version 7. However, none of MATLAB's object orientation functionality is currently used and therefore features like inheritance do not work for CNA network variables.

As already mentioned, once loaded, each CNA network project is represented in the MATLAB workspace as a struct variable (which we will denote in the following with cnap). In general, the fields of a CNA project struct variable can be roughly classified into
i) **N-fields** (= **n**etwork fields): fields that define mathematically the **n**etwork structure and its attributes (e.g. stoichiometric matrix).
ii) M-**fields** (= **m**eta fields): are fields that are additionally required to define a CNA network project (mainly GUI-related data such as text box size or text box color and other meta data such as the path of the project directory).
iii**) L-fields** (= **l**ocal fields): the field ‘cnap.local’ is itself a struct and its fields are temporary variables that are changed during actions or calculations. In particular, fields of ‘cnap.local’ are used to pass/return parameters to/from functions. For using the API functionality, users do not have to care about these fields.

Several fields are identical for mass-flow and signal-flow networks but there are also specific fields for each network type. In the following we document the N- and P- fields of mass-flow and signal-flow networks. Note that not all of the listed fields need to exist in a project variable. Which ones are necessary for which functionality is documented for each API function.

Mass-flow networks
(we assume in the following that the network has n species, q reactions and m macromolecules; w is a placeholder for an appropriate non-zero number)

N-fields of mass-flow projects (compare with sections 3.1 and 5):

cnap.type: type of the network (cnap.type = 1 for mass-flow and cnap.type=2 for signal-flow)
cnap.specID: ($n \times w$) char array containing the identifiers of the species
cnap.specLongName: ($n \times w$) char array containing the long (full) names of the species
cnap.specNotes: ($1 \times n$) cell array containing the notes for the species
cnap.specExternal: ($1 \times n$) vector indicating which species are external (1) an which not (0)
cnap.specInternal: ($1 \times w$) vector with the indices of the internal species; it holds
that cnap.specInternal=find(~cnap.specExternal);
cnap.nums: number of species in the network (cnap.nums=n)
cnap.numis: number of internal species (cnap.numis=size(cnap.specInternal,2))
cnap.numr: number of reactions in the network (cnap.numr=q)
cnap.reacID: ($q\times w$) char array containing the identifiers of the reactions
cnap.reacNotes: ($1 \times q$) cell array containing the notes for the reactions
cnap.mue: is the index of the 'mue' (biosynthesis) reaction; if 'mue' is not contained in the reacID list, then cnap.mue is empty
cnap.objFunc: ($q \times 1$) vector containing the coefficients of the reactions in the linear objective function (to be used for "Flux optimization")
cnap.reacMin: ($q \times 1$) vector containing the lower boundaries of the reaction rates (e.g. 0 for an irreversible reactions or –Inf for irreversible reactions)
cnap.reacMax: ($q \times 1$) vector containing the upper boundaries of the reaction rates (e.g. Inf for arbitrary large rates)
cnap.reacDefault: ($q \times 1$) vector containing the default (reaction rate) values to be displayed in the reaction text boxes (NaN indicates an empty value)
cnap.reacVariance: ($q \times 1$) vector containing the variances of (measured) reaction rates (used for some metabolic flux analyses; see section 3.3)
cnap.stoichMat: the ($n \times q$) stoichiometric matrix of the network
cnap.macroID: ($m \times w$) char array containing the identifiers of the macromolecules
cnap.macroLongName: ($m \times w$) char array containing the full names of the macromolecules
cnap.macroDefault: ($m \times 1$) vector containing the default values of the macromolecules (to be displayed in the macromolecule text boxes; NaN is not allowed here)
cnap.nummac: number of macromolecules (cnap.nummac=m)
cnap. macroComposition: ($m \times w$) matrix defining the stoichiometry of the macromolecules with respect to the metabolites (species): matrix element macroComposition(i,j) stores how much of metabolite i (in mmol) is required to synthesize 1 gram of macromolecule j

M-fields of mass-flow projects (compare with section 2, 3.1 and 5):

cnap.has_gui: flag that indicates whether this project variable is associated with an active GUI
cnap.netnum: index of the network in the project manager.
cnap.path: gives the (relative or absolute) path of the project directory (in which the project files are located
cnap.net_var_name: the name of the network project (identical with the name of the project variable; here denoted with the placeholder 'cnap':) This field contains thus the name of the variable as it is called in the base workspace. When loading a network a name for the network variable is automatically generated based on the directory name of the network folder. It is essential that net_var_name matches the name of the workspace network variable name for the proper operation of the graphical user interface (see technical aspects). This has to be taken into account if you are running a project with a GUI and want to modify its data from

the command prompt. Neither the variable name nor the string in *cnap.net_var_name* must be changed while the network is used because this would result in errors when using the GUI elements.

cnap.unsaved_changes: indicates whether there are unsaved changes in the network structure (1) or not (0) (made by using the network composer)

cnap.epsilon: CNA epsilon (smallest number greater than zero) to be used in CNA calculations

cnap.reacBoxes: a (q $\times$ 6) array defining the position and appearance of the reaction text boxes; cnap.reacBoxes(:,1) not in use currently; cnap.reacBoxes(:,2) defines the x-position of each box; cnap.reacBoxes(:,3) defines the y-position, cnap.reacBoxes(:,4) contains the MATLAB GUI handles of the text boxes, cnap.reacBoxes(:,5) contains the map numbers (where the boxes are to be displayed) and cnap.reacBoxes(:6) defines the style of each text box (1=editable; 2=non-editable)

cnap.macroBoxes: a (m $\times$ 6) array defining the position and appearance of the macromolecule text boxes; the columns of this array are analogous as for cnap.reacBoxes

cnap.macroSynthBoxes: a (w $\times$ 6) array defining the assembly routes including the parameters of their text boxes (w = number of defined assembly routes): cnap.macroSynthBoxes(:,1) the index of the metabolite involved in this assembly route, cnap. macroSynthBoxes(:,2) the index of the macromolecule in whose assembly the metabolite is involved; macroSynthBoxes(:,3) defines the x-position of the text boxes of the assembly routes; cnap. macroSynthBoxes(:,4) defines the y-position of the text boxes; cnap. macroSynthBoxes(:,5) contains the MATLAB GUI handles of the text boxes; cnap. macroSynthBoxes(:,6) stores the map numbers (where the boxes are to be displayed)

cnapnummacsynth: number of assembly routes (= size(cnap.macroSynthBoxes,1))

cnap.maps: is a (w $\times$ 2) cell array of strings (w = number of interactive maps in the project). cnap.maps{i,1} is the name (title) of the i-th map; cnap.maps{i,2} is the graphic file (optionally with path) of the i-th map.

cnap.nummaps: is the number of interactive maps in this project (cnap.nummaps=size(cnap.maps,1)

cnap.figs: is a (w $\times$ 9) array (w = number of interactive maps) defining some figure properties: cnap.figs(:,1) stores the handles of the figures; cnap.figs(:,2:3) stores the resolution of the figure; cnap.figs(:,4) indicates whether the resource (graphic file) of the figure is a pixel-based graphic (0) or a MATLAB figure (1); cnap.figs(:,5:9) stores (temporarily) some parameters required for installing zoom tools on the maps

cnap.color1: (1 $\times$ 3) vector with the RGB values defining reaction text box color ‘standard’

cnap.color2: (1 $\times$ 3) vector with the RGB values defining reaction text box color ‘calculated’

cnap.color3: (1 $\times$ 3) vector with the RGB values defining reaction text box color ‘special’

cnap.color4: (1 $\times$ 3) vector with the RGB values defining reaction text box color ‘defined’

cnap.specBoxColor: (1 $\times$ 3) vector with the RGB values of the color of macromolecule text boxes

cnap.textColor: (1 $\times$ 3) vector with the RGB values defining the text color in reaction and macromolecule text boxes

cnap.macroSynthColor: (1 $\times$ 3) vector with the RGB values defining the text color in assembly text boxes

cnap.reacBoxWidth: (1 $\times$ w) vector (w = number of interactive maps) defining the width of reaction text boxes (separately for each map)

cnap.reacBoxHeight: (1 $\times$ w) vector (w = number of interactive maps) defining the height of reaction text boxes (separately for each map)

cnap.specBoxWidth: (1 $\times$ w) vector (w = number of interactive maps) defining the width of macromolecule text boxes (separately for each map)

cnap.specBoxHeight: (1 × w) vector (w = number of interactive maps) defining the height of macromolecule text boxes (separately for each map)
cnap.reacFontSize: (1 × w) vector (w = number of interactive maps) defining the font size of the text in the reaction text boxes (separately for each map)
cnap.specFontSize: (1 × w) vector (w = number of interactive maps) defining the font size of the text in the macromolecule text boxes (separately for each map)
cnap.has_gui: flag that indicates whether this project variable is associated with an active GUI
cnap.pan_icon_handles (only version 7.3 and higher): stores GUI handles
cnap.figmenu: stores GUI handles

Signal-flow networks

(we assume in the following that the network has n species, q reactions/interactions; w is a placeholder for an appropriate non-zero number)

N-fields of signal-flow projects (compare with sections 4.1 and 5):

cnap.type: type of the network (cnap.type = 2 for signal-flow)
cnap.nums: number of species in the network (cnap.nums=n)
cnap.numr: number of reactions in the network (cnap.numr=q)
cnap.specID: (n × w) char array containing the identifiers of the species
cnap.specLongName: (n × w) char array containing the long (full) names of the species
cnap.specNotes: (1 × n) cell array containing the notes for the species
cnap.specDefault: (n × 1) vector containing the default values to be displayed in the species text boxes (NaN indicates an "empty")
cnap.reacID: (q× w) char array containing the identifiers of the interactions
cnap.reacNotes: (1 × q) cell array containing the notes for the interactions
cnap.reacDefault: (q × 1) vector containing the default values to be displayed in the interaction text boxes (NaN indicates an empty value)
cnap.monotony: (1 × q) vector containing the monotony flag for the interactions (0 if not monotone; non-zero if monotone)
cnap.excludeInLogical: (1 × q) vector containing the "exclude in logical calculations" flag of the interactions (1 if to be excluded; 0 otherwise)
cnap.incTruthTable: (1 × q) vector containing the "incomplete truth table" flag of the interactions (1 for incomplete truth table, 0 otherwise)
cnap.timeScale: (q × 1) vector containing the time scales of the interactions
cnap.interMat: (n × q) interaction matrix of the network (rows: species; columns: interactions); defines together with cnap.notMat the logic of each interaction (each interaction is an AND clause; see also section 4.1 and reference [11])
cnap.notMat: (n × q) matrix (rows: species; columns: interactions); stores the NOTs appearing in the interactions; if species i participates with its inverted value (by a NOT operation) in the j-th interaction then cnap.notMat(i,j) = 0 (otherwise 1)

M-fields of signal-flow projects (compare with section 2, 4.1 and 5):

cnap.has_gui: flag that indicates whether this project variable is associated with an active GUI
cnap.netnum: index of the network in the project manager.
cnap.path: gives the (relative or absolute) path of the project directory (in which the project files

are located

cnap.net_var_name: the name of the network project (identical with the name of the project variable; here denoted with the placeholder ‘cnap’:) This field contains thus the name of the variable as it is called in the base workspace. When loading a network a name for the network variable is automatically generated based on the directory name of the network folder. It is essential that net_var_name matches the name of the workspace network variable name for the proper operation of the graphical user interface (see technical aspects). This has to be taken into account if you are running a project with a GUI and want to modify its data from the command prompt.

cnap.unsaved_changes: indicates whether there are unsaved changes in the network structure (1) or not (0) (made by using the network composer)

cnap.epsilon: CNA epsilon (smallest number greater than zero) to be used in CNA calculations

cnap.reacBoxes: a (q $\times$ 6) array defining the position and appearance of the reaction text boxes; cnap.reacBoxes(:,1) not in use currently; cnap.reacBoxes(:,2) defines the x-position of each box; cnap.reacBoxes(:,3) defines the y-position, cnap.reacBoxes(:,4) contains the MATLAB GUI handles of the text boxes, cnap.reacBoxes(:,5) contains the map numbers (where the boxes are to be displayed) and cnap.reacBoxes(:6) defines the style of each text box (1=editable; 2=non-editable)

cnap.specBoxes: a (n $\times$ 6) array defining the position and appearance of the species text boxes; the columns of this array are analogous as for cnap.reacBoxes

cnap.maps: is a (w $\times$ 2) cell array of strings (w = number of interactive maps in the project). cnap.maps{i,1} is the name (title) of the i-th map; cnap.maps{i,2} is the graphic file (optionally with path) of the i-th map.

cnap.nummaps: is the number of interactive maps in this project (cnap.nummaps=size(cnap.maps,1)

cnap.figs: is a (w $\times$ 9) array (w = number of interactive maps) defining some figure properties: cnap.figs(:,1) stores the handles of the figures; cnap.figs(:,2:3) stores the resolution of the figure; cnap.figs(:,4) indicates whether the resource (graphic file) of the figure is a pixel-based graphic (0) or a MATLAB figure (1); cnap.figs(:,5:9) stores (temporarily) some parameters required for installing zoom tools on the maps

cnap.color1: (1 $\times$ 3) vector with the RGB values defining text box color ‘standard’

cnap.color2: (1 $\times$ 3) vector with the RGB values defining text box color ‘calculated’

cnap.color3: (1 $\times$ 3) vector with the RGB values defining text box color ‘special’

cnap.color4: (1 $\times$ 3) vector with the RGB values defining text box color ‘defined’

cnap.specBoxColor: (1 $\times$ 3) vector with the RGB values defining the color of species text boxes

cnap.textColor: (1 $\times$ 3) vector containing the RGB values of the text color to be used for both species and reaction text boxes

cnap.macroSynthColor: (1 $\times$ 3) vector: (currently unused in signal-flow networks)

cnap.reacBoxWidth: (1 $\times$ w) vector (w = number of interactive maps) defining the width of reaction text boxes (separately for each map)

cnap.reacBoxHeight: (1 $\times$ w) vector (w = number of interactive maps) defining the height of reaction text boxes (separately for each map)

cnap.specBoxWidth: (1 $\times$ w) vector (w = number of interactive maps) defining the width of species text boxes (separately for each map)

cnap.specBoxHeight: (1 $\times$ w) vector (w = number of interactive maps) defining the height of species text boxes (separately for each map)

cnap.reacFontSize: (1 $\times$ w) vector (w = number of interactive maps) defining the font size of the text in the reaction text boxes (separately for each map)

cnap.specFontSize: $(1 \times w)$ vector (w = number of interactive maps) defining the font size of the text in the species text boxes (separately for each map)
cnap.pan_icon_handles (only version 7.3 and higher): stores GUI handles
cnap.figmenu: stores GUI handles

Knowing the structure of the CNA project variable, you may easily export selected data from a project (e.g. the stoichiometric Matrix). You may also manually change some entries, however, it is then up to you to ensure that the network remains consistent. For example, if you add a reaction (manually) to the stoichiometric matrix, you also have to update all fields that contain reaction specific information (such as reacID, reacMin, reacMax etc.).

7.2. API functions for creating and copying network projects

CNAgenerateMFNetwork

Assume that you have a stoichiometric reaction network that you would like to analyze with CNA (or some CNA functionalities). Usually, you will have the stoichiometric matrix and the reversibilities of the reactions. However, a CNA mass flow project requires many more variables (fields) to be defined. Here, it would be desirable to have a function, which generates a CNA mass flow project from such partial information. This is exactly one purpose of the API function

Usage: [cnap, errval] = **CNAgenerateMFNetwork**(cnap)

The project variable ‘cnap’ is usually a structure that contains some (N-)fields of a CNA mass-flow project whose existing fields are checked for consistency and to which missing fields are added and initialized with default values. The output errval indicates whether consistency errors have been found (errval=1) or not (errval=0).
If one calls **CNAgenerateMFNetwork** and passes cnap without any N-field name (or without the field cnap.stoichMat), an empty CNA mass-flow project structure will be generated and initialized. If cnap already has some N-fields of a CNA mass-flow project, all missing N-fields will be generated and filled with default values and then returned. Additionally, the M-fields reacBoxes, macroBoxes, nummacsynth, macroSynthBoxes and epsilon will be appended with default values as well. For example, the text box position of reactions (cnap.reacBoxes) and macromolecules (cnap.macroBoxes) will be set to the upper left corner. These M-fields are required to save the project using **CNAsaveNetwork** (see below). Upon saving the network, the project can later be registered via the project manager where also all other required M-fields (text box size and style etc.) can then be defined.
Even though the returned project variable ‘cnap’ lacks most M-fields, it can already be used for calling several CNA functions (see below) which do not access GUI elements.
Importantly, if the passed argument cnap contains already several N-fields (e.g. cnap.stoichMat, cnap.nums, cnap.numr) this function will check the consistency of the predefined fields and return a nonzero errval if an inconsistency is found. The procedure will also be stopped at this point, so the creation of the network variable will then be incomplete.

Here is an example: assume you have (only) the stoichiometric matrix of a network in the MATLAB workspace. Let's call this matrix N. Create now a new structure z having the (only) field "stoichMat":
```
    z.stoichMat=N;
```
Entering now
```
        [z, errval] = CNAgenerateMFNetwork(z);
```
will create a CNA mass-flow project structure in the variable ‘z’ containing all N-fields and the few M-fields mentioned above. Non-existing fields are initialized with default values, for example z.reacMin, z.objFunc will be zero vectors, reacMax an Infinity vector. With this variable you can already call functions that do not access the GUI (see below).You may then save the generated network as a CNA project with **CNAsaveNetwork**(see below)
```
    z.path= ‘path/to/project/directory’;
    z=CNAsaveNetwork(z);
```
Note that it is necessary to specify the save directory in z.path before calling **CNAsaveNetwork**.
If you then register the project in the project manager you will automatically set the rest of the M-fields of this

project (including a network map, if you want). You can then reload the project later either with the "Load w/o GUI" button or (if you provided a network map) as usual with ‘Start’.
Note that in the above example **CNAgenerateMFNetwork** will first check whether z has a field "stoichMat". If not, an empty field stoichMat will be added to z and all other required fields will be initialized accordingly. As mentioned, the correct dimensions of existing fields will be checked. For example, if your input variable z has the two fields "stoichMat" and "reacMin" and if size(cnap.stoichMat,2)~=size(cnap.reacMin,1) you will get an error message and (errval will be 1). The function will then stop and the project variable remains incomplete.

Another Example
Here is another, more explicit example how to use **CNAgenerateMFNetwork** to create a new metabolic network model.
If not done yet, start CNA by calling **startcna** (we here do that without launching the project manager; see section 7.8):

    startcna(1);

Consider the following reaction scheme:
R1: A = 2 B + C
R2: B + D = E
R3: C + E = 3 F
which gives the stoichiometric matrix (rows: metabolites, columns: reactions):

```
simple_net.stoichMat= [-1 0 0; 2 -1 0; 1 0 -1; 0 -1 0; 0 1 -1; 0 0 3];
```

The metabolites B, C, E are considered internal:

```
simple_net.specInternal= [2 3 5];
```

If all reactions are irreversible and you don't want to assign your own metabolite or reaction names you can already call

```
simple_net= CNAgenerateMFNetwork(simple_net);
```

to set up the project variable. If you want to assign e.g. metabolite names you can use

```
simple_net.specID= strvcat('A', 'B', 'C', 'D', 'E', 'F');
```

to do so (assigning reacID works analogously). Reaction reversibility can be controlled with the reacMin and reacMax fields. Typically all reactions have +Inf as reacMax and the distinction between reversible and irreversible reactions is made by setting reacMin to -Inf or 0 respectively. In order to make reactions two and three reversible you can e.g. execute

```
simple_net.reacMin= zeros(size(simple_net.stoichMat, 2), 1);
simple_net.reacMin([2 3])= -Inf;
```

You can now call

```
simple_net= CNAgenerateMFNetwork(simple_net);
```

which sets reacMax to +Inf for all reactions and configures the remaining fields of the project variable.
To save the network create a new directory (here called simple_path like the project variable), set up the path field to point to it and lastly call **CNAsaveNetwork** (as explained in section 7.3):

```
simple_net.path= './simple_net';
simple_net= CNAsaveNetwork(simple_net);
```

You can later register the network in the project manager (cf. chapter 2 and 7.3 below) and now continue working with the project through the API functions described below. When you are finished, you can call

```
close_cna;
```

which closes all projects running with a GUI, removes the CNA directories from the MATLAB path and deletes the **cnan** variable (section 7.9).

CNAgenerateSFNetwork

Analogous as for mass-flow projects, there is also a function

Usage: [cnap, errval] = **CNAgenerateSFNetwork**(cnap)

for creating a signal-flow project (variable) from partial information. Again, the fields in ‘cnap’ will be checked for consistency and all missing N-fields and the M-fields reacBoxes, specBoxes and epsilon will be added. Here, **CNAgenerateSFNetwork** starts with looking whether the field ‘cnap.interMat’ exists. If not, an empty project will be created, otherwise, all other signal-flow N-fields will be added (if they do not exist yet) or checked for consistency (if already present in cnap).

An example: assume you have (only) the interaction matrix and the NOT matrix of a Boolean network somewhere in the MATLAB workspace (for the definition of the interaction and NOT matrix see reference [11]). Let's call these matrices I and N, respectively. Create now a new structure z having the fields interMat and notMat:

```
z.interMat=I;
z.notMat=N;
```

Entering now

```
[z,errval] = CNAgenerateSFNetwork(z);
```

will create a CNA signal-flow project variable ‘z’ that has all N-fields (defining the network topology) of a signal-flow network and the M-fields reacBoxes, specBoxes and epsilon. Non-existing fields are initialized with default values. With this variable you can already call functions that do not access the GUI (see below). You may save the generated network project as a CNA SFN project via **CNAsaveNetwork** (see below):

```
z.path= ‘path/to/project/directory’;
znew=CNAsaveNetwork(znew);
```

Note that it is necessary to specify the save diretory in z.path before calling **CNAsaveNetwork**.
If you then register then the project in the project manager (or by editing the file "networks") you will automatically set the rest of the M-fields of this project (including a network map, if you want). You can then relaod the project later either with the "Load w/o GUI" button or (if you provided a network map) as usual with ‘Start’.
Again, the correct dimensions of existing fields will be checked. For example, if any(size(z.interMat)~=size(z.notMat)) you will get an error message and errval will be 1.

The following two functions facilitate to export/copy a CNA network project into a new struct variable. Only the (N-) fields defining the network structure are copied whereas all GUI-related fields are not transferred.

CNAgetMFNetwork

Usage: mfn = **CNAgetMFNetwork**(cnap,biocomp);

cnap is a CNA mass-flow project structure and biocomp is a (cnap.nummac,1) vector defining the biomass composition. If macromolecules are not defined for this project set biocomp=[].
All those fields of the project cnap will be copied into the output mfn that define the network topology and related parameters. GUI-related (P-) and temporary (L-) fields are not copied. Hence, mfn will contain all N-fields of cnap, except epsilon, has_gui, reacBoxes, macroBoxes, macroSynthBoxes and nummacsynth.
If the project cnap has macromolecules (and uses thus CNA's biomass synthesis reaction 'mue'), the vector biocomp will be used together with cnap.macroComposition to calculate the stoichiometry of reaction 'mue' which is then inserted into the column with index cnap.mue.
This function provides thus a convenient way to copy/export the network structure of a CNA mass-flow project into a new variable, thereby getting rid of GUI and temporary variables of the project. The new struct variable can then be used independently of CNA.

CNAgetSFNetwork

Usage: sfn = **CNAgetSFNetwork**(cnap);

cnap is a CNA signal-flow project structure. All those fields of the project cnap will be copied into the output sfn that define the network topology and related parameters. GUI-related (P-) and temporary (L-) fields are not copied. Hence, sfn will contain all N-fields of cnap, except epsilon, has_gui, reacBoxes and specBoxes.
This function provides thus a convenient way to copy/export the network structure of a CNA signal-flow project into a new variable, thereby getting rid of GUI and temporary variables of the project. The new struct variable can then be used independently of CNA.

7.3. Saving and loading CNA projects

CNA saves a network project via several ASCII files (section 5.2). The following functions can be used to save and load network projects to/from these project files.

CNAsaveNetwork

Usage: cnap = **CNAsaveNetwork**(cnap);

This function stores the network structure of a mass-flow project variable cnap. It creates/overwrites all the files of a CNA network project (see section 5.2.) except app_para.m. cnap must have all N-fields of a CNA mass-flow project and additionally the M-fields cnap.reacBoxes, cnap.macroBoxes, cnap.nummacsynth and cnap.macroSynthBoxes. For example, a CNA project variable created with **CNAgenerateMFNetwork** or **CNAgenerateSFNetwork** can directly be saved with this function. Note that the cnap.path field must be set to an appropriate directory (absolute or relative path) where the project files will be saved. Existing project files will be overwritten! If the directory does not exist then it will be created. The function returns the mass-flow project.

Once you have saved a network project, you can register it in the CNA network list as follows (cf. end of section 2):
1) Click on "New" in the "Network Project Manager".
2) Enter the (preferably relative) path to the project directory where the project files have been stored. Define the name of the project. Additionally, you may declare source files for the network maps – if none is available choose the “dummy” network map as inserted by default.
3) Click on "Save" and answer the next question with "Yes".
4) The project now appears at the bottom of the project list in the project manager.

A saved network project can also be reloaded via **CNAloadNetwork** described below. However, unless the network has been registered in the CNA network list it can only be loaded without GUI (see *no_gui* option below). The reason for this is that registration creates an app_para.m file, which contains GUI configuration options, in the project directory. If a suitable app_para.m file (e.g. taken from some other directory) is added to a project directory, then its project can also be loaded with GUI without having been registered in the project manager.

CNAloadNetwork

Usage: result = **CNAloadNetwork**(network, no_gui, return_cnap)

This function loads a CNA (mass-flow or signal-flow) network project with or without GUI.

Arguments:
- network (mandatory): declares the network to be loaded
  - if isempty(network) == true then the network selected by the project manager is chosen (which must have been loaded);
  - if network is a number then the corresponding network as specified in the cnan.net list (based on the 'networks' file) is loaded.
  - network can also be a cell array with {directory, type} specifying the network to be loaded: directory: string specifying the directory (relative or absolute path), type: 1=mass-flow; 2=signal-flow
- no_gui (optional, default: false): whether the project is to be loaded with or without GUI. (Note that networks which have been generated with **CNAgenerateS(M)FNetworks** and saved with **CNAsaveNetwork** without registering it via the project manager cannot be loaded with GUI).
- return_cnap (optional, default: false): whether a CNA project variable should be returned

It is assumed that cnan has been set up appropriately, e.g. by calling startcna.

Results:

- result: The result depends on return_cnap which can only be set to true if no_gui is true; if return_cnap is true then a CNA project variable (of type mass-flow or signal-flow) is returned, otherwise it is instantiated in the base workspace with the variable name based on the directory name of the project. In the first case a value of cnap.local.errval > 0 indicates an error. In the latter case the variable name of the new project variable is returned as string in case of success or an empty string in case of failure.

7.4 Import and Export of SBML and METATOOL Models

The following API functions serve for import/export of CNA mass flow projects from/to SBML and METATOOL. (Note that the representation of logical/Boolean networks is currently not supported by SBML, hence, SBML import for signal-flow networks is not possible.) Import functions create a CNA mass flow project which can afterwards be saved via **CNAsaveNetwork** and then, optionally, be registered in the project manager's list of networks (see above)

We start with import / export of METATOOL models.

CNAmetatool2MFNetwork

Usage: [cnap, errval] = **CNAmetatool2MFNetwork**(fname)

For importing stoichiometric networks defined in METATOOL format. This function takes the name of the input file as argument and uses a parser for METATOOL input files together with the **CNAgenerateMFNetwork** function (see above) to create a new project variable which is then returned by the function. The new project variable can then be used with the CNA API functions or saved as a CNA project (via **CNAsaveNetwork**) and finally be registered as a new project (as described in section 7.3).

Arguments:

- fname is the filename of the Metatool input file that is to be converted. if no file name is specified, a dialog box occurs.

Result:

- cnap is a mass-flow project variable (which can afterwards be saved via **CNAsaveNetwork**)
- errval indicates whether some error occured during conversion (1: error; 0: no error).

CNAMFNetwork2metatool

Usage: err = CNAMFNetwork2metatool(cnap,fname,macromol)

Exports a CNA mass-flow project to a METATOOL-compatible file.

Arguments:

- cnap (mandatory): a CNA mass-flow project
- fname (optional): the name of the METATOOL file to be generated; if fname is not defined or empty a dialog box will appear
- macromol (optional): vector containing the macromolecule values (concentrations); can be empty when cnap.mue or cnap.macroComposition is empty (default: cnap.macroDefault)

Result:

- err: whether an error occurred (err>0) or not (err=0).

There are two methods for importing an SBML model: The first ‘hand-made’ method, **CNAsbml2MFNetwork**, reads and parses the SBML file directly but because it does not use a professional XML parser it may not always work correctly. Alternatively you can first use the SBML toolbox (http://www.sbml.org/Software/SBMLToolbox) for converting an SBML file into a MATLAB struct which can then be converted to a CNA MFN project variable via **CNAsbmlModel2MFNetwork**.

CNAsbml2MFNetwork

Usage: [cnap, errval] = **CNAsbml2MFNetwork**(fname)

Arguments:

- fname is the filename of the SBML source file that is to be converted. if no file name is specified, a dialog box occurs.

Results:

- the mass-flow project variable cnap (which can afterwards be saved via **CNAsaveNetwork**)
- errval indicating whether some error occured during conversion (1: error; 0: no error).

Important note: When entering the command, after a short moment, the user may select a compartment which serves in the SBML model as container for (exclusively) external metabolites. [Note: *Usually, the flag “boundaryCondition” can be used in SBML to indicate that a metabolite is external. However, to my own experience, it seems that quite often the external metabolites are put instead into a compartment "extracellular" (or similar) to indicate that these species are (at least for stoichiometric studies) external. Therefore, the user may select here one of the compartments which do possibly serve as the container for external metabolites in the SBML model. During the conversion, a species is then considered to be external if its “boundaryCondition” is true or if the species is in the "external" compartment. Thus, in case the external metabolites are indicated by their “boundaryCondition” (as it should be) or if even no external metabolite exists, it is not mandatory to define the “extracellular” compartment*.]

This function works in concert with **CNAgenerateMFNetwork**. When saving the generated CNA project via **CNAsaveNetwork** and registering it afterwards in the project manager (see section 7.3), the project will be loadable from the project manager. However, all textboxes are put on the upper left corner of the network map (or dummy picture, respectively) by default. You may either sort and move the text boxes now by using the “Move text boxes” functionality from the network composer or you just take it as it is and start the network analysis (see also the Note before section 2.2).

Concerning the SBML format, please note that **CNAsbml2MFNetwork** can only read SBML files that comply with the following rules

- name and id fields should not have spaces in it (otherwise they are ignored), besides, *CellNetAnalyzer* is NOT case-sensitive.
- new tags like <listOfSpecies> should start in a new line.
- stoichiometryMath is not supported.

It seems, that these requirements are fulfilled in most SBML files, although deviations are in general allowed.

However If you encounter any problem with **CNAsbml2MFNetwork**, install the SBML toolbox (http://www.sbml.org/Software/SBMLToolbox) and use the *TranslateSBML* function to convert the SBML file into a SBMLModel MATLAB struct. Use this result as input to the **CNAsbmlModel2MFNetwork** function:

CNAsbmlModel2MFNetwork

Usage: [cnap, errval]= CNAsbmlModel2MFNetwork(SBMLModel, ext_comparts)

This function converts a MATLAB SBMLModel (obtained from the TranslateSBML function of the SBML toolbox) into a CNA mass-flow network (MFN) project.

Arguments:

- SBMLModel: a strucutre containing a SBML model that was created by reading a SBML file using the SBML Toolbox (http://sbml.org/Software/SBMLToolbox) it is assumed that the SBMLModel is free of errors (you can use the validate functionality of the SBML Toolbox to check this)
- ext_comparts (optional): Usually, all metabolites declared with *boundaryCondition="true"* will be set to external. In addition ext_comparts can be specified as a cell array of compartment ids (strings) - the metabolites which are located in the listed compartments are considered as external. If *ext_comparts* is not provided the user is automatically queried at the command line for each compartment whether its metabolites should also be considered as external or not.

Results:

- cnap: a CNA (mass-flow) project variable which can be saved via **CNAsaveNetwork**
- errval: 0 if conversion was successful, nonzero if an error has occured

The following function exports a CNA MFN project to an SBML model and saves it in a xml file:

CNAMFNetwork2sbml

Usage: err = **CNAMFNetwork2sbml**(cnap,fname,macromol)

Exports CNA mass-flow project to an SBML file (with path specified in fname).

Arguments:

- cnap (mandatory): a CNA mass-flow project
- fname (optional): the name of the SBML file to be generated; if fname is not defined or empty a dialog box will appear
- macromol (optional): vector containing the macromolecule values (concentrations); can be empty when cnap.mue or cnap.macroComposition are empty (default: cnap.macroDefault)

Results:

- err: whether an error occurred (err>0) or not (err=0).

7.5 Accessing the GUI of CNA

Most API functions of CNA do not access (and do not explicitly need) the GUI of CNA, i.e. they access only N-fields of a CNA project variable. The API functions described in this section, allow you to access the GUI (i.e. the interactive maps and text boxes) of a network project loaded with GUI. CNA itself uses internally different functions for doing the same, however, the API functions described in this section simplify this procedure for the user.

The following four functions allow you to read/write numerical values from/into the text boxes of a network project.

CNAreadMFNValues

Usage: [reacval, macroval**] = CNAreadMFNValues**(cnap)**;**

Given a mass-flow project that has been loaded with GUI (i.e. all N- and M-fields must exist in cnap), this function returns the numerical values currently set in the text boxes of reactions (q × 1 vector reacval) and macromolecules (m × 1 vector macroval), respectively. If the text box of the *i*-th reaction does not contain a numerical value, then reacval(i)=NaN. Nothing is changed in the project variable cnap; it is therefore not returned.

CNAwriteMFNValues

Usage: cnap = **CNAwriteMFNValues**(cnap,reacval,reaccol,macroval);

Given a mass-flow project that has been loaded with GUI (all N- and M-fields must exist in cnap), this function writes numerical values into the text boxes. The q×1 vector **reacval** contains the values to be written into the reaction text boxes, i.e. reacval(i) appears in the text box of the i-th reaction. An unknown/undefined value is indicated by value NaN which will then be represented in the box as '###'. The *q*×1 vector reaccol specifies the color of each reaction text box. Each element in reaccol is an integer value from the set {1,2,3,4} defining the color index of the associated reaction text box. The indices have the following meaning (for colors in CNA see chapter 2): 1="standard", 2="calculated", 3="defined", 4="special".
The *m* × 1 vector macroval contains the numerical values to be written in the text boxes of the macromolecules. They usually define the biomass composition, i.e. the percentages of the biomass constituents given by g macromolecule/gDW (see chapter 3). If you don't have macromolecules in your model simply use an empty matrix [ ]. The color of macromolecule text boxes is fixed and can thus not be changed. The function returns the network project with a few updated L-fields. For example, the values displayed in the text boxes before using this function are saved so that they can be recalled by the 'Reset last scenario' function.

There are two analogous functions for reading/writing text box values in signal-flow networks:

CNAreadSFNValues

Usage: [reacval, specval]=**CNAreadSFNValues**(cnap);

Given a signal-flow project that has been loaded with GUI (i.e. all signal-flow N- and M-fields must exist in cnap), this function returns the numerical values currently set in the text boxes of reactions (q × 1 vector reacval) and species (n × 1 vector specval), respectively. If the text box of the *i*-th reaction does not contain a numerical value, then reacval(i)=NaN; the same is done for species. Nothing is changed in the project variable cnap; it is therefore not returned.

CNAwriteSFNValues

Usage: cnap = **CNAwriteSFNValues**(cnap,reacval,reaccol,specval,speccol);

Given a signal-flow project that has been loaded with GUI (all signal-flow N- and M-fields must exist in cnap), this function writes numerical values into the reaction and species text boxes. The q×1 vector reacval contains the values to be written into the reaction text boxes, i.e. reacval(i) appears in the text box of the i-th reaction. Analogously, the n×1 vector specval contains the values to be written into the species text boxes, i.e. specval(i) appears in the text box of the i-th species. An unknown/undefined value can be indicated by value NaN which will then be represented in the resepctive reaction/species box as '###'. The text box colors are specified by the q×1 vector reaccol (reaction text boxes) and by the m×1 vector speccol (species text boxes). Each element in reaccol and speccol is an integer value from the set {1,2,3,4} defining the color index of the associated reaction/species text box. The indices have

the following meaning: 1=”standard”, 2=”calculated”, 3=”defined”, 4=”special“. Note that “standard” is different for species and reactions; the other three color indices refer to the same color (see also chapter 2).
The function returns the network project with a few updated L-fields. For example, the values displayed in the text boxes before using this function are saved so that they can be recalled by the ‘Reset last scenario’ function.

The following function can be applied to both types of projects and can be used to integrate own entries (and callbacks) into the CNA pull-down menu.

CNAaddMenuEntry

Usage: menhandles = **CNAaddMenuEntry**(cnap,mname,fname);

cnap is a signal- or mass-flow project variable. This function adds a new entry to the CNA pulldown menu. mname and fname are strings[1] defining the name of the menu entry and the associated callback function, respectively. The added menu item will read:
“User function: <mname>”
and is appended to the CNA menu in all interactive maps associated with the project. If the inserted menu entry is selected, the (user-created) function fname will be called. fname will usually correspond to a m- or MEX-file. Note that the extensions (.m, .dll, .mexglx and so on) must not be contained in fname.
The function returns the handles of the created menu entries (if your network project consists of w network maps, then w handles will be returned). The handles may be used to delete inserted menu entries at a later stage. Arbitrary menu entries can be inserted by the user. The project variable cnap is not returned as it remains unchanged.

Example
The following example illustrates the usage of the API functions described in this section and also how to access fields of a CNA project structure. Assume you have a mass flow network in which you have computed a flux distribution. You might want to normalize the fluxes by a certain reaction rate (e.g. by a substrate uptake reaction). In addition, by using the text box colors you want to distinguish between reactions carrying no flux, positive flux (running forward) and negative flux (running backward). A MATLAB function (normalize_flux.m) which interfaces CNA and performs this task could look like this (you find this function in the directory CellNetAnalyzer/code/api):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function cnap = normalize_flux(cnap,reacname)

index=[];for z = 1:cnap.numr        if(strcmp(reacname,deblank(cnap.reacID(z,:))))
index=z;            break;        endend
if(isempty(index))  warndlg(['Reaction name ',reacname,' does not exist'],'Error');
       return;
end

[rr,mm]=CNAreadMFNValues(cnap);

if(rr(index)==0 | isnan(rr(index)))
       warndlg('Normalization cannot be performed','Error');
       return;
end

rr=rr/rr(index);
colidx=ones(length(rr),1);
colidx(find(rr==0))=2;
colidx(find(rr>0))=3;
colidx(find(rr<0))=4;

cnap=CNAwriteMFNValues(cnap,rr,colidx,mm);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

[1] There are other ways to install callbacks besides specifying a string. For detailed information see the MATLAB documentation for GUI callbacks and section 8.1.

In case you want to make this function callable from the CNA menu of a project MetNet (i.e. there is a project variable MetNet) you should write the following two lines in the MATLAB command window:

```
rname='Glucose_uptake';   %example for setting name of the reaction
menhandles=CNAaddMenuEntry(MetNet,'Flux normalization',...
                         'MetNet=normalize_flux(MetNet, rname);');
```

7.6 API functions for computing network features

The following API functions give you the opportunity to call CNA's algorithms for computing complex network features, such as paths, cycles, elementary modes and so forth. Most of these functions (exceptions are, for example, **CNAcomputeCutsets** and **CNATranswesd**) require a CNA network project from which they access usually only some selected N-fields (i.e. no access to GUI). You may thus call these functions with projects that have been loaded with or without GUI and also projects with "partial" information generated e.g. with **CNAgenerateMFNetwork** or **CNAgenerateSFNetwork** as described aboved. The results are usually returned as MATLAB arrays or matrices. The network project itself is not changed or modified.

Here we document the functions and also indicate which N-fields in the network project are required. In general, the documentation for all API functions can also be obtained online in MATLAB by entering: help <function name>.

7.6.1 Analysis of Signal-flow networks

The following API functions are useful for analyzing signal-flow networks. This includes functions for exploring interaction graphs and for studying the behavior of logical networks.

CNAcomputeCycles

This function enumerates cycles (feedback loops) in signal flow networks.

Usage: [cycles, csigns]= **CNAcomputeCycles**(cnap, edge_constr, node_constr, undirected)

Arguments:

- cnap is a CellNetAnalyzer (signal-flow) project variable and mandatory argument. The function accesses the following fields in cnap :
  cnap.interMat: contains the incidence matrix of a graph or of a hypergraph; in the latter case hyperarcs will be split internally and the resulting cycles will be mapped back to the hypergraph
  cnap.notMat: contains the minus signs along the (hyper)arcs
  cnap.reacID: names of the columns (arcs/hyperarcs) in cnap.interMat
  cnap.numr: number of interactions (columns in cnap.interMat)
  cnap.epsilon: smallest number greater than zero

The other arguments are optional:

- edge_constr: [] or a vector of length 'cnap.numr'. if(edge_constr(i)==0) then only those cycles are computed that do not include the interaction i; if(edge_constr(i)~=0 & edge_constr(i)~=NaN) enforces interaction i, i.e. only those cycles will be computed that involve interaction i; interaction i remains unconstrained if(edge_constr(i)=NaN); several interactions may be suppressed/enforced simultaneously(default: [])
- node_constr: [] or a vector of length 'cnap.nums'. if(node_constr(i)==0) then only those cycles are computed that do not include species i; if(node_constr(i)~=0 & node_constr(i)~=NaN) enforces species i,

i.e. only those cycles will be computed that involve species i; species i remains unconstrained if(node_constr(i)=NaN); several species may be suppressed/enforced simultaneously; (default: [])

- undirected: determines whether or not to treat edges as undirected (1) or directed (0); interactions cannot be enforced in undirected graphs (default: 0)

Results:

- cycles: matrix that contains the cycles row-wise (columns correspond to edges/the interactions)
- csigns: vector indicating for each path whether it is positive (1) or negative (-1)

CNAcomputePaths

This function enumerates paths in signal flow networks.

Usage: [paths, psigns]= **CNAcomputePaths**(cnap, edge_constr, ...
node_constr, add_in, addout, specstart, specend)

Arguments:

- cnap is a CellNetAnalyzer (signal-flow) project variable and mandatory argument. The function accesses the following fields in cnap:
  cnap.interMat: contains the incidence matrix of a graph or of a hypergraph; in the latter case hyperarcs will be split internally and the resulting cycles will be mapped back to the hypergraph
  cnap.notMat: contains the minus signs along the (hyper)arcs
  cnap.reacID: names of the columns (arcs/hyperarcs) in cnap.interMat
  cnap.numr: number of interactions (columns in cnap.interMat)
  cnap.epsilon: smallest number greater than zero

The other arguments are optional:

- edge_constr: [] or a vector of length 'cnap.numr'. if(edge_constr(i)==0) then only those cycles are computed that do not include the interaction i; if(edge_constr(i)~=0 & edge_constr(i)~=NaN) enforces interaction i, i.e. only those cycles will be computed that involve interaction i; interaction i remains unconstrained if(edge_constr(i)=NaN); several interactions may be suppressed/enforced simultaneously (default: [])
- node_constr: [] or a vector of length 'cnap.nums'. if(node_constr(i)==0) then only those cycles are computed that do not include species i; if(node_constr(i)~=0 & node_constr(i)~=NaN) enforces species i, i.e. only those cycles will be computed that involve species i; species i remains unconstrained if(node_constr(i)=NaN); several species may be suppressed/enforced simultaneously
  (default: [])
- addin: [0|1] wheter or not to add nodes from the input layer to the set of start nodes (default: [0)
- addout: [0|1] wheter or not to add nodes from the output layer to the set of end nodes (default: 0)
- specstart: either a string of start node names separated by whitespace or an array of start node indices (default: [])
- specend: either a string of end node names separated by whitespace or an array of end node indices (default: [])

Results:

- paths: matrix that contains the paths row-wise (columns correspond to the edges)
- psigns: vector indicating for each path whether it is positive (1) or negative (-1)

CNAcomputeShortestSignedPaths

This function calculates the shortest signed paths in signal flow networks.

Usage: [distpos, distneg, idx, dist]= **CNAcomputeShortestSignedPaths**(cnap, maxdist, edge_constr, ...
node_constr, algo)

Arguments:

- cnap is a CellNetAnalyzer (signal-flow) project variable and mandatory argument. The function accesses the following fields in cnap:
  cnap.interMat: contains the incidence matrix of a graph or of a hypergraph; in the latter case hyperarcs will be split internally
  cnap.notMat: contains the minus signs along the (hyper)arcs
  cnap.reacID: names of the columns (arcs/hyperarcs) in cnap.interMat
  cnap.specID: names of the rows (species) in cnap.interMat
  cnap.nums: number of species (rows in cnap.interMat)

The other arguments are optional:

- maxdist: limits the maximal path length of a shortest path; must be a number greater zero; use 'Inf' to leave the maximal path length unrestricted (default: Inf)
- edge_constr: [] or a vector of length 'cnap.numr'. if(edge_constr(i)==0) then (hyper-)arc i is removed from the graph before calculation (default: [])
- node_constr: [] or a vector of length 'cnap.nums'. if(node_constr(i)==0) then species i is removed from the graph before calculation (default: [])
- algo: selects one of the following algorithms (default: 1=exhaustive):
  1: exhaustive depth-first traversal
  2: approximative algorithm
  3: two-step algorithm (exact and in certain cases faster than algo 1)

Results:

- distpos: matrix with the shortest positive path lengths; '0' means no positive path between the corresponding pair of nodes was found
- distneg: matrix with the shortest negative path lengths; '0' means no negative path between the corresponding pair of nodes was found
- idx: maps the rows/columns of distpos, distneg, dist onto the column indices in cnap.interMat (idx = 1:cnap.nums if no species was removed)
- dist: matrix with the shortest (unsigned) paths lengths; '0' means no path between the corresponding pair of nodes was found

CNAcomputeDepMat

Calculates the dependency matrix in signal flow networks.

Usage: [depmat,idx, distpos,distneg]= **CNAcomputeDepMat**(cnap, maxdist, edge_constr, node_constr, algo)

Arguments:

- cnap is a CellNetAnalyzer (signal-flow) project variable and mandatory argument. The function accesses the following fields in cnap:
  cnap.interMat: contains the incidence matrix of a graph or of a hypergraph; in the latter case hyperarcs will be split internally
  cnap.notMat: contains the minus signs along the (hyper)arcs
  cnap.reacID: names of the columns (arcs/hyperarcs) in cnap.interMat
  cnap.specID: names of the rows (species) in cnap.interMat
  cnap.nums: number of species (rows in cnap.interMat)

The other arguments are optional:

- maxdist: limits the maximal path length of a shortest path; must be a number greater zero; use 'Inf' to leave the maximal path length unrestricted (default: Inf)
- edge_constr: [] or a vector of length 'cnap.numr'. if ~isnan(edge_constr(i)) then (hyper-)arc i is removed from the graph before calculation (default: [])

- node_constr: [] or a vector of length 'cnap.nums'. if ~isnan(node_constr(i)) then species i is removed from the graph before calculation (default: [])
- algo: selects one of the following algorithms (default: 1=exhaustive):
  1: exhaustive depth-first traversal
  2: approximative algorithm
  3: two-step algorithm (exact and in certain cases faster than algo 1)

Results:

- depmat: dependency matrix of the model; the entries in the matrix have the following meaning:
  depmat(i,j)=1: i does not affect j (non-affecting)
  depmat(i,j)=2: i is ambivalent factor of j (i affect j positively and negatively)
  depmat(i,j)=3: i is weak inhibitor of j (i affects j only via negative paths; some of these paths touch negative feedback loops)
  depmat(i,j)=4: i is weak activator of j (i affects j only via positive paths; some of these paths touch negative feedback loops)
  depmat(i,j)=5: i is strong inhibitor of j (i affects j only via negative paths; none of these paths touches a negative feedback loop)
  depmat(i,j)=6: i is a strong activator of j (i affects j only via positive paths; none of these paths touches a negative feedback loop)
- idx: maps the rows/columns of distpos, distneg, dist onto the column indices in cnap.interMat (idx = 1:cnap.nums when no species were removed)

The following two results are a "side-product" that could als separately computed via **CNAcomputeShortest-SignedPaths:**

- distpos: matrix with the shortest positive path lengths; '0' means no positive path between the corresponding pair of nodes was found
- distneg: matrix with the shortest negative path lengths; '0' means no negative path between the corresponding pair of nodes was found

CNATranswesd

Computes the transitive reduction of a weighted signed (interaction) graph. Useful as false positive reduction method in reverse engineering of cellular interaction graphs (see ref. [19]). A CNA project variable is not required.

Usage: [graph] = CNATranswesd(graph,epsilon,fullcheck,pathexact,acyclic,maxpathl)

Arguments:

- graph: In contrast to most CNA API functions, this function does not require a CNA project variable. A mandatory argument is 'graph' which is a structure with the following fields:
  graph.adjpos: adjacency matrix of the positive edges: if graph.adjpos(i,k)>0, then a positive edge from i to k exists and graph.adjpos(i,k) stores its weight.
  graph.adjneg: adjacency matrix of the negative edges: if graph.adjneg(i,k)>0, then a negative edge from i to k exists and graph.adjneg(i,k) stores its weight.

The other four arguments are optional:

- epsilon: confidence factor. An edge from i to k with weight w can be explained by a path from i to k with weight z (and is therefore deleted) if z<w*epsilon. Default: 1. For further explanations see [19].
- fullcheck: whether all path lengths have to be recalculated after removing an edge (1) or not (0). Default: 1.
- pathexact: whether path length have to be calculated exactly (1) or approximately (0). Default: 1.
- acyclic: if 1, only edges between nodes from different (graph) components will be considered for removal during transitive reduction. Default: 0.
- maxpathl: maximum number of edges that a path may have if it is used to explain an edge. Choose Inf if any appropriate path (see parameter epsilon above) can explain an edge. Default: Inf.
  If 2<maxpathl<INF (and maxpathl not too large) it is recommended to use this feature only in combination with pathexact=fullcheck=1.

Results:

- Graph: The returned result is a graph with the same fields as in the input graph plus graph.recadjpos and graph.recadjneg describing the computed transitive reduction of the input graph.

CNAcomputeLSS

Calculates the logical steady state in a signal-flow network for a given set of input stimuli or fixed signal/species values.

Usage: [spec_lss, inter_lss]= CNAcomputeLSS(cnap, spec_vals, inter_vals)

Arguments:

- cnap: is a CellNetAnalyzer (signal-flow) project variable and mandatory argument. The function accesses the following fields in cnap:
  cnap.interMat: contains the interactions
  cnap.notMat: contains the minus signs along the (hyper)arcs
  cnap.excludeInLogical: if cnap.excludeInLogical(i) ~= 0 then reaction i will be excluded from the calculation
  cnap.nums: number of species (rows in cnap.interMat)
  cnap.numr: number of interactions (columns in cnap.interMat)

The other arguments are optional:

- spec_vals: [] or a vector of length 'cnap.nums'; if non-empty the value of spec_vals(i) is used as predefined value for species i; use 'NaN' as value to leave species undefined (default: [])
- inter_vals: [] or a vector of length 'cnap.numr'; if non-empty the value of inter_vals(i) is used as predefined value for interaction i; use 'NaN' as value to leave interactions undefined (default: [])

Results:

- spec_lss: gives the computed logical steady states of the species (NaN indicates that the steady state of the corresponding species is undetermined)
- inter_lss: the signal flow along the interactions in the computed logical steady states (NaN means that the value of the corresponding interaction is undetermined)

CNAcomputeMIS

Calculates (logical) minimal intervention sets in signal flow networks fulfilling a set of goals defined by a scenario. It is possible to define multiple scenarios whose goals all have to be fulfilled by the MIS. The goals and fixed species/interactions are specific for each scenario while the remaining parameters apply to all scenarios.

Usage: [mis, idx, ec, stat]= **CNAcomputeMIS**(cnap, scen, max_mis_size, err_tol, count_minerr,...
 spec_restr, allow_inact, allow_actv, excl_fix_spec , dispval, fpath, fname)

Arguments:

- cnap is a CellNetAnalyzer (signal-flow) project variable and mandatory argument. The function accesses the following fields in cnap (see also manual):
  cnap.interMat: contains the incidence matrix of a graph or of a hypergraph
  cnap.notMat: contains the minus signs along the (hyper)arcs
  cnap.incTruthTable: vector containing the incomplete truth table flag of the interactions
  cnap.monotony: vector containing the monotony flag of the interactions
  cnap.specID: names of the rows (species) in cnap.interMat
  cnap.reacID: names of the columns (arcs/hyperarcs) in cnap.interMat
  cnap.nums: number of species in the network
  cnap.numr: number of (hyper)arcs in the network

The other arguments are:

- scen(mandatory): struct array of scenarios with 6 fields; each field is either empty or contains a row vector that describes the intervention goal or the state of fixed species/interactions
  these fields contain the intervention goals:
  goal_spec: species values that must result from the intervention
  goal_spec_not: species values that are forbidden in the intervention
  goal_inter: interaction values that must result from the intervention
  goal_inter_not: interaction values that are forbidden in the intervention
  these fields contain the fixed states:
  fix_spec: fixed species
  fix_inter: fixed interactions
  when a non-empty vector is given it must have as many elements as interactions resp. species in the network; its entries correspond to the interactions/species and contain either the value of the goal/state or NaN when no goal/state is given each index entry of the struct array describes the goals/states of one scenario, i.e. the fields of scen(i) describe the i-th scenario

The remaining arguments are optional:

- max_mis_size: the maximum number of interventions in one MIS (default: Inf)
- err_tol: number of errors allowed in a MIS to count as accepted (default: 0)
- count_minerr: whether the minimal number of errors produced by an candidate MISs should be stored and displayed at the end in case that no MISs could be found satisfying all intervention goals (note that this may increase computation time considerably!) (default: 0)
- spec_restr: an empty vector or a vector of restrictions on the activatibility of the species; in the latter case the vector must have a length equal to cnap.nums; it either contains NaN when no restrictions are imposed on a species or one of the following:
  -1: no inactivation allowed
  -2: no activation allowed
  -3: neither inactivation nor activation allowed
  (default: [])
- allow_inact: flag which determines whether species can be activated (default: true)
- allow_actv: flag which determines whether species can be inactivated (default: true)
- excl_fix_spec: flag which determines whether species with a fixed state are excluded to be excluded from the interventions (default: false)
- dispval: controls the output printed to the console
  0: print only warnings
  1: output preprocessing information (summary information in case of multiple scenarios)
  2: additional preprocessing output for each scenario (applicable only when multiple scenarios are defined)
- fpath, fname: if both are specified the MIS are saved in the directory fpath under fname

Results

- mis: matrix that contains the MIS row-wise; depending on the value the corresponding species (cf. idx) is:
  1: activated
  0: not part of the intervention set
  -1: deactivated
- idx: maps the columns in 'mis' onto the column indices in cnap.interMat, i.e. idx(i) refers to the column number in cnap.interMat (and to the row in cnap.reacID)
- ec: ec(i) is the number of goals that were not fulfilled by mis(i, :); always <= err_tol
- stat: information about the way in which MIS calculation terminated; especially useful when no MIS were calculated; possible values are:
  0: calculation finished normally
  1: goals are already fulfilled without any intervention
  2: neither activatable nor removable species in at least one scenario
  3: if count_minerr was on and no cut sets were found this status indicates that 'ec' contains the minimal number of discrepancies
  -1: incompatible goals within one scenario
  -2: all scenarios without goals

7.6.2 Analysis of Mass-flow networks

The following API functions can be used for analyzing mass-flow networks.

CNAoptimizeFlux

Performs flux optimization in mass-flow networks: ojective function cnap.objFunc is minimized by linear programming (LP).

Usage: [flux_vec, success, status]= **CNAoptimizeFlux**(cnap, constraints, c_macro, solver, dispval)

Arguments:

- cnap is a CellNetAnalyzer (mass-flow) project variable and mandatory argument. The function accesses the following fields in cnap (see also manual):

cnap.stoichmat: the stoichiometric matrix of the network
cnap.objFunc: vector containing the coefficients of the reactions in the linear objective function
cnap.numr: number of reactions (columns in cnap.stoichMat)
cnap.numis: number of internal species
cnap.mue: index of the biosynthesis reaction; can be empty
cnap.macroComposition: matrix defining the stoichiometry of the macromolecules with respect to the metabolites (species); matrix element macroComposition(i,j) stores how much of metabolite i (in mmol) is required to synthesize 1 gram of macromolecule j
cnap.specInternal: vector with the indices of the internal species
cnap.macroDefault: default concentrations of the macromolecules
cnap.reacMin: lower boundaries of reaction rates (if reacMin(i)=0 --> reaction i is irreversible)
cnap.reacMax: upper boundaries of reaction rates

The other arguments are optional:

- constraints: constraints is a (numrx1) or empty vetor; if the value constraints(i) is a number then the flux through reaction i is kept at the given rate during optimization; if the value constraints(i) is NaN then the flux through reaction i is determined by the LP solver (within the bounds given by cnap.reacMin and cnap.reacMax); if constraints is an empty vector all reaction rates are determined by the LP solver (default: [ ])
- c_macro: vector containing the macromolecule values (concentrations); can be empty when cnap.mue or cnap.macroComposition is empty (default: cnap.macroDefault)
- solver: selects the LP solver
  0: GLPK
  1: linprog (Matlab Optimization Toolbox)
  (default: 0)
- dispval: controls the output printed to the console
  0: no output
  1: regular output
  2: if solver == 1 then the LP is run a second time starting from random values; when the second result is sufficiently different from the first it can be concluded that the solution is probably not unique
  (default: 0)

Results

- flux_vec: when success == true: result of the optimization when succes == false: NaN if system is not underdetermined or the result returned by the LP solver (may not be meaningful)
- success: flag indicating whether the optimization was successful
- status: solver status; for interpretation check the documentation of the selected LP solver

CNAfluxVariability

Flux variability analysis in mass-flow networks.

Usage: [minFlux,maxFlux,success,status] = **CNAfluxVariability**(cnap,reacval,macromol,solver)

Given a mass-flow project (with or without GUI) and a set of predefined fluxes this function determines the range of feasible fluxes for each reaction by solving linear optimization problems (classical Flux Variability Analysis; requires MATLAB optimization toolbox).

Arguments:

- cnap: (mandatory) is a CellNetAnalyzer (mass-flow) project variable. The function accesses the following fields of cnap:
  cnap.stoichmat: the stoichiometric matrix of the network
  cnap.numr: number of reactions (columns in cnap.stoichMat)
  cnap.numis: number of internal species
  cnap.mue: index of the biosynthesis reaction; can be empty
  cnap.macroComposition: matrix defining the stoichiometry of the macromolecules with respect to the metabolites (species); matrix element macroComposition(i,j) stores how much of metabolite i (in mmol) is required to synthesize 1 gram of macromolecule j
  cnap.specInternal: vector with the indices of the internal species
  cnap.macroDefault: default concentrations of the macromolecules
  cnap.reacMin: minimal reaction rates
  cnap.reacMax: maximal reaction rates

The other arguments are optional:

- reacval: is a (numrx1) vector specifying the predefined fluxes; reacval(i) is either a number defining a fixed flux or t is NaN indicating a free flux (whose feasible upper and lower boundaries subject to the predefined fluxes are determined by this function) (default value of reacval: a (numr x 1) NaN vector)
- macromol: vector containing the macromolecule values (concentrations); can be empty when cnap.mue or cnap.macroComposition is empty (default: cnap.macroDefault)
- solver: selects the LP solver: 0 = GLPK; 1= linprog (Matlab Optimization Toolbox); default is 0.

Results:

- minFlux: contains the mnimal feasible fluxes (for every reaction) consistent with predefined values in reacval; note that minFlux(i)=reacval(i) for all predefined fluxes (i.e. where reacval(i) was a numerical value); when success == false: minFlux(j)=NaN for all j not predefined
- maxFlux: contains the maximal feasible fluxes (for every reaction) consistent with predefined values in reacval; note that maxFlux(i)=reacval(i) for all predefined fluxes (i.e. where reacval(i) was a numerical value); when success == false: maxFlux(j)=NaN for all j not predefined

Note that a flux i is uniquely determined if minFlux(i)=maxFlux(i).

- success: flag indicating whether the optimizations were successful
- status: solver status (useful for checking reasons of unsuccessful optimization (e.g. overly stirngent systems); for interpretation check the documentation of the selected LP solver

CNAcomputeEFM

Computes elementary modes / elementary vectors or a minimal generating set (convex basis) of flux cones or flux poloyhedra 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 + reversibility constraints forms a polyhedral (flux) cone. Elementary modes correspond to particular (elementary) rays with an irreducible set of non-zero elements and include all extreme rays of the flux cone (but possibly more). In contrast, the convex basis of a flux cone is the lineality space plus the set of *extreme* rays of that cone. (ii) In the *inhomogeneous* case, inhomogeneous constraints (e.g. fixing a reaction rate to a non-zero value or introducing upper and/or lower boundaries for the rates) form a flux polyhedron. CNAcomputeEFM will then compute either the elementary vectors (with irreducible number of non-zero entires) of the flux polyhedron (including all extreme rays and extreme points) OR, again, only a minimal set of unbounded (lineality space + extreme rays) and bounded (extreme points) generators spanning the resulting flux polyhedron (Minkowsi sum). Note that the zero point will not be delivered, even if it is an extreme point of the solution space. Most applications focus on elementary modes in the homogeneous setting but there are also applications for inhomogeneous specifications. For more information see reference [23].

Usage: [efm,rev,idx,ray] = **CNAcomputeEFM**(cnap, constraints, mexversion, irrev_flag,...
convbasis_flag, iso_flag, c_macro, display, efmtool_options)

Arguments:

- cnap is a CellNetAnalyzer (mass-flow) project variable and mandatory argument. The function accesses the following fields in cnap (see also manual):
  cnap.stoichmat: the stoichiometric matrix of the network
  cnap.numr = number of reactions (columns in cnap.stoichMat)
  cnap.mue: index of the biosynthesis reaction; can be empty
  cnap.macroComposition: matrix defining the stoichiometry of the
  cnap.specInternal: vector with the indices of the internal species
  cnap.reacID: names of the columns (reactions) in cnap.stoichMat
  cnap.specID: names of the rows (species) in cnap.stoichMat
  cnap.macroID: names of the macromolecules
  cnap.macroDefault: default concentrations of macromolecules
  cnap.reacMin: lower boundaries of reaction rates (if reacMin(i)=0 --> reaction i is irreversible)
  cnap.reacMax: upper boundaries of reaction rates
  cnap.epsilon : smallest number greater than zero (for numerical purposes)

The other arguments are optional:

- constraints: is a matrix specifying homogeneous and inhonogeneous constraints on the reaction rates; the matrix is either empty (no constraints considered - this is also the default value) or has cnap.numr many rows and up to 4 columns:
  - COLUMN1 specifies excluded/enforced reactions: if(constraints(i,1)==0) then only those modes / rays / points will be computed that do not include reaction i; constraints(i,1)~=0 and constraints(i)~=NaN enforces reaction i, i.e. only those modes / rays / points will be computed that involve reaction i; for all other reactions choose constraint(i,1)=NaN; several reactions may be suppressed/enforced simultaneously
  - COLUMN2: specifies lower boundaries for the reaction rates (choose NaN if none is active). Note that zero boundaries (irreversibilities) are better described by cnap.reacMin. In any case, the lower boundary eventually considered will be zero if cnap.reaMin(i)==0 and constraints(i,2)<0.
  - COLUMN3: specifies upper boundaries for the reaction rates (choose NaN if none is active)
  - COLUMN4: specifies equalities for the reaction rates (choose NaN if none is active).
  If columns 2,3, or 4 are not specified or if they do not contain any non-zero (non-NaN) value, then the elementary modes (convbasis_flag=0) or minimal generating set (convbasis_flag=1) of the flux cone will be computed (homogeneous problem). Any non-zero non-NaN value in COLUMN 2,3, or 4 renders the problem to be inhomogeneous and the solution space to be a (flux) polyhedron. This function will then compute either the elementary vectors with maximum number of zeros (including all extreme rays and extreme points) of the flux polyhedron (if convbasis_flag=0) OR, again, only a minimal set of unbounded (lineality space + extreme rays) and bounded (extreme points) generators spanning the resulting flux polyhedron (if convbasis_flag=0). For the inhomogenous case, 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) and the returned vector ‘rev’ indicates whether a mode/ray is reversible or not (see below). Be careful to not define inconsistent constraints, such as constraints(i,:)=[NaN,2,3,1] (reaction i cannot be in the range of [2,3] and exactly 1 at the same time).
  Note that cnap.reacMin and cnap.reacMax cannot be used for specifying inhomogeneous constraints. cnap.reacMin is only used for marking reaction reversibility.
- mexversion: [0|1|2|3|4] 0: scripts, 1: CNA mex files, 2: Metatool mex files, 3: CNA and Metatool mex files 4: Marco Terzer's EFM tool (see http://www.csb.ethz.ch/tools/index) (the toolbox must be installed and in the MATLAB path); (default: 3)
- irrev_flag: [0|1] wheter or not to consider reversibilities of reactions 0: all reactions are reversible (default: 1)
- convbasis_flag: [0|1] whether all elementary vectors (including all extreme rays and extreme points) of the flux cone / flux polyhedron are to be computed [0] or whether only a minimal generating set (convex basis) [1] is to be calculated. For example, in a homogeneous system, setting this flag to 0 will compute the elementary modes of the flux cone. Default: 0.
- iso_flag: [0|1] wheter or not to consider isoenzymes (parallel reactions) only once (default: 0)

- c_macro: vector containing the macromolecule values (concentrations); can be empty when cnap.mue or cnap.macroComposition is empty (default: cnap.macroDefault)
- display: control 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 used)

Results:

- efm: matrix that contains (row-wise) the elementary modes (elemenatry vectors) or a minimal set of generators (lineality space + extreme rays/points), depending on the chosen scenario; the columns correspond to the reactions; the column indices of efms (with respect to the columns in cnap.stoichMat) are stored in the returned variable idx (note that columns are removed in efms if the corresponding reactions are not contained in any mode)
- rev: vector indicating for each mode 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').

CNAcomputeCutsets

The *Berge* algorithm for calculation of minimal cut sets for a given set of paths/cycles/elementary modes. (Mathematically, the Berge algorithm performs a hypergraph transversal). A CNA project variable is not required.

Usage: cutsets= **CNAcomputeCutsets**(targets, mcsmax, names, sets2save, earlycheck)

Arguments:

- targets (mandatory): matrix that row-wise contains the paths/cycles/elementary modes; the only distinction made is between zero and non-zero elements; mandatory argument, has to be non-empty

The following arguments are optional:

- mcsmax: maximal size of cutsets to be calculated; must be a value grater 0; Inf means no size limit (default: Inf)
- names: a char matrix - the rows are names corresponding to the columns of 'targets'; used to display results from preprocessing (default: [ ]; the names are then set to ‘I1’,’I2’,...)
- sets2save: struct array with sets of modes/paths/cycles that should be preserved (not be hit by the cut sets computed). The following fields are required:
  sets2save(i).tabl2save = i-th set of paths/cycles/modes that should be saved sets. As for the “targets”, tabl2save is a matrix that row-wise contains the paths/cycles/ modes. The matrix must have the same number of columns as 'targets'.
  sets2save(i).min2save = specifies the minimum number of sets (paths/cycles/modes) in sets2save(i).tabl2save that should not be hit by the cutsets to be computed.
  By default sets2save is empty ([ ]).
- earlycheck: whether the test checking for the fulfillment of constraints in sets2save should be caried out during (1) or after (0) computation of cut sets [default: 1; makes only sense in combination with sets2save]

Results:

- cutsets: matrix that contains the cutsets row-wise; a 1 means that the reaction/interaction is part of the cutset, 0 means the reaction/interaction is not involved. Each cutset hits all modes stored in ‘targets’ while it does not hit at least ‘sets2save(i).min2save’ many modes in ‘sets2save(i).tabl2save’ for each specified set i.

CNAapplyCASOP

Calculation of reaction weights and reaction ranking for identification of knockout or overexpression candidates according to our recently proposed method CASOP (= Computational Approach for Strain Optimization aiming at high Productivities) [18]. Applicable in mass-flow networks. The target metabolite (desired product) has to be defined as an external metabolite. The biomass reaction can be defined either by the standard biomass synthesis reaction “mue” (see Section 3) or by a user defined biomass synthesis reaction. In the latter case, an external biomass metabolite has to be defined (which must be produced in the user-defined biomass synthesis reaction) and the names of the biomass metabolite and synthesis reaction are additional arguments for CNAapplyCASOP.

```
Usage: [ReacWeightMatrix, RatingMatrix]= CNAapplyCASOP(cnap, gamma_vec, k_vec, product_name, ...
                molm_product, uptake_reaction_names, rating_boundaries_vec, plot_reaction_ids, ...
                legendID, EMoptions, BMreaction, BMmetabolite)
```

Arguments (see also CASOP publication [18]):

- cnap: CellNetAnalyzer (mass-flow) project variable. The function accesses the following fields of cnap:
  cnap.stoichmat: the stoichiometric matrix of the network
  cnap.numr = number of reactions (columns in cnap.stoichMat)
  cnap.mue: index of the biosynthesis reaction; can be empty
  cnap.macroComposition: matrix defining the stoichiometry of the
  cnap.specInternal: vector with the indices of the internal species
  cnap.reacID: names of the columns (reactions) in cnap.stoichMat
  cnap.specID: names of the rows (species) in cnap.stoichMat
  cnap.specInternal: array of indices of internal species
  cnap.specExternal: (1 x n) vector indicating which species are external (1) an which not (0)
  cnap.nums: number of species in the network
  cnap.numis: number of internal species
  cnap.macroID: names of the macromolecules
  cnap.macroDefault: default concentrations of the macromolecules
  cnap.reacMin: lower boundaries of reaction rates (if reacMin(i)=0 --> reaction i is irreversible)
  cnap.reacMax: upper boundaries of reaction rates
  cnap.epsilon : smallest number greater than zero (for numerical purposes)

- gamma_vec: vector with proportions of product at container V
- k_vec: vector with exponents for quantitative weighting (see CASOP publication for details)
- product_name: string with name of target metabolite (must be an external metabolite)
- molm_product: molar mass of the product
- uptake_reaction_names: cell array with names of (substrate) uptake reactions (each row gives one name = string)   --> these reactions characterize the (limited) ressources required for synthesizing the product and they serve as normalization factor when computing the yield
- rating_boundaries_vec:
  either a single value (must be an element of gamma_vec) --> then rating type 1 (= importance at this gamma) is applied
  or a vector with two values (both must be elements in gamma_vec) --> then rating type 2 (= difference of importances between upper and lower gamma value) is applied

The other arguments are optional:

- plot_reaction_ids: cell array with strings of names of those reactions whose importances are to be plotted (each row gives one name = string) (default: empty array - plot nothing)
- legendID: plot with (1, default) or without (0) legend
- EMoptions: structure with fields containing optional arguments for EM computation (for details see CNAcomputeEFM). Default values:
  ```
  EMoptions.constraints=[]
  EMoptions.mexversion=4
  EMoptions.irrev_flag=1
  EMoptions.convbasis_flag=0
  EMoptions.iso_flag=0
         EMoptions.c_macro=cnap.macroDefault;
         EMoptions.display='None'
  ```

- BMreaction: string with name of biomass synthesis reaction (to be used if 'mue" is not used as standard biomass reaction)
- BMmetabolite: string with name of biomass metabolite which must be external and be a product within the BMreaction (to be used if 'mue" is not used as standard biomass reaction)
  Note: If a BMmetabolite is specified, the stoichiometry of the BMreaction producing this metabolite must be defined in such a way that 1 gram biomass (metabolite) is produced by this reaction.

Results:

- ReacWeightMatrix: cell array where ReacWeightMatrix{i,j} holds the vector with reaction importances for the i-th value in gamma_vec and j-th value in k_vec.
- RatingMatrix: cell array where RatingMatrix(i,j) holds the rating value for the i-th reaction in cnap.reacID and the j-th value in k_vec

7.7 API functions for loading and saving values

CNAloadMFNValues

Load reaction and macromolecule values of mass-flow networks from a (CNA) val-file.

Usage: [reacval, macroval]= CNAloadMFNValues(cnap, fname)

All parameters are mandatory:

- cnap: CellNetAnalyzer (mass-flow) project variable
- fname: name of the val-file

The following results are returned:

- reacval: vector of reaction values as specified in the val-file (contains NaN for reactions that were not specified in the val_file) or [] when err=2
- macroval: vector of macromolecule values as specified in the val-file (contains NaN for macromolecules that were not specified in the val_file) or [] when err=2
- err: error code; possible values:
  0: no error occurred
  1: invalid identifier(s) in the val-file; these are skipped
  2: val-file could not be opened

CNAsaveMFNValues

Save reaction and macromolecule values of a mass-flow project to a (CNA) val-file.

Usage: err= CNAsaveMFNValues(cnap, fname, reacval, macroval)

All parameters are mandatory:

- cnap: CellNetAnalyzer (mass-flow) project variable
- fname: name of the val-file
- reacval: vector of reaction values (if a reaction has the value NaN it is not saved in the file)
- macroval: vector of macromolecule values (if a macromolecule has the value NaN it is not saved in the file)

The following results are returned:

- err: reports whether an error occurred (1) or not (0)

CNAloadSFNValues

Load interaction and species values from a (CNA) val-file in signal-flow networks.

Usage: [reacval, specval]= CNAloadSFNValues(cnap, fname)

All parameters are mandatory:

- cnap: CellNetAnalyzer (signal-flow) project variable
- fname: name of the val-file

The following results are returned:

- reacval: vector of interaction values as specified in the val-file (contains NaN for interactions that were not specified in the val_file) or [] when err=2
- specval: vector of species values as specified in the val-file (contains NaN for species that were not specified in the val_file) or [] when err=2
- err: error code; possible values:
  0: no error occurred
  1: invalid identifier(s) in the val-file; these are skipped
  2: val-file could not be opened

CNAsaveSFNValues

Saves interaction and species values into to a (CNA) val-file.

Usage: err= CNAsaveSFNValues(cnap, fname, reacval, specval)

All parameters are mandatory:
- cnap: CellNetAnalyzer (signal-flow) project variable
- fname: name of the val-file
- reacval: vector of interaction values (if an interaction has the value NaN it is not saved in the file)
- specval: vector of species values (if a species has the value NaN it is not saved in the file)

The following results are returned:

- err: reports whether an error occurred (1) or not (0)

7.8 Starting a network project w/o GUI

It is possible to work with a project without using the GUI (at least to some extent) which might be useful for certain applications. You can load the network without the GUI by clicking on "Start w/o GUI" in the project manager.
You may even suppress the project manager GUI when starting CNA, e.g. if you want to operate in batch-mode, if you call CNA only for calculating some network features, or if you are using octave. You can do this by calling **startcna(1)**. As described in a previous section, the API function **CNAloadNetwork** allows one to load network projects (optionally) without GUI, i.e., without interactive maps. Note again that you must definitely initialize CNA (with **startcna**) before using any of the API functionality presented in this chapter. Calling **startcna** - which is

located in *CellNetAnalyzer‘s* main directory - adds all relevant CNA directories to the MATLAB path, initializes the global variable **cnan** and launches the project manager window (unless called with **startcna**(1)). To finish CNA enter **close_cna**, which closes all projects that are running with a GUI, removes the CNA directories from the MATLAB path and deletes the **cnan** variable.

7.9 The global cnan variable and CNA options

When starting the CNA with **startcna**, the global variable *cnan* is initialized and appears in the base workspace. *cnan* is a structure variable whose fields are used to store data for the project manager, certain GUI handles of projects with an active GUI and information about the CNA configuration. For the user the most relevant field is *cnan.options* which controls the behavior of the CNA and its GUI to some degree. The current options and their possible values are:
cnan.options.ask_when_closing_project: controls whether a confirmation dialog comes up when closing one of the network maps. Possible values are 0: never ask; 1: ask when there are unsaved changes; 2: ask always.
cnan.options.ask_when_saving_net: controls whether a confirmation dialog comes up when saving the network structure. Possible values are 0: never ask; 1: ask if the project has an active GUI.
cnan.options.ask_when_closing_composer: controls whether the user is asked to save changes when closing the network composer and unsaved changes exist. Possible values are 0: never ask; 1: ask if unsaved changes exist.

7.10 Example for using the CNA API from another application

Suppose you have an application that provides a stoichiometric matrix and information about reaction reversibilites. Let’s assume that the stoichiometric matrix is called *st* and that there is a vector irrev that contains 0 for reversible reactions and 1 for irreversible ones. The length of *irrev* must be the same as the number of columns of *st*. Futhermore, let *st* contain only internal metabolites. The following example illustrates how you can use the API of CNA to calculate the elementary modes and the corresponding minimal cut sets from the latter:

```
...
startcna(1); % initialize CNA without project manager
cnap.stoichMat= st;
cnap.reacMin= zeros(length(irrev), 1); % must be a column vector
cnap.reacMin(irrev == 0)= -Inf; % set up reversibility
%all other fields of cnap will be set (with default values) by the following command
cnap= CNAgenerateMFNetwork(cnap);
[ems, irrev_ems, ems_idx] = CNAcomputeEFM(cnap);
mcs=CNAcomputeCutsets(ems);
close_cna; % close CNA and clean up
...
```

Of course you could also encapsulate these commands in a separate function. In any case, you have to make sure that the *CellNetAnalyzer* directory is in the MATLAB path because otherwise *startcna* will not be found (and the CNA paths would not be set).

7.11 CNA plugins

The API functions of *CellNetAnalyzer* allow interested developers to build their own applications interfacing CNA. As demonstrated, it can also be used to build CNA plugins such as ODEfy (which is now even a permanent part of CNA). On CNA’s website you can find a list of downloadable plugins developed by others.

8. References

Articles with * can be found in the “doc” directory; articles with “+” are open access and thus freely available at the journal’s website. Please let us know if you would like to have a copy of one of the cited refernces.

Mass-flow (stoichiometric, metabolic) networks:

*[1] Klamt S, Schuster S, Gilles ED. 2002. Calculability analysis in underdetermined metabolic networks illustrated by a model of the central metabolism in purple nonsulphur bacteria. **Biotech Bioeng**, 77 (7) , 734-751.

*[2] Stelling J, Klamt S, Bettenbrock K, Schuster S and Gilles ED. 2002/2003. Metabolic network structure determines key aspects of functionality and regulation. **Nature**, 420, 190-193.

[3] Klamt S, Stelling J, Ginkel M and Gilles ED. 2003. *FluxAnalyzer*: Exploring structure, pathways and flux distributions on interactive flux maps. **Bioinformatics**, 19(2), 261-269

*[4] Klamt S and Stelling J. 2003. Two approaches for metabolic pathway analysis? **Trends in Biotechnology**, 21(2), 64-69.

[5] Stephanopoulos et al. 1998. Metabolic Engineering. Academic Press.

[6] Van der Heijden et al. 1994. **Biotech Bioeng,** 43, 3-10

[7] Schuster S, Fell DA and Dandekar T. 2000. A general definition of metabolic pathways useful for the systemic organization and analysis of complex metabolic networks. **Nature Biotechnology,** 18, 326-332.

[8] Heinrich R and Schuster S. 1996. The regulation of cellular systems. Chapman & Hall.

*[9] Klamt S and Gilles ED. 2004. Minimal cut sets in biochemical reaction networks. **Bioinformatics**, 20(2), 226-234.

[9a] Klamt S. 2006. Generalized concept of minimal cut sets in biochemical networks. **Biosystems**, 83, 233-247.

*+[10] Gagneur J and Klamt S. 2004. Computation of elementary modes: a unifying framework and the new binary approach. **BMC Bioinformatics** 5:175.

[15] Terzer M and Stelling J. 2008: Large-scale computation of elementary flux modes with bit pattern trees. **Bioinformatics** 24(19):2229-35.

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

[21] Hädicke O and Klamt S. (2011) Computing complex metabolic intervention strategies by constrained minimal cut sets. **Metabolic Eingineering** 13:204-213.

Signal-flow (signaling, regulatory) networks (freely available from the journal web-sites):

*+[11] Klamt S, Saez-Rodriguez J, Lindquits J, Simeoni L and Gilles ED. 2006. A methodology for the structural and functional analysis of signaling and regulatory networks. **BMC Bioinformatics, 7:56**.

+[13] Saez-Rodriguez J, Simeoni L, Lindquist JA, Hemenway R, Bommhardt U, Arndt B, Haus UU, Weismantel R, Gilles ED, Klamt S and Schraven B. 2007. A logical model provides insights into T cell receptor signaling. **PLoS Computational Biology** 3:163.

+[14] Klamt S and von Kamp A. 2009. Computing paths and cycles in biological interaction graphs. BMC **Bioinformatics**, 10:181.

+[16] Wittman D, Krumsiek J, Saez-Rodriguez J, Lauffenburger D, Klamt S, Theis F. 2009. Transforming Boolean Models to Continuous Models: Methodology and Application to T-Cell Receptor Signaling. **BMC Systems Biology** 3:98.

*+[17] Samaga R, Saez-Rodriguez J, Alexopoulos L, Sorger PK and Klamt S. 2009. The logic of ERGR/ErbB signaling: theoretical properties and analysis of high-throughput data. **PLoS Computational Biology**, 5(8):e1000438

+ [19] Klamt S, Flassig R and Sundmacher K 2010. TRANSWESD: inferring cellular networks with transitive reduction. **Bioinformatics**, 26:2160-2168.

+[20] Samaga R, von Kamp A and Klamt S. 2010. Computing combinatorial intervention strategies and failure modes in signaling networks. **Journal of Computational Biology**, 17:39-53.

[23] Urbanczik R. 2007. Enumerating constrained elementary flux vectors of metabolic networks. IET Systems Biology 1, 274-279.

Reference for citing *CellNetAnalyzer*:

*+[12] Klamt S, Saez-Rodriguez J and Gilles ED. 2007. Structural and functional analysis of cellular networks with CellNetAnalyzer. **BMC Systems Biology**, 1:2.

[22] Klamt S and von Kamp A. (2011) An application programming interface for CellNetAnalyzer. **BioSystems**,105: 162-168.

CellNetAnalyzer Homepage:

http://www.mpi-magdeburg.mpg.de/projects/cna/cna.html

Online Manual: http://www.mpi-magdeburg.mpg.de/projects/cna/manual/toc_frame.html