SYNBADm: SYNthetic Biology Automated Optimal Design in Matlab
version 1.0.0
User's guide

Irene Otero-Muras, David S. Henriques, Julio R. Banga
ireneotero@iim.csic.es
davidh@iim.csic.es
julio@iim.csic.es

Contents

1 SYNBADm

- Webpage: `https://sites.google.com/site/synbadm/`
- Authors: Irene Otero-Muras, David Saque Henriques, Julio R. Banga
- e-mail: ireneotero@iim.csic.es
- Version: 1.0.0
- License: GPLv3
- Copyright: CSIC, Spanish National Research Council

1.1 Software requirements and compatibility

-SYNBADm is compatible with Matlab 64-bit under Windows and Linux.

-SYNBADm has been tested with Matlab version 2015b.

-SYNBADm allows fast dynamic simulation by automatically converting dynamic models to C code. This feature requires the installation of a compatible C++ compiler. For more information, go to:

`http://es.mathworks.com/support/compilers/R2015b/index.html`

-Alternatively, dynamic models can be integrated with Matlab solvers (without requiring a C++ compiler), but the execution times will be much longer.

1.2 Overview

SYNBADm is a toolbox for automated optimal design of biocircuits (gene regulatory networks) with Synthetic Biology applications in mind, and making use of Mixed Integer Nonlinear Programming (MINLP).

Starting from a library of standard components, and given a set of user specifications and performance criteria, the algorithm selects the combination(s) of devices with optimal performance and meeting the specifications.

Main features of SYNBADm:

- **Single Objective Optimal Design of Biocircuits:** Starting from a library of standard components, the algorithm selects the combination(s) of devices with optimal performance (according to the objective function defined by the user).
- **Multiple Objective Optimal Design of Biocircuits:** Two competing objectives are optimized, i.e. starting from a library of standard components, the algorithm finds the Pareto front of optimal (non-dominated) solutions [6], i.e. the set of best trade-offs.
- Users can easily define their own design (objective) functions, using the provided templates as examples.
- Users can easily generate their own libraries of components (with Mass Action Kinetics or Hill Kinetics).
- Several efficient Mixed Integer NonLinear Programming solvers (*ESS*, *MITS*, *ACOmi*) are provided.
- Supports dynamic simulation (ODE integration) using *CVODES* [10].

1.3 Installation

1. Unzip and copy the toolbox folder SYNBAD to a directory of your choice. IMPORTANT: Do not change the name of the SYNBAD folder.

2. Linux only: the first time you use SYNBADm in Linux, (i) in Matlab change to the SYNBAD main folder and run `»SYNBAD_install`; (ii) compile the default library files (in order to use C++ integrators), changing to the folder MA_library and executing:

   `»SYNBAD_Makelibrary_MA_C('MA_input_library')`.

   Then change to the folder HL_library and execute:

   `»SYNBAD_Makelibrary_HL_C('HL_input_library')`.

   This step is needed only the first time you use SYNBADm, before running `»SYNBAD_Startup`.

3. From Matlab, change to the SYNBAD directory and run `»SYNBAD_Startup`, which adds all the relevant files to the path. IMPORTANT: before using SYNBADm, remember to run `»SYNBAD_Startup` in every new Matlab session.

Now you can start using SYNBADm for optimal biocircuit design. A number of detailed test examples are given in Section 10.

1.4 Questions and troubleshooting

For questions, feedback and troubleshooting, please contact `ireneotero@iim.csic.es`

1.5 About this manual

- Basic knowledge of dynamical systems and their simulation in Matlab is assumed.

- In this manual we use `verbatim fonts` for Matlab scripts, functions, commands and mat files, **bold fonts** for Matlab structures (and structure fields) and *cursive fonts* for optimization and initial value problem (IVP) solvers.

2 Quick start

- Start Matlab.
- Go to the SYNBAD directory.
- Type: `SYNBAD_Startup` (SYNBADm folders are added to the Matlab path).
- Go to the USR_Libraries folder and generate the library of components. Library options are specified in the structure **library**, using a library input file. To generate the files, call one of the SYNBADm library functions (see Fig. 1) using as argument the name of the library input file. Detailed instructions on how to generate a library of components are given in Section 3. You can also use a predefined library.
- Go to the USR_ObjFuns folder and generate the objective function(s). Instructions on how to generate an objective function are given in Section 4. You can also use a predefined objective funcion.
- Go to the USR_Inputs folder and define the design problem. Design options are specified in the structure **inputs**, using a problem input file. Instructions are given in Section 5.
- For single objective design: from the main SYNBAD directory, call `SYNBAD_Design_SO` using as argument the name of the problem input file. The results are stored in the file `RESULTS_DESIGN.mat`.
- For multiple objective design: from the main SYNBAD directory, call `SYNBAD_Design_MO` using as argument the name of the problem input file. The results are stored in the file `RESULTS_MO_DESIGN.mat`.
- For simulation and circuit scheme plot: from the main SYNBAD directory, call `SYNBAD_Simulate` using as argument the name of the problem input file.
- For plotting Pareto front of solutions: from the main SYNBAD directory, call `SYNBAD_Pareto_Plot` using as arguments the name of the problem input file and the name of the mat file storing the results of the multi-objective optimization.

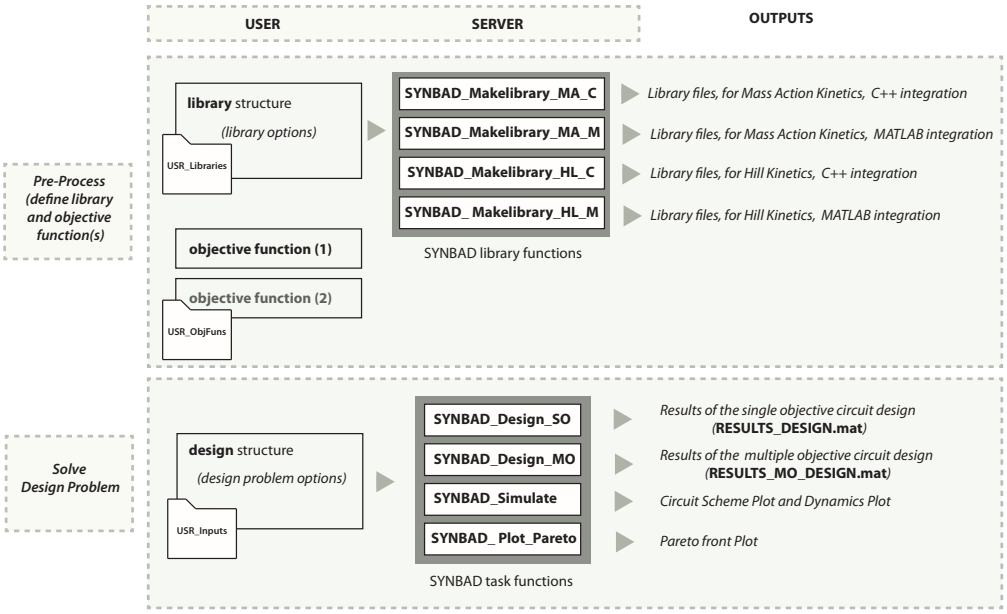

Figure 1: Scheme of the toolbox.

3 How to create a library of components

Library files are stored in the folder USR_Libraries. Two types of kinetics are supported: Mass Action and Hill kinetics.

In libraries using Mass Action kinetics, protein and mRNA dynamics are considered, and every reaction is endowed with mass action kinetics (the kinetic formalism is taken from [8], more details in Appendix A1).

In libraries using Hill kinetics, only protein dynamics are considered, and kinetics of Hill type are assumed (the kinetic formalism is taken from [1], more details in Appendix A2).

IMPORTANT: The current version of SYNBADm supports only constitutive and negative regulation of the promoters. For further extensions, the SYNBADm library functions (see Fig. 1) must be modified.

3.1 How to create a library of the Hill type

- Go to the folder HL_Library.

- Using a library input file, set the content of each field of the **library** structure, including the short name to identify the library files (we use HL here as a short name to identify the library files). Save the library input file.

- Call the corresponding SYNBADm library function with the name of the library input file as an argument to create the library files (including the odefile with the differential equations). For example, if the name of the library input file is `HL_input_library`, run:

  `»SYNBAD_Makelibrary_HL_M('HL_input_library')` to generate Matlab library code (for integration with Matlab IVP solvers).

  `»SYNBAD_Makelibrary_HL_C('HL_input_library')` to generate C++ library code (for integration with *CVODES*, recommended).

- The ordinary differential equations generated can be checked viewing the generated `HL_odefile`.

- Values for initial conditions can be set in the `HL_default_states.m` file by overwriting the default values (and saving the file with a different name).

- Values of parameters can be set in the `HL_default_parameters.m` file by overwriting the default values (and saving the file with a different name).

3.1.1 Hill type library options

Within the library input file, a number of library options are defined assigning values to the following fields of the structure **library**:

- **name_of_function:** Short name to identify the library files.

- **promoters:** Row cell array of strings containing the names of the promoters.

- **transcripts:** Row cell array of strings containing the names of the protein coding regions.

- **prom_tf:** List of transcripts repressing each promoter.

- **prom_nhill:** Hill coefficient for each repressor-promoter pair (row vector with as many components as promoters).

- **inducers:** Row cell array of strings containing the names of the inducers.

- **ind_tr:** Row cell array of strings containing the names of the transcript being bound by each inducer.

See the script `HL_input_library.m` in subsection 3.1.3 for an example.

3.1.2 Hill type library files

The SYNBADm library functions generate, within the folder HL_Library, the following files (here HL is used as a short name to identify the library files):

- `HL_transcripts_and_promoters.m`: Number of transcripts and promoters.

- `HL_odefile.m, HL_odefile_c.c`: Differential equations describing the dynamics of the system (Matlab and C++ versions respectively).

- `HL_default_states.m`: Default values for the initial conditions. The user can modify the values and save the folder with a different name, for example, `HL_default_states_1.m`.

- `HL_default_parameters.m`: Default values for the parameters. The user can modify the values and save the folder with a different name, for example, `HL_default_parameters_1.m`.

3.1.3 Default Hill type Library (HL)

The default HL library contains (see Fig. 2):

- 8 promoters: P_{lac1}, P_{Plac2}, P_{Plac3}, P_{Plac4}, P_{λ}, P_{Ptet1}, P_{Ptet2}, P_{araC}.

- 4 transcripts (repressors): $tetR$, $lacI$, cI, $araC$ ($LacI$ represses P_{lac1}, P_{lac2}, P_{lac3} and P_{lac4}; cI represses P_{λ}; $tetR$ represses P_{tet1} and P_{tet2}; and $araC$ represses P_{araC}).

- 2 inducers: $IPTG$, aTc ($IPTG$ binds $lacI$ while aTc binds $tetR$).

The input library file used to generate such a library is defined in `HL_input_library.m`

HL_input_library.m script

```
library.Promoters={'Plac1','Plac2','Plac3','Plac4','Plambda','Ptet1', ...
'Ptet2','ParaC'};
library.Transcripts={'tetR','lacI','cI','araC'};
library.Prom_tf={'lacI','lacI','lacI','lacI','cI','tetR','tetR','araC'};
library.Prom_nhill=[4,4,4,4,2,2,2,2];
library.Inducers={'IPTG','aTc'};
library.Ind_tr={'lacI','tetR'};
```

To check the dynamic equations, we can open the corresponding Matlab odefile (see the table below).

differential equations within HL_odefile.m script generated by SYNBADm

```
function dzdt = HL_odefile(t,z,parstr)
...
dtetR= +Y(1,1)*alpha_lacI/(1+K_Plac1*lacI^4) ...
 +Y(1,2)*alpha_lacI/(1+K_Plac2*lacI^4) ...
 +Y(1,3)*alpha_lacI/(1+K_Plac3*lacI^4) ...
 +Y(1,4)*alpha_lacI/(1+K_Plac4*lacI^4) ...
 +Y(1,5)*alpha_cI/(1+K_Plambda*cI^2) ...
 +Y(1,6)*alpha_tetR/(1+K_Ptet1*tetR^2) ...
 +Y(1,7)*alpha_tetR/(1+K_Ptet2*tetR^2) ...
 +Y(1,8)*alpha_araC/(1+K_ParaC*araC^2) ...
 -kf_tetRaTc*tetR*aTc ...
 +kb_tetRaTc*tetRaTc ...
 -kdeg_tetR*tetR;
dlacI= +Y(2,1)*alpha_lacI/(1+K_Plac1*lacI^4) ...
 +Y(2,2)*alpha_lacI/(1+K_Plac2*lacI^4) ...
 +Y(2,3)*alpha_lacI/(1+K_Plac3*lacI^4) ...
 +Y(2,4)*alpha_lacI/(1+K_Plac4*lacI^4) ...
 +Y(2,5)*alpha_cI/(1+K_Plambda*cI^2) ...
 +Y(2,6)*alpha_tetR/(1+K_Ptet1*tetR^2) ...
 +Y(2,7)*alpha_tetR/(1+K_Ptet2*tetR^2) ...
 +Y(2,8)*alpha_araC/(1+K_ParaC*araC^2) ...
 -kf_lacIIPTG*lacI*IPTG ...
 +kb_lacIIPTG*lacIIPTG ...
 -kdeg_lacI*lacI;
dcI= +Y(3,1)*alpha_lacI/(1+K_Plac1*lacI^4) ...
 +Y(3,2)*alpha_lacI/(1+K_Plac2*lacI^4) ...
 +Y(3,3)*alpha_lacI/(1+K_Plac3*lacI^4) ...
 +Y(3,4)*alpha_lacI/(1+K_Plac4*lacI^4) ...
 +Y(3,5)*alpha_cI/(1+K_Plambda*cI^2) ...
 +Y(3,6)*alpha_tetR/(1+K_Ptet1*tetR^2) ...
 +Y(3,7)*alpha_tetR/(1+K_Ptet2*tetR^2) ...
```

```
 +Y(3,8)*alpha_araC/(1+K_ParaC*araC^2) ...
 -kdeg_cI*cI;
daraC= +Y(4,1)*alpha_lacI/(1+K_Plac1*lacI^4) ...
 +Y(4,2)*alpha_lacI/(1+K_Plac2*lacI^4) ...
 +Y(4,3)*alpha_lacI/(1+K_Plac3*lacI^4) ...
 +Y(4,4)*alpha_lacI/(1+K_Plac4*lacI^4) ...
 +Y(4,5)*alpha_cI/(1+K_Plambda*cI^2) ...
 +Y(4,6)*alpha_tetR/(1+K_Ptet1*tetR^2) ...
 +Y(4,7)*alpha_tetR/(1+K_Ptet2*tetR^2) ...
 +Y(4,8)*alpha_araC/(1+K_ParaC*araC^2) ...
 -kdeg_araC*araC;
dlacIIPTG= +kf_lacIIPTG*lacI*IPTG ...
 -kb_lacIIPTG*lacIIPTG ...
 -kdeg_lacIIPTG*lacIIPTG;
dtetRaTc= +kf_tetRaTc*tetR*aTc ...
 -kb_tetRaTc*tetRaTc ...
 -kdeg_tetRaTc*tetRaTc;
...
```

3.2 How to create a library of the Mass Action type

- Go to the folder MA_Library.

- Using a library input file, set the content of each field of the **library** structure, including the short name to identify the library files (we use MA here as a short name to identify the library files). Save the library input file

- Call the corresponding SYNBADm library function with the name of the library input file as an argument to create the library files (including the odefile with the differential equations). For example, if the name of the library input file is `MA_input_library`, run:

  `»SYNBAD_Makelibrary_MA_M('MA_input_library')` to generate Matlab library code (for integration with Matlab IVP solvers).

  `»SYNBAD_Makelibrary_MA_C('MA_input_library')` to generate C++ library code (for integration with *CVODES*, recommended for faster performance).

- The ordinary differential equations generated can be checked at the generated odefile.

- Values of initial conditions can be set at the `MA_default_states.m` file, by overwriting the default values (and saving the file with a different name).

- Values of parameters can be set at the `MA_default_parameters.m` file by overwriting the default values (and saving the file with a different name).

3.2.1 Mass action type library options

In the library input file, the different options are defined assigning values to the following fields of the structure **library**:

- **name_of_function:** Short name to identify the library files.
- **promoters:** Row cell array of strings containing the names of the promoters.
- **transcripts:** Row cell array of strings containing the names of the protein coding regions.
- **prom_tf:** List of transcripts repressing each promoter.
- **inducers:** Row cell array of strings containing the names of the inducers.
- **ind_tr:** Row cell array of strings containing the names of the transcript being bound by each inducer.

See the script `MA_input_library.m` in subsection 3.2.3 for an example.

3.2.2 Mass action type library files

The SYNBADm library functions generate, within the folder MA_Library, the following files (here MA is used as a short name to identify the library files):

- `MA_transcripts_and_promoters.m`: Number of transcripts and promoters
- `MA_odefile.m, MA_odefile_c.c`: Differential equations describing the dynamics of the system (Matlab and C++ versions, respectively).

- `MA_default_states.m`: Default values for the initial conditions. The user can modify the values and save the folder with a different name, for example, `MA_default_states_1.m`

- `MA_default_parameters.m`: Default values for the parameters. The user can modify the values and save the folder with a different name, for example, `MA_default_parameters_1.m`

3.2.3 Default Mass Action type (MA) Library

The default MA library contains (see Fig. 3):

- 4 promoters: $P1$, $P2$, $P3$, $P4$ (corresponding to P_λ, P_{tet}, P_{arac} (denoted also as P_{bad}) and P_{lacI}).

- 11 transcripts (repressors): cI, $tetR$, $araC$, $lacI$, $luxI$, $luxR$, $lasI$, $lasR$, $ccdB$, $ccdA$ and $ccdA2$ (cI represses $P1$, $tetR$ represses $P2$, $araC$ represses $P3$ and $LacI$ represses $P4$).

The input library file used to generate such a library is defined `in MA_input_library.m`.

`MA_input_library.m` script

```
library.Promoters={'P1',P2','P3',P4' };
library.Transcripts={'cI','tetR','araC','lacI','luxI','luxR','lasI', ...
'lasR','ccdB','ccdA','ccdA2'};
library.Prom_tf={'cI','tetR','araC','lacI'};
library.Inducers={};
library.Ind_tr={};
```

To check the dynamic equations, we can open the corresponding Matlab odefile (see the table below).

differential equations within `MA_odefile.m` script generated by SYNBADm

```
function dzdt = MA_odefile(t,z,parstr)
...
% reaction rates
rf_pt(1)=kf_pt_1*P1*cI;
rb_pt(1)=kb_pt_1*P1cI;
rdeg_pt(1)=kdeg_pt_1*P1cI;
rf_pt(2)=kf_pt_2*P2*tetR;
rb_pt(2)=kb_pt_2*P2tetR;
rdeg_pt(2)=kdeg_pt_2*P2tetR;
rf_pt(3)=kf_pt_3*P3*araC;
rb_pt(3)=kb_pt_3*P3araC;
```

```
rdeg_pt(3)=kdeg_pt_3*P3araC;
rf_pt(4)=kf_pt_4*P4*lacI;
rb_pt(4)=kb_pt_4*P4lacI;
rdeg_pt(4)=kdeg_pt_4*P4lacI;
rtransc(1)=ktransc_1*P1cI;
rleak(1)=kleak_1*P1;
rtransc(2)=ktransc_2*P2tetR;
rleak(2)=kleak_2*P2;
rtransc(3)=ktransc_3*P3araC;
rleak(3)=kleak_3*P3;
rtransc(4)=ktransc_4*P4lacI;
rleak(4)=kleak_4*P4;
rtransl(1)=ktransl_1*cIm;
rdeg_m(1)=kdeg_m_1*cIm;
rdeg(1)=kdeg_1*cI;
rtransl(2)=ktransl_2*tetRm;
rdeg_m(2)=kdeg_m_2*tetRm;
rdeg(2)=kdeg_2*tetR;
rtransl(3)=ktransl_3*araCm;
rdeg_m(3)=kdeg_m_3*araCm;
rdeg(3)=kdeg_3*araC;
rtransl(4)=ktransl_4*lacIm;
rdeg_m(4)=kdeg_m_4*lacIm;
rdeg(4)=kdeg_4*lacI;
rtransl(5)=ktransl_5*luxIm;
rdeg_m(5)=kdeg_m_5*luxIm;
rdeg(5)=kdeg_5*luxI;
rtransl(6)=ktransl_6*luxRm;
rdeg_m(6)=kdeg_m_6*luxRm;
rdeg(6)=kdeg_6*luxR;
rtransl(7)=ktransl_7*lasIm;
rdeg_m(7)=kdeg_m_7*lasIm;
rdeg(7)=kdeg_7*lasI;
rtransl(8)=ktransl_8*lasRm;
rdeg_m(8)=kdeg_m_8*lasRm;
rdeg(8)=kdeg_8*lasR;
rtransl(9)=ktransl_9*ccdBm;
rdeg_m(9)=kdeg_m_9*ccdBm;
deg(9)=kdeg_9*ccdB;
rtransl(10)=ktransl_10*ccdAm;
rdeg_m(10)=kdeg_m_10*ccdAm;
rdeg(10)=kdeg_10*ccdA;
rtransl(11)=ktransl_11*ccdA2m;
rdeg_m(11)=kdeg_m_11*ccdA2m;
```

```
rdeg(11)=kdeg_11*ccdA2;

% ODEs
dP1=-max(Y(:,1))*rf_pt(1) + max(Y(:,1))*rb_pt(1) + max(Y(:,1))*rdeg_pt(1);
dP1cI=max(Y(:,1))*rf_pt(1) - max(Y(:,1))*rb_pt(1) - max(Y(:,1))*rdeg_pt(1);
dP2=-max(Y(:,2))*rf_pt(2) + max(Y(:,2))*rb_pt(2) + max(Y(:,2))*rdeg_pt(2);
dP2tetR=max(Y(:,2))*rf_pt(2) - max(Y(:,2))*rb_pt(2) - max(Y(:,2))*rdeg_pt(2);
dP3=-max(Y(:,3))*rf_pt(3) + max(Y(:,3))*rb_pt(3) + max(Y(:,3))*rdeg_pt(3);
dP3araC=max(Y(:,3))*rf_pt(3) - max(Y(:,3))*rb_pt(3) - max(Y(:,3))*rdeg_pt(3);
dP4=-max(Y(:,4))*rf_pt(4) + max(Y(:,4))*rb_pt(4) + max(Y(:,4))*rdeg_pt(4);
dP4lacI=max(Y(:,4))*rf_pt(4) - max(Y(:,4))*rb_pt(4) - max(Y(:,4))*rdeg_pt(4);
dcI=-max(Y(:,1))*rf_pt(1) + max(Y(:,1))*rb_pt(1)+ max(Y(1,:))*rtransl(1) ...
     - max(Y(1,:))*rdeg(1);
dcIm=-max(Y(1,:))*rdeg_m(1) ...
 + Y(1,1)*rtransc(1) + Y(1,1)*rleak(1)...
 + Y(1,2)*rtransc(2) + Y(1,2)*rleak(2)...
 + Y(1,3)*rtransc(3) + Y(1,3)*rleak(3)...
 + Y(1,4)*rtransc(4) + Y(1,4)*rleak(4);
dtetR=-max(Y(:,2))*rf_pt(2) + max(Y(:,2))*rb_pt(2)+ max(Y(2,:))*rtransl(2) ...
     - max(Y(2,:))*rdeg(2);
dtetRm=-max(Y(2,:))*rdeg_m(2) ...
 + Y(2,1)*rtransc(1) + Y(2,1)*rleak(1)...
 + Y(2,2)*rtransc(2) + Y(2,2)*rleak(2)...
 + Y(2,3)*rtransc(3) + Y(2,3)*rleak(3)...
 + Y(2,4)*rtransc(4) + Y(2,4)*rleak(4);
daraC=-max(Y(:,3))*rf_pt(3) + max(Y(:,3))*rb_pt(3)+ max(Y(3,:))*rtransl(3) ...
     - max(Y(3,:))*rdeg(3);
daraCm=-max(Y(3,:))*rdeg_m(3) ...
 + Y(3,1)*rtransc(1) + Y(3,1)*rleak(1)...
 + Y(3,2)*rtransc(2) + Y(3,2)*rleak(2)...
 + Y(3,3)*rtransc(3) + Y(3,3)*rleak(3)...
 + Y(3,4)*rtransc(4) + Y(3,4)*rleak(4);
dlacI=-max(Y(:,4))*rf_pt(4) + max(Y(:,4))*rb_pt(4)+ max(Y(4,:))*rtransl(4) ...
     - max(Y(4,:))*rdeg(4);
dlacIm=-max(Y(4,:))*rdeg_m(4) ...
 + Y(4,1)*rtransc(1) + Y(4,1)*rleak(1)...
 + Y(4,2)*rtransc(2) + Y(4,2)*rleak(2)...
 + Y(4,3)*rtransc(3) + Y(4,3)*rleak(3)...
 + Y(4,4)*rtransc(4) + Y(4,4)*rleak(4);
dluxI=max(Y(5,:))*rtransl(5) - max(Y(5,:))*rdeg(5);
dluxIm=-max(Y(5,:))*rdeg_m(5) ...
 + Y(5,1)*rtransc(1) + Y(5,1)*rleak(1)...
 + Y(5,2)*rtransc(2) + Y(5,2)*rleak(2)...
 + Y(5,3)*rtransc(3) + Y(5,3)*rleak(3)...
```

```
 + Y(5,4)*rtransc(4) + Y(5,4)*rleak(4);
dluxR=max(Y(6,:))*rtransl(6) - max(Y(6,:))*rdeg(6);
dluxRm=-max(Y(6,:))*rdeg_m(6) ...
 + Y(6,1)*rtransc(1) + Y(6,1)*rleak(1)...
 + Y(6,2)*rtransc(2) + Y(6,2)*rleak(2)...
 + Y(6,3)*rtransc(3) + Y(6,3)*rleak(3)...
 + Y(6,4)*rtransc(4) + Y(6,4)*rleak(4);
dlasI=max(Y(7,:))*rtransl(7) - max(Y(7,:))*rdeg(7);
dlasIm=-max(Y(7,:))*rdeg_m(7) ...
 + Y(7,1)*rtransc(1) + Y(7,1)*rleak(1)...
 + Y(7,2)*rtransc(2) + Y(7,2)*rleak(2)...
 + Y(7,3)*rtransc(3) + Y(7,3)*rleak(3)...
 + Y(7,4)*rtransc(4) + Y(7,4)*rleak(4);
dlasR=max(Y(8,:))*rtransl(8) - max(Y(8,:))*rdeg(8);
dlasRm=-max(Y(8,:))*rdeg_m(8) ...
 + Y(8,1)*rtransc(1) + Y(8,1)*rleak(1)...
 + Y(8,2)*rtransc(2) + Y(8,2)*rleak(2)...
 + Y(8,3)*rtransc(3) + Y(8,3)*rleak(3)...
 + Y(8,4)*rtransc(4) + Y(8,4)*rleak(4);
dccdB=max(Y(9,:))*rtransl(9) - max(Y(9,:))*rdeg(9);
dccdBm=-max(Y(9,:))*rdeg_m(9) ...
 + Y(9,1)*rtransc(1) + Y(9,1)*rleak(1)...
 + Y(9,2)*rtransc(2) + Y(9,2)*rleak(2)...
 + Y(9,3)*rtransc(3) + Y(9,3)*rleak(3)...
 + Y(9,4)*rtransc(4) + Y(9,4)*rleak(4);
dccdA=max(Y(10,:))*rtransl(10) - max(Y(10,:))*rdeg(10);
dccdAm=-max(Y(10,:))*rdeg_m(10) ...
 + Y(10,1)*rtransc(1) + Y(10,1)*rleak(1)...
 + Y(10,2)*rtransc(2) + Y(10,2)*rleak(2)...
 + Y(10,3)*rtransc(3) + Y(10,3)*rleak(3)...
 + Y(10,4)*rtransc(4) + Y(10,4)*rleak(4);
dccdA2=max(Y(11,:))*rtransl(11) - max(Y(11,:))*rdeg(11);
dccdA2m=-max(Y(11,:))*rdeg_m(11) ...
 + Y(11,1)*rtransc(1) + Y(11,1)*rleak(1)...
 + Y(11,2)*rtransc(2) + Y(11,2)*rleak(2)...
 + Y(11,3)*rtransc(3) + Y(11,3)*rleak(3)...
 + Y(11,4)*rtransc(4) + Y(11,4)*rleak(4) ;
```

3.3 Encoding a gene circuit structure in a vector of binary variables

A ***library of components*** contains a number P of promoters and a number T of transcripts. Each promoter-transcript pair defines a ***device***. The number of composable devices from the elements in the library is $N = P \times T$.

Once the files of a library are generated, SYNBADm automatically assigns an index to each device (i.e., to each promoter-transcript pair). A gene circuit is composed of devices (from 1 to N devices).

The structure of a circuit is defined by a binary N-vector y such that:

- $y(i) = 1$ if the promoter-transcript pair with index i is active in the circuit.
- $y(i) = 0$ else.

Starting from the structure vector, by computing `Yt = reshape(y,P,T)`, `Y=Yt'` we obtain the $T \times P$ superstructure matrix where:

- $Y(i,j) = 1$ if the pair constituted by transcript i and promoter j is active
- $Y(i,j) = 0$ else.

The superstructure matrix of a given circuit (as well as the circuit dynamics) is depicted using the `SYNBAD_Simulate` function.

In Fig. 2 we indicate how a circuit is encoded in a vector of binary variables (in this case using the default library of the Hill type).

Note that the binary vector y completely defines the structure and connectivity of the circuit. In this case, devices 1 and 15 are active, i.e. $P_{lac1} - tetR$ and $P_{tet2} - lacI$. Since $tetR$ represses P_{tet2} and $lac1$ represses P_{lac1} (as it is specified in the library options, see Section 3.1.3), we obtain the circuit scheme depicted in Fig. 2.

In Fig. 3 we give another example of how a circuit is encoded in a vector of binary variables (in this case using the default library of the Mass Action type). Note that the binary vector y completely defines the structure and connectivity of the circuit. In this case, devices 2, 8 and 13 are active, i.e. $P_{tet} - cI$, $P_{lacI} - tetR$ and $P_\lambda - lacI$. Since $lacI$ represses P_{lacI}, $tetR$ represses P_{tet}, and cI represses P_λ (as it is specified in the library options, see Section 3.2.3), we obtain the circuit scheme depicted in Fig. 3.

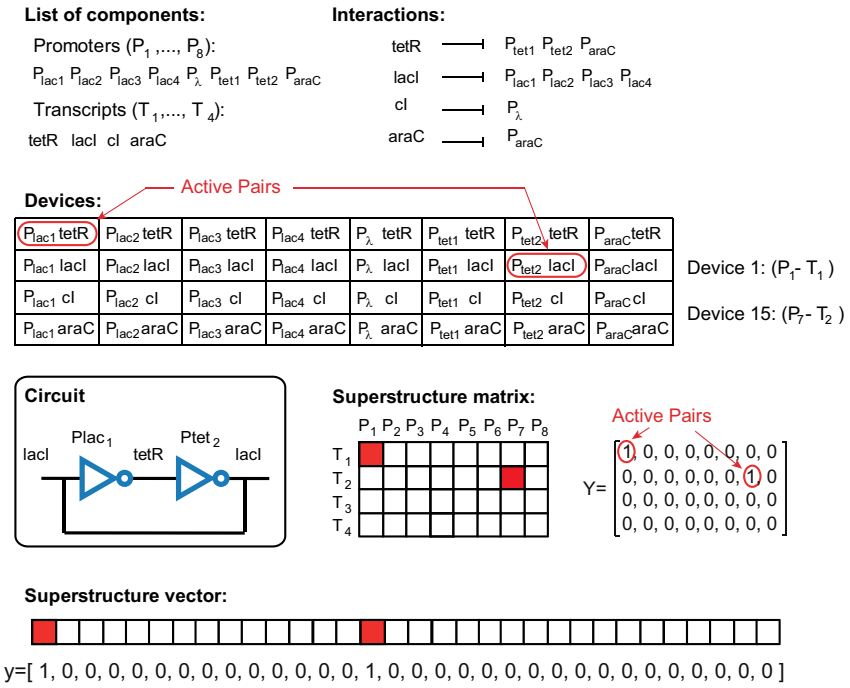

Figure 2: Example circuit representation (default HL library). The structure of a circuit is stored in a binary vector.

4 How to define the objective functions

Objective functions (i.e. performance criteria) are defined using specific scripts located in the USR_ObjFuns folder. We recommend to use one of the objective function files provided as a template. The objective function script must contain the following sections:

1. Declaration: `function[f,g]=OF_function(vy,opstr)` declares a function named OF_function accepting as arguments the decision vector *vy* (internal SYNBADm notation) and the structure **opstr** (internal SYNBADm structure).

2. Header block: Fixed block for assignment of variables.

3. Integration block: Includes the call to the IVP solvers (i.e. ODE integrator).

4. Performance index definition `f=...` Definition of the function used to evaluate the performance of the system (objective function).

5. Constraints for maximum and minimum number of devices.

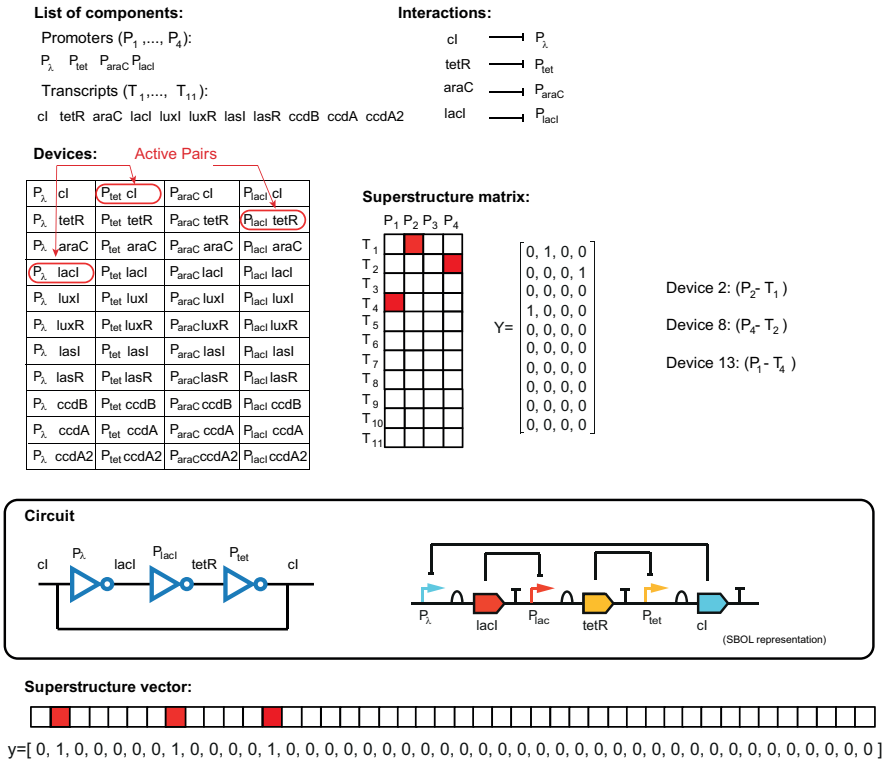

Figure 3: Example circuit representation (default MA library). The structure of a circuit is stored in a binary vector.

Next we see an example on how to construct an objective function script denoted `OF_function.m`. (The name of the script must be the same as the name of the function).

1. Declaration

```
function [f,g] = OF_function(vy,opstr)
```

2. Header block

```
idx = opstr.idx;
par = opstr.par;
n_real = opstr.n_real;
n_int = opstr.n_int;
x = vy(1:n_real);
y_int = vy(n_real+1:n_real+n_int);
y_bin = vy(n_real+n_int+1:end);
y = [y_int;y_bin];
k = par.value;
for ii=1:1:size(idx,2)
 k(idx{ii})=x(ii);
end
D_max = opstr.D_max;
D_min = opstr.D_min;
```

3. Integration block

Initial conditions for IVP (mass action type library)

```
Y=reshape(y,n_Promoters,n_Transcripts);
Y=Y';
eval(sprintf('states= %s(Y);',opstr.def_states));
z0=states.z0;
```

Initial conditions for IVP (Hill type library)

```
eval(sprintf('states= %s;',opstr.def_states));
z0=states.z0;
```

Parameters, inputs and structure binary variables

```
parstr.k = k;
parstr.y = y;
parstr.u = [40, 0];
```

Integration tspan

```
tspan = 0:0.1:1000;
```

Call the initial value problem ()IVP) solver (ODE integrator)

```
atol = opstr.ivpsol_atol;
rtol = opstr.ivpsol_rtol;
ivpopt = odeset('RelTol',rtol,'AbsTol',atol);
if opstr.name_odefile(end)=='c'
[t,z]=SYNBAD_CVODES(opstr.name_odefile,tspan,z0,parstr,rtol,atol,Inf);
else
[t,z] = ode15s(odefile_f,tspan,z0,ivopt,parstr);
end
```

4. Performance index

```
f=z(end)  % here you define the score (performance) function
```

5. Constrains for maximum and minimum number of devices (fixed block)

```
g(1)=+D_max-sum(y);
g(2)=-D_min-sum(y);
```

5 How to define the design problem

In order to solve design problems with SYNBADm, a structure **inputs** must be defined. We recommend to define the structure in a problem input file. The structure **inputs** contains the specifications of the design problem, including:

- Model specifications (name of the odefile, files containing parameter and initial values, number of variables, etc), contained in the substructure **model**.
- Design specifications (upper and lower bounds for the variables, allowed minimum and maximum number of devices), contained in the substructures **design** and **modesign**.
- Simulation specifications, declared in the substructure **simulate**.
- Options for the optimization, declared in the substructure **optsol**.
- Options for the integration, declared in the substructure **ivpsol**.

Next we describe the fields of the structure **inputs** in detail, with their corresponding substructures.

5.1 Model specifications

The structure **model** contains the following fields:

- **lib_type:** Type of library, choose 'MA_library', or 'HL_library'.
- **ode_name:** Name of the odefile, it must end in *_c* for C++ odefile.
- **n_real_var:** Number of real (continuous) decision variables.
- **n_integer_var:** Number of integer decision variables.

- **n_binary_var:** Number of binary decision variables (it is the number of promoters multiplied by the number of transcripts in the library).

- **def_param_function:** Name of the file containing the values of the parameters.

- **def_state_function:** Name of the file containing the values of the initial conditions.

- **transc_promot_function:** Name of the file containing the number of transcripts and promoters.

- **u_values:** Vector containing the numerical value of the inputs.

5.2 Design specifications

Next we define the fields contained in the structure **design**. First, one of the following cells **idx** or **par_x** is used to indicate which variables of the real type are going to be selected as decision variables.

- **idx:** Cell array of vectors where idx{i} is a vector containing the indices of the parameters corresponding to the real decision variable i.

- **par_x:** Cell array of strings where par_x{i} is a cell of strings containing the names of the parameters corresponding to the real decision variable i.

Remember that only one of the fields is completed, and the other is left empty (= [];). If no real variables are selected as decision variables, both fields are left empty.

- **var_L:** Vector containing the lower bounds for the decision variables (in the following order: real, integer, binary).

- **var_U:** Vector containing the upper bounds for the decision variables (in the following order: real, integer, binary).

- **var_0:** Vector containing the initial guess for the optimization (in the following order: real, integer, binary).

- **n_binary_var:** Number of binary decision variables (it is the number of promoters multiplied by the number of transcripts in the library).

- **D_max:** Number of allowed maximum number of devices in the circuit (applies for *MITS*, *ESS* and *ACOmi*).

- **D_min:** Number of allowed minimum number of devices in the circuit (applies for *MITS*, *ESS* and *ACOmi*).

For a single objective optimization problem, we provide the name of the objective function to use in the following subfield of the **design** structure:

- **objective:** Name of the script containing the objective function to use in the single optimization design problem.

For a multiple objective optimization problem, we use the field **modesign** of the **inputs** structure, with the following subfields:

- **objective1:** Name of the script containing the first objective function to use in the multiobjective optimization design problem. (The first objective function corresponds to J_1 in Fig. 4.)

- **objective2:** Name of the script containing the second objective function to use in the multiobjective optimization design problem. (The second objective function corresponds to J_2 in Fig. 4.)

- **min_objective_1:** Vector containing the values of objectives 1 and 2 evaluated at the optimum for the objective function 1 (i.e., $[\underline{J}_1, \overline{J}_2]$ according to Fig. 4).

- **min_objective_2:** Vector containing the values of objectives 1 and 2 evaluated at the optimum for the objective function 2 (i.e., $[\overline{J}_1, \underline{J}_2]$ according to Fig. 4).

IMPORTANT: check that the following inequality is fulfilled:

inputs.modesign.min_objective_1(2)>inputs.modesign.min_objective_2(2)

Finally, we set the number of intervals:

- **econstraint_nint:** Number of intervals (objective function 2 axis) for the epsilon-constraint strategy.

5.3 Simulation options

The field **simulate** contains the specifications in order to perform a dynamic simulation of a given biocircuit with `SYNBAD_Simulate`. The following subfields must be defined:

- **var_circuit:** Vector of decision variables for the circuit to simulate (in the following order: real, integer, binary).
- **tspan:** Vector specifying the time intervals for integration.
- **objective:** Cell containing the names of the objective functions to be evaluated for the selected circuit.

5.4 Optimization solver options

The options for the optimization solver are contained in the field **optsol**, with the following subfields:

- **optsolver:** Choose among 'ESS', 'MITS', 'ACO' or 'VNS'.
- **maxtime:** Maximum optimization time for the global solver (in seconds).

Local solver options: when using *ESS* or *ACO* solvers, always select 'misqp' as the local solver, using the following subfields of **optsol**, respectively:

- **ess.local.solver:** selects 'misqp' as the option for the local solver.
- **aco.local.solver:** selects 'misqp' as the option for the local solver.

5.5 Initial value problem (IVP) solver options

The options for the IVP solver (ODE integrator) are assigned in the field **IVPsol**, with the following subfields:

- **rtol:** Relative integration tolerance.
- **atol:** Absolute integration tolerance.

6 Tasks

6.1 Solving a single objective design problem

1. Define the structure **inputs** with the problem specifications in a problem input file (e.g. `problem_input_file.m`).

2. In the SYNBAD main directory, execute the Matlab command:

   `»SYNBAD_Design_SO('problem_input_file')`

3. The optimization problem is solved, and the results will be stored in the file `RESULTS_DESIGN.mat` (if you want to keep it, rename it and move it elsewhere in order to avoid overwriting in future computations).

The structure **results** will have, among other fields, the optimum value of the objective function found (**fbest** for *ESS*, **f** for *MITS*), and the optimal decision vector (**xbest** for ESS, **x** for *MITS*).

6.2 Solving a multiple objective design problem

6.2.1 Epsilon-constraint strategy

SYNBADm solves the MINLP multiple objective design problem using the epsilon-constraint strategy (see Fig. 4).

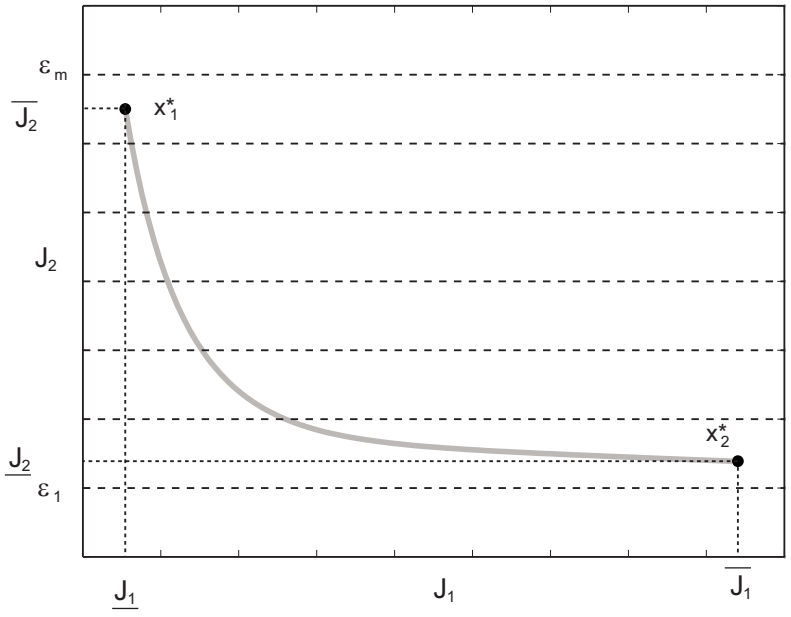

Figure 4: Scheme of the epsilon-constraint strategy.

The user chooses a primary objective function (objective function 1, or J_1), and a secondary objective function (objective function 2, or J_2).

The minima of objective function 1 (point V_1) and objective function 2 (point V_2) determine the extremes of the Pareto optimal front.

The algorithm works as follows: the objective function 1 is minimized by intervals using objective 2 values as constraints (upper and lower bounds for the intervals).

The number of intervals is given by the user (**inputs.modesign.econstraint_nint**).

6.2.2 Steps to solve a multiobjective problem

1. Solve the single objective problem for objective function 1 and store the results.

2. Solve the single objective problem for objective function 2 and store the results.

3. Complete the multiobjective problem specifications in the problem input file.

4. In the SYNBAD main directory, execute the Matlab command:

   `»SYNBAD_Design_MO('problem_input_file').`

5. Computations will proceed (it can take a while) and after completion, the results of the epsilon-constraint optimization will be stored in `RESULTS_MO_DESIGN.mat` (it is recommended to rename it and move it elsewhere in order to avoid overwriting by future computations). The file contains the cell **results**, where **results{i}** contains the results for each interval i.

To generate a figure with the Pareto front of solutions (i.e. the set of best trade-offs found), simply call `SYNBAD_Plot_Pareto` using as arguments the name of the problem input file and the name of the file containing the solution of the multiobjective optimization problem, as follows:

`»SYNBAD_Plot_Pareto('Problem_input_file', 'RESULTS_MO_DESIGN').`

6.3 How to simulate the dynamics of a biocircuit

In order to get the scheme of a given biocircuit and simulate its dynamics,

1. Define the simulation specifications in the problem input file.

2. In the SYNBAD main directory, call:

   `»SYNBAD_Simulate('problem_input_file')`

3. Two figures are obtained, the dynamics of the circuit and the matrix with the structure of the circuit. Besides, the selected objective function(s) is(are) evaluated.

7 Optimization Solvers

SYNBADm includes four global optimization solvers which are based on metaheuristics, combining stochastic global search with efficient local search method. Below we include further details and some basic recommendations for choosing the solver depending on the problem.

7.1 *eSS*: enhanced Scatter Search

- Mixed Integer Nonlinear Programming problems.
- It handles constraints.
- Incorporates local solver: *MISQP* mixed-integer sequential quadratic programming [3].
- More details in Egea *et al* [2].

-IMPORTANT: set the option `inputs.optsol.aco.local.solver = 'misqp'` in the problem input file.

7.2 *MITS*: Mixed Integer Tabu Search.

- Mixed Integer Nonlinear Programming problems.
- Incorporates the local solver *MISQP* mixed-integer sequential quadratic programming [3].
- More details in Exler *et al* [4].

7.3 *ACOmi*: Ant Colony Optimization for mixed integer domain.

- Mixed Integer Nonlinear Programming problems.

- Incorporates local solver: *MISQP* mixed-integer sequential quadratic programming [3].

- More details in Schlueter *et al* [9].

-IMPORTANT: set the option `inputs.optsol.aco.local.solver = 'misqp'` in problem input file

7.4 *VNS*: Variable Neighbourhood search.

- Integer Nonlinear Programming problems (it does not handle constraints).

- More details in Mladenovic *et al* [5].

7.5 Basic recommendations (how to choose the right solver)

-Use *VNS* in unconstrainted single objective problems with integer (or binary) variables only.

-For problems with constraints (e.g. minimum and maximum number of devices) and/or mixed integer-continuous variables, *eSS*, *MITS* or *ACOmi* can be used. The three of them are suitable for single and multiobjective design, but their comparative performance will be problem dependent. For new problems with no prior knowledge about the expected solution, it is a good practice to use all of them.

-In our experience, *eSS* showed a very good performance in synthetic gene circuit design independently of the balance of real/integer variables. *MITS* and *ACOmi* performed better for problems without (or with few) real decision variables. However, due to their stochastic nature, we recommend to solve each new problem with the three different solvers and compare results.

8 Initial value problem (IVP) solvers

For each evaluation of the objective functions, a dynamic simulation must be performed. That is, the outer optimization problem requieres the solution of an inner initial value problem (i.e. integration of the dynamics) for each evaluation. SYNBADm allows the use of several ODE integrators for dynamic simulation, including:

- Matlab default integrators (ode45, ode15s).
- *CVODES* [10]. In this case, the use of C models requires the previous installation of a compatible C++ compiler, as described more information at `http://es.mathworks.com/support/compilers/R2015b/index.html`

We strongly recommend the use of *CVODES* since it will result in much faster integrations. The Matlab integrators (such as ode15s) should only be used in those cases where the installation of a compatible C++ compiler was not possible.

9 Application examples

In this section we illustrate the use of SYNBADm for different problems. Note that the solutions obtained might differ slightly from run to run (due to the stochastic nature of the optimization solvers). The examples described below have been implemented in SYNBADm and can be solved automatically using the test-run scripts stored in the Examples folder (see Section 10).

9.1 Example 1: Optimal design of a switch-like circuit (binary variables-Hill kinetics)

Starting from the default Hill library (see Section 3.1.3 and Fig. 2) with 8 promoters and 4 transcripts, we aim to find circuits behaving in a switch-like manner such that the steady state level of $LacI$ is high upon aTc and low upon $IPTG$ induction, whereas the steady state level of $tetR$ is low upon aTc and high upon $IPTG$ induction (as in [1]). The kinetics is of Hill type for the lumped transcription-translation reactions.

We set a minimum (and maximum) of 3 devices (i.e. 3 active promoter-transcript pairs) in the target circuit.

To solve the problem using SYNBADm, we proceed as indicated next.

1. We use the default HL library (see the library input file with the specifications and the generated odefile in Section 3.1.3). Library files are already provided.

2. We keep the values of the initial conditions in `HL_default_values.m` (we save in a script `HL_default_states_1.m`), and modify the values of the default parameters `HL_default_parameters.m` (we save the new values in a file `HL_default_parameters_1.m`). The values of the parameters are taken from [1].

Parameter values in `HL_default_parameters_1.m`
`K_Plac1=10;`
`K_Plac2=0.01;`
`K_Plac3=0.001;`
`K_Plac4=0.00001;`
`K_Plambda=0.33;`
`K_Ptet1=0.014;`
`K_Ptet2=1.4;`
`K_ParaC=2.5;`
`alpha_tetR=1.215;`
`alpha_lacI=1.215;`
`alpha_cI=2.92;`
`alpha_araC=1.215;`
`kdeg_tetR=0.0346;`
`kdeg_lacI=0.0346;`
`kdeg_cI=0.0693;`
`kdeg_araC=0.0115;`
`kf_lacIIPTG=0.05;`
`kf_tetRaTc=0.05;`
`kb_lacIIPTG=0.1;`
`kb_tetRaTc=0.1;`
`kdeg_lacIIPTG=0.0693;`
`kdeg_tetRaTc=0.0693;`

3. We define the objective function in the script `OF_Switch.m`. The target behaviour is encoded in an objective function such that, the desired performance is achieved at the minimum value of the objective function. We consider the following function [1]:

$$f = -\frac{1}{2}\left(\frac{lacI\_aTc - lacI\_IPTG}{lacI\_aTc} + \frac{tetR\_IPTG - tetR\_aTc}{tetR\_IPTG}\right) \tag{1}$$

where $lacI\_aTc$, $lacI\_IPTG$ represent the steady state level of $lacI$ upon aTc and $IPTG$ induction respectively, and $tetR\_aTc$, $tetR\_IPTG$

represent the steady state level of $lacI$ upon aTc and $IPTG$ induction respectively. Note that the minimum value of f is in this case -1.

4. Complete problem input file: in the USR_Inputs folder we create an script `Example1.m` where we define the structure **inputs**. The contents of the script are given below.

`Example1.m` script

```
% Model options
inputs.model.lib_type ='HL_Library';
inputs.model.ode_name ='HL_odefile_c';
inputs.model.n_real_var = 0;
inputs.model.n_integer_var = 0;
inputs.model.n_binary_var = 32;
inputs.model.def_param_function= 'HL_default_parameters_1';
inputs.model.def_state_function= 'HL_default_states_1';
inputs.model.transc_promot_function= 'HL_transcripts_and_promoters';
inputs.model.u_values=[40;0];
% Design options
inputs.design.idx =[];
inputs.design.par_x=[];
inputs.design.var_L = zeros(1,32);
inputs.design.var_U = ones(1,32);
inputs.design.var_0 = zeros(1,32);
inputs.design.D_max = 3;
inputs.design.D_min = 3;
% Single objective design options
inputs.design.objective='OF_Switch';
% MINLP solver options
inputs.optsol.optsolver='MITS';
inputs.optsol.maxtime = 150;
% IVP solver options
inputs.ivpsol.rtol=1.0D-7;
inputs.ivpsol.atol=1.0D-7;
```

5. Single objective design: in the SYNBAD main folder we run

   `»SYNBAD_Design_SO('Example1')`

   the solution is stored in the file `RESULTS_DESIGN.mat`.

We obtain the solution vector:

```
x=[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
```

```
0];
```

The value of the objective function is:

```
f = -0.99999927345;
```

In order to simulate the circuit dynamics and obtain the matrix superstructure, we set the obtained solution vector, time integration interval and objective function to be evaluated in the problem input file `Example1.m`, as well as the desired values of the inputs ($IPTG$ and aTc):

Options for simulation of the optimal circuit in `Example1.m`

```
inputs.simulate.var_circuit = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0];
inputs.simulate.tspan = 0:1:1000;
inputs.simulate.objective={'OF_Switch'};
inputs.model.u_values=[40;0];
```

We obtain the plots in Fig. 5 A-B.

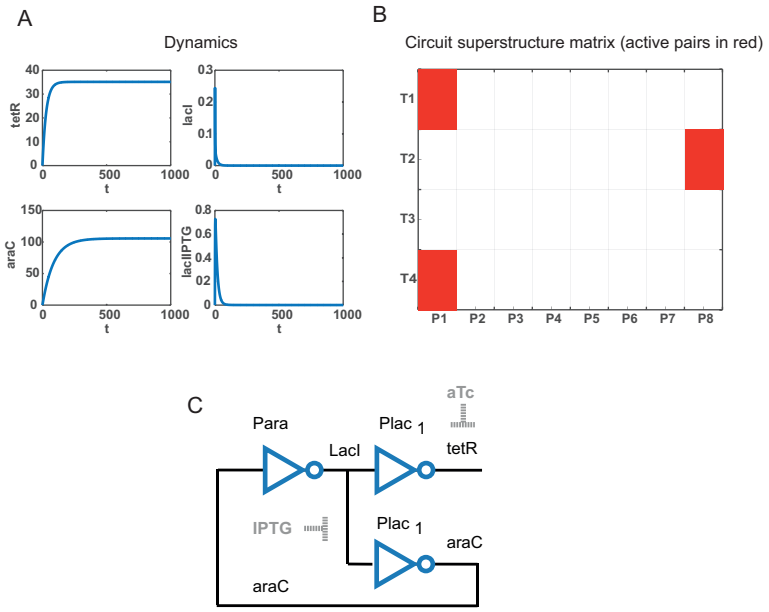

Figure 5: Optimal circuit found for Example 1, A) dynamics, B) circuit superstructure matrix, C) circuit scheme.

As it can be deduced from the superstructure matrix in Fig. 5B, the active pairs are $P_1 - T_1$, $P_1 - T_4$ and $P_8 - T_2$. The labeling of promoters and transcripts follow the order given in the library input file. Therefore, the active

pairs are $P_{lac1} - tetR$, $P_{lac1} - araC$ and $P_{araC} - lacI$. Since $araC$ represses P_{araC} and $lacI$ represses P_{lac1} (see `HL_input_library.m`), the circuit scheme corresponds to the one depicted in Fig 5C.

9.2 Example 2: Optimal design of a switch-like circuit (binary variables-Hill kinetics)

Now we aim to find the circuit with optimal switch-like performance (see Example 1) but without constraint in the number of devices allowed in the final circuit. For the unconstrained problem, we use the solver *VNS*.

The problem input file `Example2.m` is equal to `Example1.m` except from the options related to the optimization solver:

Optimization solver options in `Example2.m`

```
inputs.optsol.optsolver = 'VNS';
```

Remember that the options D_max and D_min do not apply for *VNS*. Proceeding as in the previous example we obtain the following solution:

```
x = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0
0 0]; f = -0.99999927346;
```

9.3 Example 3: Optimal design of a switch-like circuit (combined real and binary variables-Hill kinetics)

We aim to find the circuit of 3 devices with optimal switch-like performance (see Example 1), but taking as decision variables also the following parameters:

- Degradation constants (all with the same values): $kdeg\_tetR$, $kdeg\_lacI$, $kdeg\_cI$ and $kdeg\_araC$,
- Binding constant $kf\_tetRaTc$.

`Example3.m` script

```
% Model options
inputs.model.lib_type ='HL_Library';
inputs.model.ode_name ='HL_odefile_c';
inputs.model.n_real_var = 2;
inputs.model.n_integer_var = 0;
inputs.model.n_binary_var = 32;
```

```
inputs.model.def_param_function= 'HL_default_parameters_1';
inputs.model.def_state_function= 'HL_default_states_1';
inputs.model.transc_promot_function= 'HL_transcripts_and_promoters';
inputs.model.u_values=[40;0];
% Design options
inputs.design.idx ={[13,14,15,16],18;}
% inputs.design.par_x={{'kdeg_tetR','kdeg_lacI', ...
%'kdeg_cI','kdeg_araC'},{'kf_tetRaTc'}};
inputs.design.var_L = [0.05, 0.01, zeros(1,32)];
inputs.design.var_U = [0.1, 0.1, ones(1,32)];
inputs.design.var_0 = zeros(1,34);
inputs.design.D_max = 3;
inputs.design.D_min = 3;
% Single objective design options
inputs.design.objective='OF_Switch';
% MINLP solver options
inputs.optsol.optsolver='ESS';
inputs.optsol.ess.local.solver = 'misqp';
inputs.optsol.maxtime = 150;
% IVP solver options
inputs.ivpsol.rtol=1.0D-7;
inputs.ivpsol.atol=1.0D-7;
```

We obtain the following solution:

```
x = [2.965672738301116e-04 0.058059726411727 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]

f = -1
```

The first two entries of the vector x correspond to the optimal values for the real variables, i.e.:

$x(1) = kdeg\_tetR = kdeg\_lacI = kdeg\_cI = kdeg\_araC = 2.9656763e-04$

$x(2) = kf\_tetRaTc = 0.058059$

The remaining entries of the vector define the structure (y) of the optimal circuit:

$y = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$

9.4 Example 4: Optimal design of an oscillatory circuit (binary variables-mass action kinetics)

We aim to find endogenous oscillators starting from the default Mass Action library (see Section 3.2.3 and Fig. 3) with 11 transcripts and 4 promoters. Kinetics is of mass action type for transcription and translation reactions and mRNA dynamics is taken into account.

We allow for a minimum (and maximum) of 3 devices in the final circuit.

To solve the problem using SYNBADm, we proceed as follows:

1. We use the default MA library (see the library input file with the specifications and the generated odefile in Section 3.2.3).

2. We keep the values of the initial conditions in `MA_default_values.m` (we save in a script `MA_default_states_1.m`), and modify the values of the default parameters `MA_default_parameters.m` (we save the new values in a `file MA_default_parameters_1.m`). The values of the parameters are taken from [8].

Parameter values in `MA_default_parameters_1.m`

```
NA = 6.0221415e23;% Avogadro
V = 1e-14;% Cell volume
NAV = NA*V/1e9; % For concentration in nM
kf_pt_1=1*NAV;
kf_pt_2=1*NAV;
kf_pt_3=1*NAV;
kf_pt_4=1*NAV;
kb_pt_1=0.5;
kb_pt_2=0.5;
kb_pt_3=0.000001;
kb_pt_4=0.5;
kdeg_pt_1=0.075;
kdeg_pt_2=0.075;
kdeg_pt_3=0.075;
kdeg_pt_4=0.075;
ktransc_1=0.00005;
ktransc_2=0.00005;
ktransc_3=0.00001;
ktransc_4=0.00005;
kleak_1=0.12;
kleak_2=0.09;
kleak_3=0.1;
```

```
kleak_4=0.1;
ktransl_1=0.1;
ktransl_2=0.1;
ktransl_3=0.1;
ktransl_4=0.1;
ktransl_5=0.1;
ktransl_6=0.1;
ktransl_7=0.1;
ktransl_8=0.1;
ktransl_9=0.1;
ktransl_10=0.1;
ktransl_11=0.1;
kdeg_m_1=0.001;
kdeg_m_2=0.001;
kdeg_m_3=0.001;
kdeg_m_4=0.001;
kdeg_m_5=0.001;
kdeg_m_6=0.001;
kdeg_m_7=0.001;
kdeg_m_8=0.001;
kdeg_m_9=0.001;
kdeg_m_10=0.001;
kdeg_m_11=0.001;
kdeg_1=0.001;
kdeg_2=0.001;
kdeg_3=0.001;
kdeg_4=0.001;
kdeg_5=0.001;
kdeg_6=0.001;
kdeg_7=0.001;
kdeg_8=0.001;
kdeg_9=0.001;
kdeg_10=0.001;
kdeg_11=0.001;
```

3. Objective function: We define the objective function in the script `OF_Oscil.m`. The target behaviour is encoded in an objective function such that, the desired performance is achieved at the minimum value of the objective function. For a sustained oscillation, we maximize the first peak of the autocorrelation function [7]. The autocorrelation function is normalized such that the minimum value of the objective

function f is -1.

4. Problem input file: in the USR_Inputs folder we create an script `Example4.m` where we define the structure **inputs**. The script is described below.

`Example4.m` script

```
% Model options
inputs.model.lib_type ='MA_Library';
inputs.model.ode_name ='MA_odefile_c';
inputs.model.n_integer_var = 0;
inputs.model.n_real_var = 0;
inputs.model.n_binary_var = 44;
inputs.model.def_param_function= 'MA_default_parameters_1';
inputs.model.def_state_function= 'MA_default_states';
inputs.model.transc_promot_function = 'MA_transcripts_and_promoters';
inputs.model.u_values = [];
% Design options
inputs.design.objective = 'OF_Oscil';
inputs.design.idx = [];
inputs.design.par_x = [];
inputs.design.var_L = zeros(1,44);
inputs.design.var_U = ones(1,44);
inputs.design.var_0 = zeros(1,44);
inputs.design.D_max = 3;
inputs.design.D_min = 3;
% Optimization solver options
inputs.optsol.optsolver = 'ESS';
inputs.optsol.maxtime = 600;
inputs.optsol.ess.local.solver = 'misqp';
% IVP solver options
inputs.ivpsol.rtol = 1.0D-7;
inputs.ivpsol.atol = 1.0D-7;
```

5. Single objective design: in the SYNBAD main folder we run:

   `»SYNBAD_Design_SO('Example4')`

   the solution is stored in the file `RESULTS_DESIGN.mat`.

We obtain the solution vector:

```
x = [0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```

The value of the corresponding objective function is:

```
f = -0.5118;
```

In order to simulate the circuit dynamics and obtain the matrix superstructure, we set the obtained solution vector, tspan and objective function to be evaluated in the problem input file `Example4.m`.

Options for simulation of the optimal circuit in `Example4.m`

```
inputs.simulate.var_circuit = [0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
inputs.simulate.tspan = 0:10:40000;
inputs.simulate.objective={'OF_Oscil'};
inputs.model.u_values=[];
```

We obtain the plots in Fig. 6A-B.

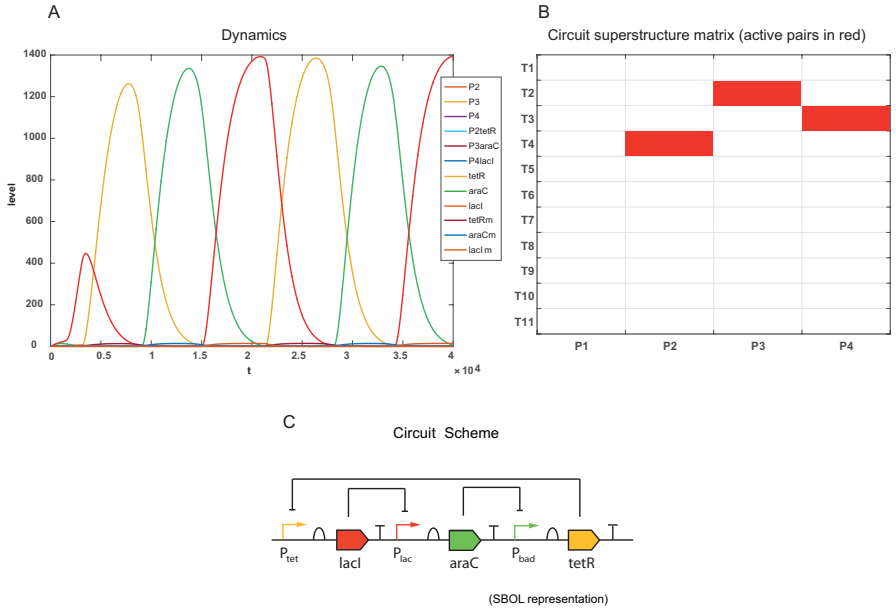

Figure 6: Optimal circuit found for Example 4, A) dynamics, B) circuit superstructure matrix, C) circuit scheme.

As it can be deduced from the superstructure matrix in Fig. 6B, the active pairs are $P_3 - T_2$, $P_4 - T_3$ and $P_2 - T_4$. The labeling of promoters and transcripts follow the order given in the library input file. Therefore, the active pairs are $P_{araC} - tetR$, $P_{lac1} - araC$ and $P_{tet} - lacI$. Since $tetR$ represses P_{tet}, $lacI$ represses P_{lacI} and $araC$ represses P_λ (see `MA_input_library.m`), the circuit scheme corresponds to the one depicted in Fig 6C.

9.5 Example 5: Multi-objective design optimizing switch-like performance and protein production cost)

Starting from the library in Example 1, we aim to find circuits behaving in a switch-like manner (as in Example 1), minimizing at the same time the protein production cost. We set a minimum (and maximum) of 3 devices (i.e. 3 active promoter-transcript pairs) in the target circuit.

To solve the problem using SYNBADm, we proceed as follows:

1. We define the objective function to minimize protein production cost in a function script `OF_Cost.m`.

2. We solve the single objective problem optimizing switch-like performance (see Example 1). We simulate for the obtained optimum to get the values of the objective functions 1 and 2 (using as option in the problem input file `inputs.simulate.objective={'OF_Switch', 'OF_Cost'};`).

3. We solve the single objective problem optimizing protein cost, and proceed as in the previous step to obtain the values of the objective functions 1 and 2 at the optimal circuit.

4. Set the options for multiobjective optimization at the problem input file `Example5.m`.

   Multiobjective design options in `Example5.m` script

   ```
   inputs.modesign.objective1 = 'OF_Switch';
   inputs.modesign.objective2 = 'OF_Cost';
   inputs.modesign.min_objective_1 = [-0.999999273459281 3.602033265810649e+03];
   inputs.modesign.min_objective_2 = [-0.959449867537731 259.147216801563];
   inputs.modesign.econstraint_nint = 7;
   ```

   It is important to check that **inputs.modesign.min_objective_1(2)** is greater than **inputs.modesign.min_objective_2(2)**.

5. From the SYNBAD main folder, call:

   ```
   » SYNBAD_Design_MO('Example5');
   ```

   the results of the multiobjective problem are stored in `RESULTS_MO_DESIGN.mat`, where the cell **results** contains the results of the optimization for every interval.

6. Plot the Pareto front by calling:

```
» SYNBAD_Plot_Pareto('Example5','RESULTS_MO_DESIGN');
```

We obtain the Pareto Front depicted in Fig. 7

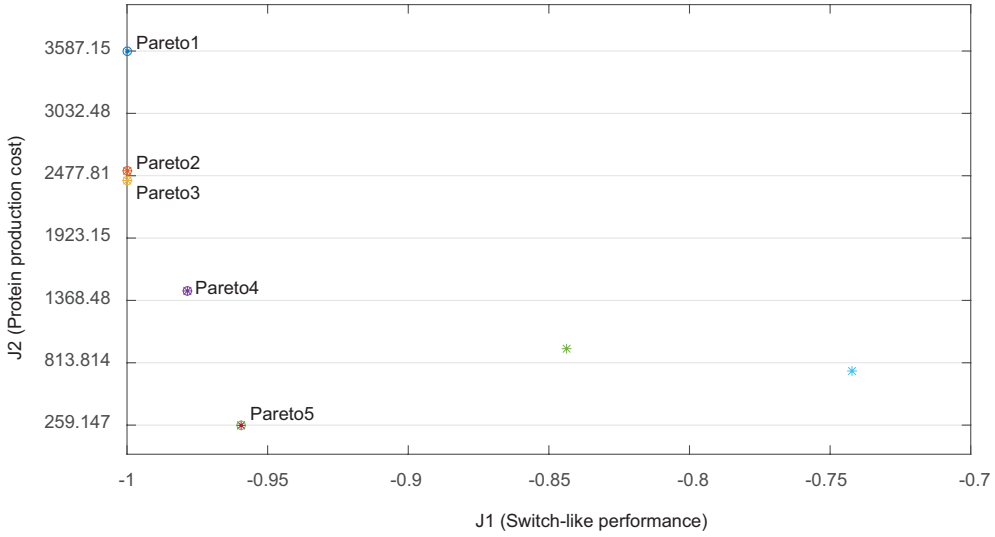

Figure 7: Solutions and Pareto Front for Example 5.

10 Test Examples

All the examples introduced in the previous section have been implemented in SYNBADm. The user can run the examples automatically from the Examples folder by typing:

»`Run_Example_1` (solves the example 1), see Section 9.1.

»`Run_Example_2` (solves the example 2), see Section 9.2.

»`Run_Example_3` (solves the example 3), see Section 9.3.

»`Run_Example_4` (solves the example 4), see Section 9.4.

»`Run_Example_5` (solves the example 5), see Section 9.5.

These examples can be used to test that everything is working properly.

Appendix A Reactions associated to biological devices

A.1 Mass Action type Library

The kinetic formalism is adapted/extended from [8]. Within this framework, all the reactions are endowed with mass action kinetics. A promoter $P1$ negatively regulated by a protein $T1$ has associated the reactions:

$$P1 + T1 \underset{k_u}{\overset{k_b}{\rightleftharpoons}} P1T1 \xrightarrow{k_{tb}} P1T1 + mT2 \tag{2}$$

where $P1$ is the promoter, $T1$ is the repressor protein, $P1T1$ is the repressor-promoter complex and $mT2$ is the $mRNA$ of the transcribed protein. The parameters k_b, k_u and k_{tb} are the protein-promoter binding rate constant, protein-promoter unbinding rate constant and the rate of transcription in the bound state. The reactions corresponding to a promoter not regulated by any transcription factor are:

$$P1 \xrightarrow{k_t} P1 + mT2 \qquad mT2 \xrightarrow{k_{dm}} \emptyset \tag{3}$$

where k_t is the constitutive rate of transcription in absence of transcription factors and k_{dm} is the degradation rate constant for the $mRNA$ degradation. In addition, we have translation:

$$mT2 \xrightarrow{k_r} mT2 + T2 \tag{4}$$

where k_r is the rate constant corresponding to the translation of $mRNA$, and degradation:

$$T2 \xrightarrow{k_d} \emptyset \qquad mT2 \xrightarrow{k_{dm}} \emptyset \tag{5}$$

where k_d and k_{dm} are the degradation rate constants of protein and $mRNA$ respectively.

Therefore, the set of reactions for the device $P1 - T2$ in presence of the repressor protein T_1 reads:

$$P1 + T1 \underset{k_u}{\overset{k_b}{\rightleftharpoons}} P1T1 \xrightarrow{k_{tb}} P1 + mT2$$

$$mT2 \xrightarrow{k_r} mT2 + T2 \qquad mT2 \xrightarrow{k_{dm}} \emptyset \qquad T_2 \xrightarrow{k_d} \emptyset \tag{6}$$

The presence of an external inducer binding repressor $T1$ will add the following reactions:

$$I + T1 \underset{k_{ui1}}{\overset{k_{bi1}}{\rightleftharpoons}} IT1 \xrightarrow{k_{di}} \emptyset$$

where k_{bi}, k_{ui} and k_{di} are the constants of binding, unbinding and degradation of the inducer complex, respectively.

A.2 Hill type Library

The kinetic formalism is adapted/extended from [1]. Within this framework, the device $P1-T2$, where $P1$ is a promoter negatively regulated by a protein $T1$, has associated the reactions:

$$P1 \xrightarrow{r_t} P1 + T2 \tag{7}$$

$$T2 \xrightarrow{k_d} \emptyset. \tag{8}$$

The first reaction has Hill-type kinetics, being the rate r_t of the lumped transcription and translation given by the expression:

$$r_t = \frac{\alpha_{p1}}{1 + K_{t1} T1^n} \tag{9}$$

where α_{p1}, K_{t1} are constants associated to the promoter and repressor respectively, and n is a Hill-like coefficient. The second reaction corresponds to the degradation of the protein $T2$ (with first order mass action kinetics).

The presence of an external inducer binding repressor $T1$ will add the following reactions (with mass action kinetics):

$$I + T1 \underset{k_{ui1}}{\overset{k_{bi1}}{\rightleftharpoons}} IT1 \xrightarrow{k_{di}} \emptyset$$

where k_{bi}, k_{ui} and k_{di} are the constants of binding, unbinding and degradation of the inducer complex, respectively.

References

[1] Dasika MS, Maranas CD (2008) Optcircuit: An optimization based method for computational design of genetic circuits. BMC Syst Biol 2, 24.

[2] Egea JA, Rodriguez-Fernandez M, Banga JR and Marti R (2007) Scatter search for chemical and bioprocess optimization. *J. Global Optim.* 37(3):481–503.

[3] Exler O, Schittkowski K (2007) A trust region sqp algorithm for mixed-integer nonlinear programming. Optim Lett 1(3), 269–280.

[4] Exler O, Antelo LT, Egea JA, Alonso AA, Banga JR (2008) A tabu search-based algorithm for mixed-integer nonlinear problems and its application to integrated process and control system design. Comput Chem Eng 32, 1877–1891.

[5] Mladenovic N, Hansen P (2010) Variable neighbourhood search: methods and applications. Annals of Operations Research, 175(1):367–407.

[6] Otero-Muras I, Banga JR (2014) Multicriteria global optimization for biocircuit design BMC Syst. Biol., 8(113), DOI: 10.1186/s12918-014-0113-3.

[7] Otero-Muras I, Banga JR (2014) Multiobjective design of synthetic oscillators from standard biological parts, Lecture Notes in Computer Science, 8859: 225–238.

[8] Pedersen M, Phillips A (2009) Towards Programming Languages for Genetic Engineering of Living Cells, J. R. Soc. Interface, 6. S437-S450.

[9] Schlueter M, Egea JA, Banga JR (2009) Extended ant colony optimization for non-convex mixed integer nonlinear programming. Comput Oper Res 36, 2217–2229.

[10] Serban R, Hindmarsh A C (2005) CVODES: the Sensitivity-Enabled ODE Solver in SUNDIALS, Proceedings of IDETC/CIE 2005, Sept. 2005, Long Beach, CA. Also available as LLNL technical report UCRL-JP-200039.