

# Bio-ModelChecker

## Bio-ModelChecker: A Biological Network Model Checker

Hooman Sedghamiz

November 21, 2018

# PREFACE AND LICENSE

Biological Model Checker (Bio-ModelChecker) is a bounded model checker for parameterization of generalized regulatory networks (Thomas Networks) created by Hooman Sedghamiz at [Center for Clinical Systems Biology \(CCSB\)](#) directed by Dr. Gordon Broderick at Rochester General Hospital.

Bio-ModelChecker treats the model parametrization problem as a multi-objective optimization inspired from *Control Theory* and *biology*. It employs the power of state of the art Constraint Programming in order to efficiently parameterize a regulatory network. Bio-ModelChecker is copyrighted 2018 by Center for Clinical Systems Biology at Rochester General Hospital, Rochester, USA. All rights reserved.

Bio-ModelChecker is available for research and evaluation purposes only. It can not be used in a commercial environment, particularly as part of a commercial product, without written permission. Bio-ModelChecker is provided as is, without any warranty. This software relies on third party packages for constraint solving such as [Google OR-tools Perron \[2011\]](#), [Minizinc Nethercote u. a. \[2007\]](#), [Chuffed Chu u.a. \[2014\]](#) and [OptimathSat Sebastiani und Trentin \[2015\]](#). For visualization Bio-MC relies on Java-Script, Gramm [Morel \[2018\]](#) and [D3 Project](#). For solution management, Bio-MC uses [SQLite3](#). Please visit their corresponding webpage for further information on licensing.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Please write to [hooman.sedghamiz@rochesterregional.org](mailto:hooman.sedghamiz@rochesterregional.org) or [gordon.broderick@rochesterregional.org](mailto:gordon.broderick@rochesterregional.org) for additional questions regarding licensing Bio-ModelChecker or obtaining more up-to-date versions.

Copyright, Feb, 2018

Center for Clinical Systems Biology (CCSB)  
Hooman Sedghamiz

# CONTENTS

<b>PREFACE AND LICENSE</b>	<b>2</b>
<b>GETTING STARTED</b>	<b>4</b>
Basics . . . . .	4
Compatibility and Dependencies . . . . .	4
Installation . . . . .	5
Quick Start . . . . .	5
Bio-ModelChecker Overview . . . . .	6
Main Panel . . . . .	6
Solution Statistics . . . . .	7
Importing a model . . . . .	8
JSON Format . . . . .	8
JSON Spec . . . . .	9
Preparing your model as JSON in your favorite Language . . . . .	10
Benchmarks . . . . .	10
Parameterizing a model . . . . .	11
Variable Ordering . . . . .	11
Edge Constraint & PostComboConst . . . . .	11
Interpreting Results . . . . .	12
Drug Target Identification . . . . .	15
Panel A . . . . .	15
Panel B . . . . .	15
Panel C . . . . .	16
Exporting MIS . . . . .	16

# GETTING STARTED

## BASICS

### COMPATIBILITY AND DEPENDENCIES

Bio-ModelChecker is currently distributed as a single installation file only for Windows 64bit operating systems. It does not have any dependencies and is shipped with [Google OR-tools Perron \[2011\]](#), [Chuffed Chu u.a. \[2014\]](#), [OptimathSat Sebastiani und Trentin \[2015\]](#) and corresponding [FlatZinc Nethercote u.a. \[2007\]](#) interpreter.

In fact, Bio-ModelChecker is independent of the solver. Any solver capable of reading **FlatZinc** language is compatible. In the future release of Bio-ModelChecker, we plan to add a solver importer panel in order to make it easier to use other solvers not currently offered in Bio-ModelChecker.

### BIO-MODELCHECKER

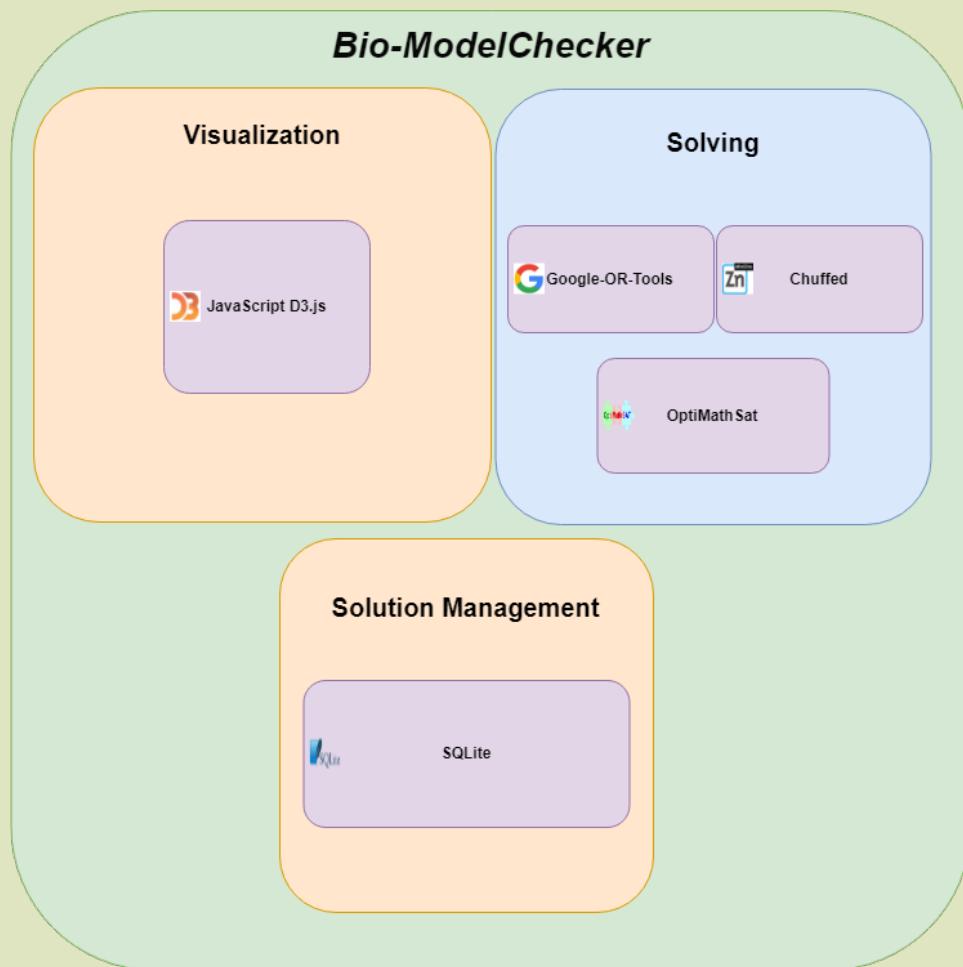


Figure 1: Building Blocks of Bio-MC.

## ADMINISTRATIVE ACCESS!

Bio-ModelChecker needs to be run under administrative access.

## INSTALLATION

In order to install Bio-ModelChecker, simply run **BMC-Installer.exe**. This file installs all the dependencies for Bio-ModelChecker and prompts you with the next steps of the installation. If successful, the installer creates an executable file called **Bio-ModelChecker.exe** and a few more directories holding the dependencies of the software.

### NOTE!

After installation you need to perform the two following actions:

1. Locate the directory where you installed **Bio-ModelChecker.exe** (e.g. under application directory). Then, you need to copy **classpath.txt** to the directory where you installed *MATLAB Run Time* under the following address : Matlab Run Time\ v91\ toolbox\ local\ (replace the old classpath.txt).
2. Again from where you installed **Bio-ModelChecker.exe** and move the complete **jxbrowser** directory to the following address : Matlab Run Time\ v91\ java\ jarext\

Please write to [hooman.sedghamiz@rochesterregional.org](mailto:hooman.sedghamiz@rochesterregional.org) if you might encounter any error during installation or might need additional information regarding the updated versions of Bio-ModelChecker.

## QUICK START

Double click **Bio-ModelChecker.exe** and click on *import* to import a model from **Benchmark** directory. Select a time-update from the upper left corner of Bio-ModelChecker (see Figure 2) and select a solver (e.g. **Chuffed**). Hit **Run**. If successful a cmd window will be opened and a solving instance of the model would run on it. Note that if you close the cmd window the solving will be killed. The parametrization results are saved in the **results** folder with the same name as the model.

# BIO-MODELCHECKER OVERVIEW

## MAIN PANEL

Bio-ModelChecker model checker module panel is illustrated in Figure 2. Right panel provides an excel type view of the adjacency network under analysis. Left panel provides settings for tuning the time update, model checking bound, type of the solver and maximum solving time out in seconds. In order to get started, one first needs to import a model and get familiar with the input format of Bio-ModelChecker. See the next section for more details.

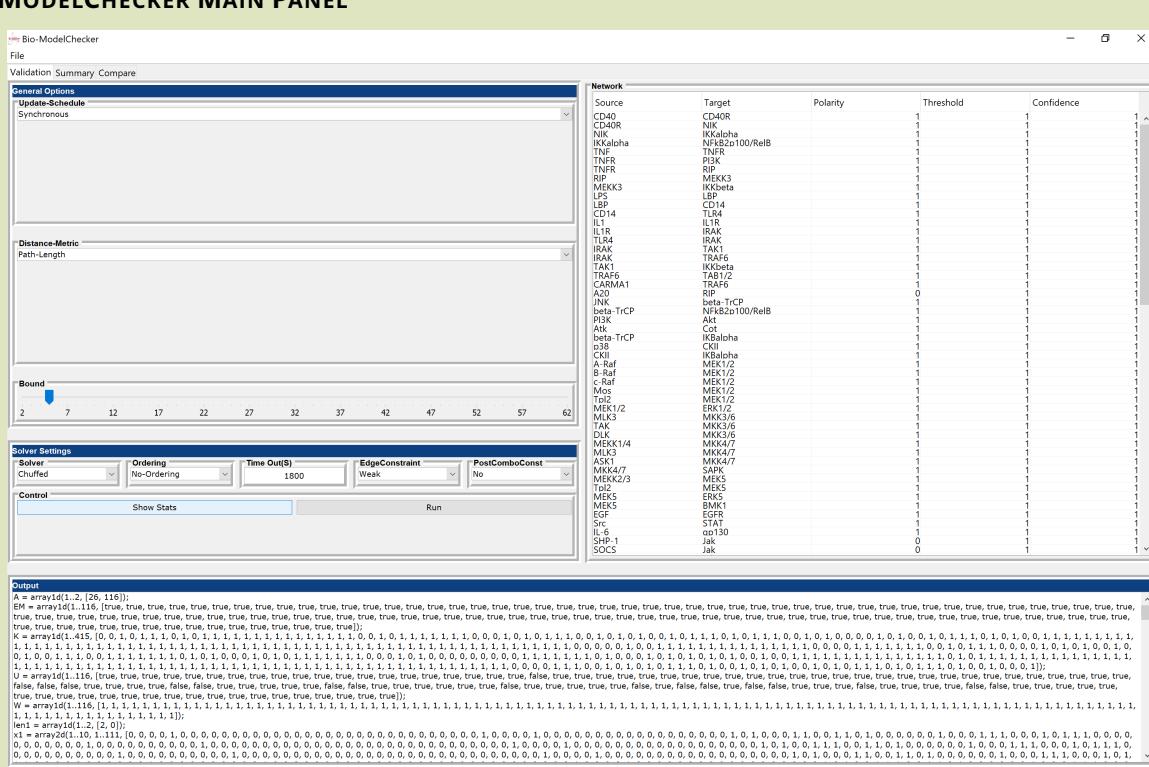


Figure 2: Right panel provides an *excel* type view of the adjacency network under analysis. Left panel provides settings for tuning the time update, model checking bound, type of the solver and maximum solving time out in seconds.

## SOLUTION STATISTICS

Clicking on the *Show Stats* button would display the stats of solutions and their corresponding normalized objective values.

### OVERALL STATS OF SOLUTIONS

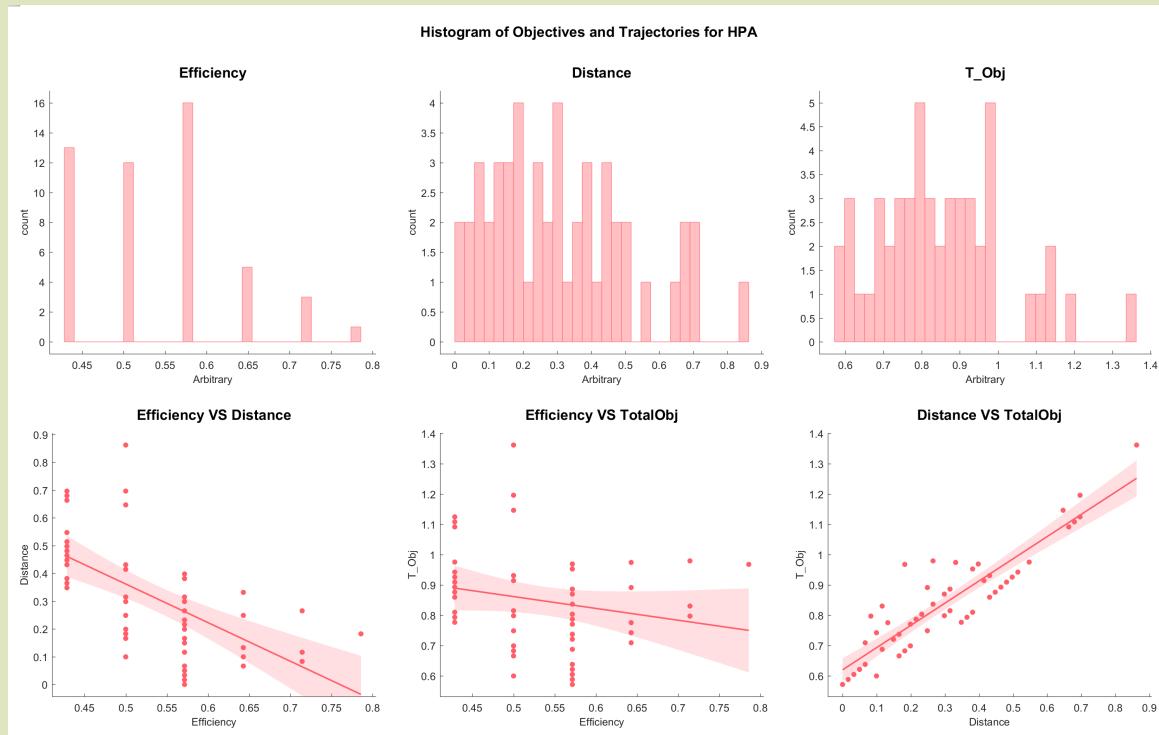


Figure 3: Stats of HPA axis parameterization. Histograms indicate the number of solutions and their corresponding objectives.

# IMPORTING A MODEL

## JSON FORMAT

Bio-ModelChecker employs *Java Script Object Notation (JSON)* as a standard input format. This is due to the fact that JSON follows a standard nesting of objects and there already exists various tools in different languages in order to conveniently convert a matrix or complex structure to a JSON file. Figure 4 is an example of Hypothalamic Pituitary Axis (HPA) model in JSON format. The main object is the model name (e.g. HPA) and the rest of its attributes are nested under it with [ ]. Note the naming of each attribute. Bio-ModelChecker uses the name of attributes as keys in order to distinguish the name of entities (e.g. titles) or the adjacency list of the network (e.g. interaction). For a detailed list of attributes and their naming convention see Table 1.

### REMARKS!

- A model always is represented as an object and has a name (e.g. HPA)
- Interaction has 5 rows, all the unknowns are denoted by -1 (See next page for spec).
- Tr holds the measurements. In this case there are two measurement instances, the first has 2 time samples and the second has 3 samples.
- L stands for maximum levels of each entity. In this case the first two entities assume a binary value while the last two are ternary.

### HPA.JSON

```
{  
  "HPA": {  
    "interaction": [  
      [1,2,2,3,4,4,1,3],  
      [3,1,4,2,3,4,4,1],  
      [-1,-1,-1,-1,-1,-1,-1,-1],  
      [0,1,0,1,1,1,-1,-1],  
      [-1,-1,-1,-1,-1,-1,-1,-1]  
    ],  
    "L": [1,1,2,2],  
    "Tr": [  
      [  
        [0,0,0,0],  
        [1,1,0,0]  
      ],  
      [  
        [0,0,2,2],  
        [0,0,1,2],  
        [0,0,0,2]  
      ]  
    ],  
    "titles": ["CRH", "ACTH", "Cort", "R"]  
  }  
}
```

Figure 4: JSON example.

## JSON SPEC

A detailed description of attributes can be found in Table 1.

### BIO-MODELCHECKER JSON ATTRIBUTE SPEC

*Attributes and their interpretation by Bio-ModelChecker*

#### **Name of entities in the model.**

Attribute key in JSON:

'title','names','entities','TITLES'.

#### **Adjacency list of the model.**

Attribute key in JSON:

'connection','interaction','edgelist','Edge','Adjacency','ADJ'.

Attribute in JSON:

- 1st Row : Target node
- 2nd Row : Source node
- 3rd Row : Threshold of action (min:1,max:inf,unknown:-1)
- 4th Row : Polarity (negative:0,positive:1,unknown:-1)
- 5th Row : Confidence (Certain:1,unknown:-1)

#### **Max Expression Level.**

Attribute key in JSON:

'Levels','L','levels'.

Note: It should be a vector the same length as the number of entities in the model.

#### **Measurement Trajectory Matrices.**

Attribute key in JSON:

'Tr','Trajectory','measurement'.

Note: It should be matrix with minimum 2 rows and columns the same size as the number of entities in the model.

Unknown measurements are denoted by -1.

#### **Node Steady States.**

Attribute key in JSON:

'attractor','atr','ATR','SS','SteadyState'.

Note: Each attractor is a vector with the same length as the number of entities. For the unknown entities one can pass in -1.

Table 1: Bio-ModelChecker Spec.

## PREPARING YOUR MODEL AS JSON IN YOUR FAVORITE LANGUAGE

---

Table 2 details a list of import&export packages available under different programming languages in order to convert a data-type into JSON format.

Language	Toolbox&Link
R	jsonlite
MatLab	JSONlab
Python	JSON
C++	JSON++
Java	JSON Oracle
Scala	CIRCE

Table 2: Packages for working with JSON format.

## BENCHMARKS

---

Bio-ModelChecker currently is shipped with 5 benchmarks that are detailed in Table 3. Each example provides a different type of model in the sense that some have missing interactions or require to reproduce a certain node steady state or measurement. The models have been chosen specifically to show different functionalities of Bio-ModelChecker. Number of measurements here stands for a measurement under a certain condition not number of samples in each measurement. For instance, a measurement might have been done under knockout of a certain input and measured for 10 time samples. The benchmarks can be found in the **Benchmark** directory.

Model	Author	Nr. Entities	Nr. Measurements	Nr. Attractors
HPA Axis	Sedghamiz u.a. [2018]	4	2	0
HPG Axis	Sedghamiz u.a. [2017]	5	1	0
Dendritic Cell	Garg u.a. [2008]	111	1	0
Irma	Cantone u.a. [2009]	6	1	0
Th-helper	Garg u.a. [2008]	23	0	3

Table 3: Benchmarks provided with Bio-ModelChecker.

## PARAMETERIZING A MODEL

In order to parameterize a model, first *import* the model (e.g. from **Benchmark** directory). Then, choose a bound for model checking. Select an updating scheme (Synchronous or Asynchronous [Chaouiya u. a. \[2003\]](#)) and a particular solver. After hitting the *Run*, Bio-ModelChecker translates all the constraints along with measurement trajectories into a **FlatZinc** that is solved with a SAT, CP, SMT or LCG solver.

- Note that Bio-MC offers two different measures for parameterization;
- 1. **Path-Length:** This measure checks the State Transition Graph (STG) associated with a model for reachability within the *bound*. Note that increasing the bound dramatically increases the solving time. Specially for larger models, one might start with a lower bound, since Bio-ModelChecker relies on CP technique which is usually really fast to prove *Unsatisfiability*.
- 2. **Taxicab:** This measure minimizes the Manhattan distance between the output of the model and the time-measurement samples. *Bound* has no use in this approach. It is advised to use this measure with the larger models since reachability analysis might be impractical.

## VARIABLE ORDERING

The *ordering* option allows the user to select the importance of objective functions. For instance, the lexicographical would give a much higher weight to the first objective which is either the *cost length* (in Path-Length) or *Manhattan error* (in Taxicab).

## EDGE CONSTRAINT & POSTCOMBOCONST

These settings let the user to sharpen the edge constrains (see [Sedghamiz u. a. \[2017\]](#)). For instance, selecting the strict option would force the negative interactions to be strictly inhibiting and dominant even when other cofactors exist. The *PostComboConst* ensures that the effect of a single factor is always equal or less than its effect when it is combined by other factors. This option is not recommended for larger models due to its complex combinatorial nature.

# INTERPRETING RESULTS

## SUMMARY PANEL

The summary panel currently provides a moving average of all feasible solution. Hovering over the matrix and each cell would instantiate a tool-tip that summarizes some statistics about the interaction over all feasible models so-far (Pressing on Apply button updates the matrix).

### SUMMARY PANEL

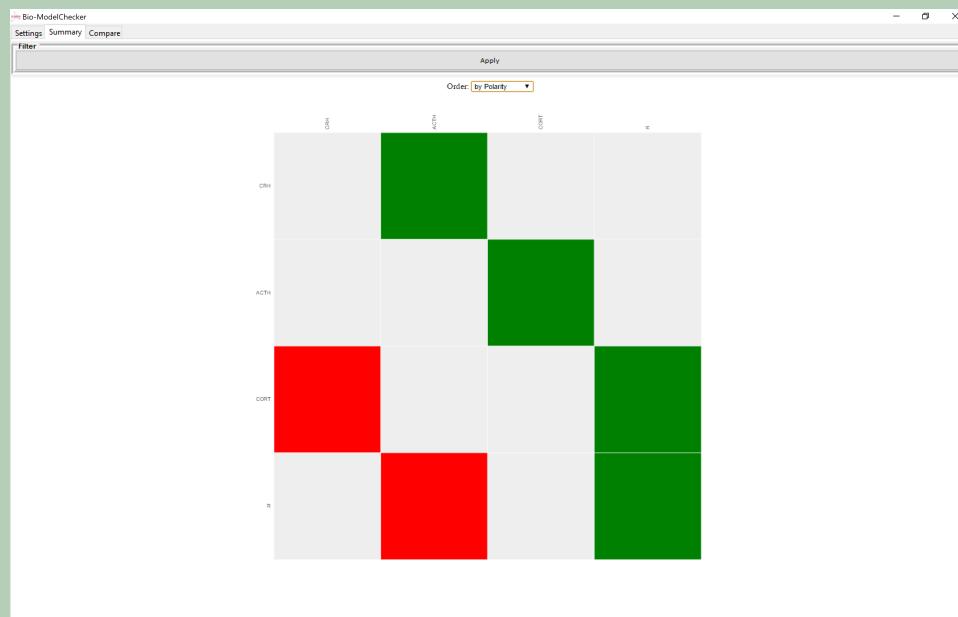
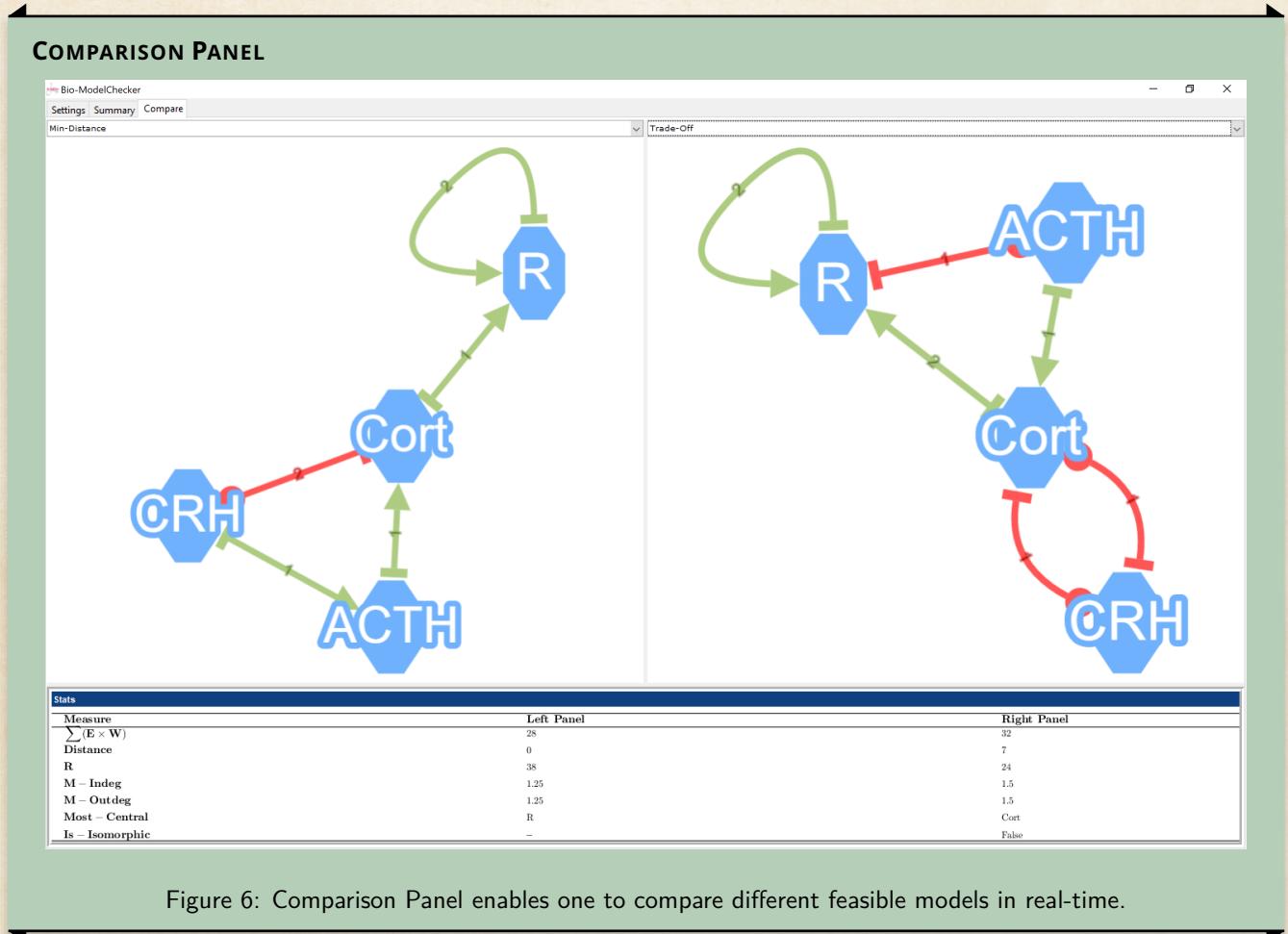


Figure 5: Summary Panel provides an interactive way of looking at all solutions in a statistical way. The entities might be reorganized based on their out-degree and other measures.

## COMPARISON PANEL

The comparison panel allows one to select a model based on an objective and compare with another visually and statistically. Since Bio-MC employs several objectives, it is of interest to find the solutions that are optimal in the dimension of only one objective. This panel is also interactive and zooming and panning is enabled. The SQLite queries are employed behind the scene in order to make the search for a solution more efficient.



## OUTPUTS

The output of the parameterization is saved in **results** folder as a **.txt** and **.db** file. The **.db** file might be read by any SQLite visualizer externally. The output if the parametrization was satisfiable looks like Figure 7;

```
HPA.TXT

A = array1d(1..3, [8, 0, 0]);
EM = array1d(1..8, [true, true, true, true, true, false, false]);
K = array2d(1..4, 1..4, [0, 1, 0, 0, 0, 1, 1, 1, 0, 2, 0, 0, 0, 1, 2, 2]);
U = array1d(1..8, [false, true, false, true, true, true, false, false]);
W = array1d(1..8, [1, 1, 1, 1, 2, 2, 1, 1]);
```

Figure 7: Sample of parametrization results.

Table 4 details each vector in the results.

Name	Interpretation
A	Objective vector: $\sum$ (Number Edges $\times$ Threshold) of action , Min path length, Max Robustness respectively. Note that if there are more than one measurement matrix, the length of this vector is longer but the order the same (first two pair belong to measurement 1 and the second pair to measurement 2 and so on.)
EM	Confidence of interactions, 1 implies that the edge is necessary, 0 edge is redundant
K	Logical parameters
U	Polarity of each edge
W	Threshold of action for each interaction.

Table 4: Result output Spec.

# DRUG TARGET IDENTIFICATION

The drug target identification is used to compute Minimal Intervention Sets (MIS) that move the system from a single or set of initial states to a *desired* steady state. This functionality can be accessed from the **DrugTargetIdentification** panel. In order to use this functionality, first import a json input model. After importing a model, import the family of the solutions associated with it (e.g. from the result directory). Figure 8 illustrates different blocks of the tool.

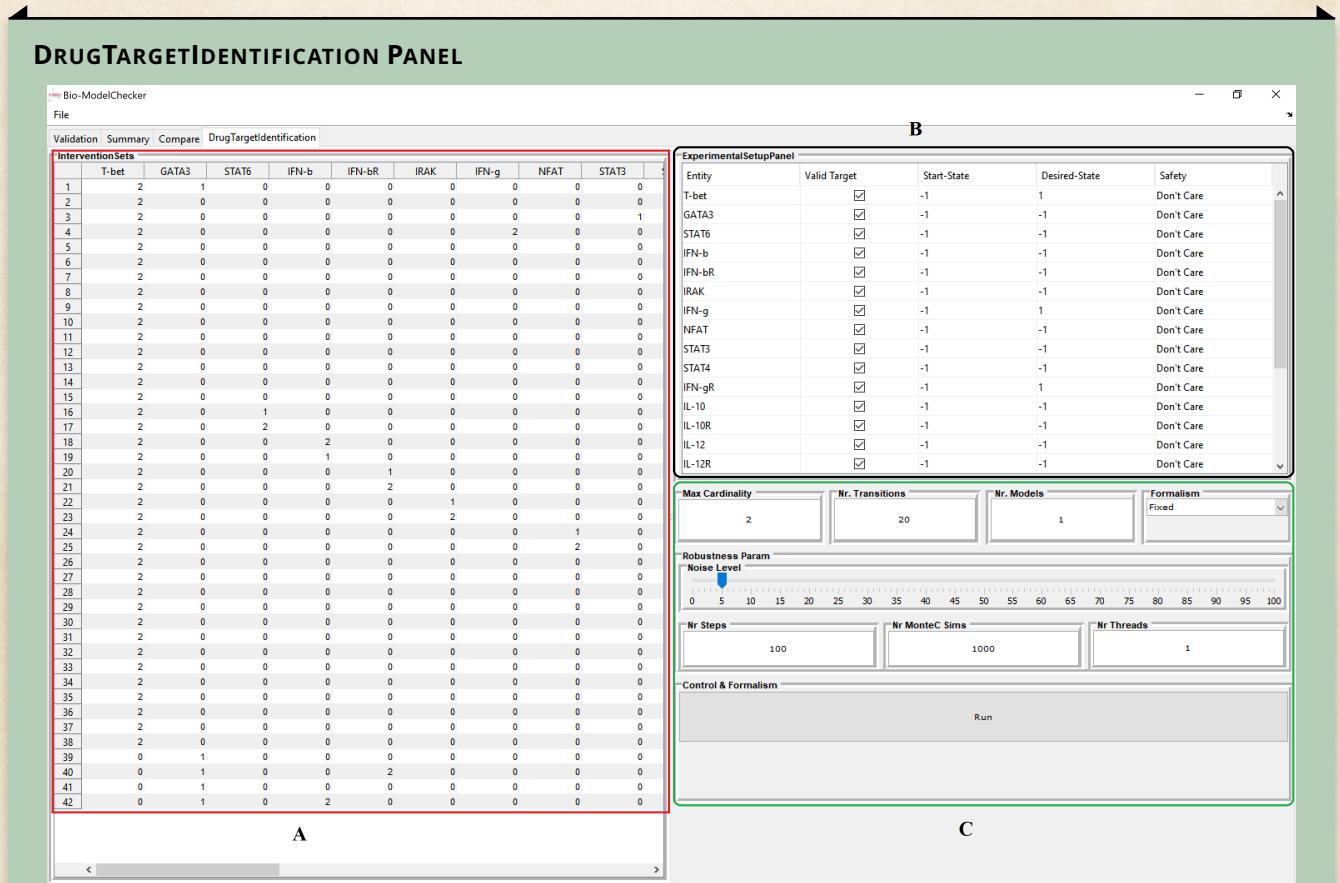


Figure 8: Drug Target Identification panel. It is possible to start from a fixed initial state or complete unknown state (e.g. all values -1).

## PANEL A

This panel details each MIS candidate and computes their associated cardinality, Robustness and efficiency based on Monte-Carlo simulations. Each entity of the network is assigned a 0 that means no perturbation, 1 that indicates an inhibition or 2 that implies a knock-in.

## PANEL B

In this panel, one is able to setup the MIS computation. The second column lets the user select the entities that can be targeted. The third indicates an initial state, -1 corresponds to an unknown initial state. The third column specifies the desired target steady state and the last column hard constraints about safety.

## PANEL C

---

In this panel, one is able to constrain the number of intervened entities and also indicate the number of transitions that the model checker would use in order to compute the MIS. The rest of the parameters are related to the Monte-Carlo simulations.

## EXPORTING MIS

---

It is possible to export the results as a csv file from the `File` tab.

# BIBLIOGRAPHY

- [Cantone u. a. 2009] CANTONE, Irene ; MARUCCI, Lucia ; IORIO, Francesco ; RICCI, Maria A. ; BELCASTRO, Vincenzo ; BANSAL, Mukesh ; SANTINI, Stefania ; BERNARDO, Mario di ; BERNARDO, Diego di ; COSMA, Maria P.: A Yeast Synthetic Network for In Vivo Assessment of Reverse-Engineering and Modeling Approaches. In: *Cell* 137 (2009), Nr. 1, S. 172–181. – URL <http://dx.doi.org/10.1016/j.cell.2009.01.055>. – ISBN 0092-8674
- [Chouiya u. a. 2003] CHAOUIYA, Claudine ; REMY, Elisabeth ; MOSS, Brigitte ; THIEFFRY, Denis: Qualitative Analysis of Regulatory Graphs : A Computational Tool Based on a Discrete Formal Framework. In: *Positive Systems* (2003), S. 119–126. ISBN 978-3-540-40342-5
- [Chu u. a. 2014] CHU, Geoffrey ; GARCIA DE LA BANDA, Maria ; MEARS, Christopher ; STUCKEY, Peter J.: Symmetries, almost symmetries, and lazy clause generation. In: *Constraints* 19 (2014), Nr. 4, S. 434–462. – ISBN 9781577355120
- [Garg u. a. 2008] GARG, Abhishek ; DI CARA, Alessandro ; XENARIOS, Ioannis ; MENDOZA, Luis ; DE MICHELI, Giovanni: Synchronous versus asynchronous modeling of gene regulatory networks. In: *Bioinformatics* 24 (2008), Nr. 17, S. 1917–1925. – ISBN 1367-4811 (Linking)
- [Morel 2018] MOREL, Pierre: Gramm: grammar of graphics plotting in Matlab. In: *Journal of Open Source Software* 3 (2018), S. 568. – URL <https://github.com/piermorel/gramm>
- [Nethercote u. a. 2007] NETHERCOTE, Nicholas ; STUCKEY, Peter J. ; BECKET, Ralph ; BRAND, Sebastian ; DUCK, Gregory J. ; TACK, Guido: MiniZinc: Towards a Standard CP Modelling Language. In: BESSIÈRE, Christian (Hrsg.): *Thirteenth International Conference on Principles and Practice of Constraint Programming* Bd. 4741. Providence, RI, USA : Springer-Verlag, sep 2007, S. 529–543. – URL <http://www.gecode.org/paper.html?id=NethercoteStuckeyEa:CP:2007>
- [Perron 2011] PERRON, Laurent: Operations Research and Constraint Programming at Google. In: LEE, Jimmy (Hrsg.): *Principles and Practice of Constraint Programming – CP 2011*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 2. – ISBN 978-3-642-23786-7
- [Sebastiani und Trentin 2015] SEBASTIANI, Roberto ; TRENTIN, Patrick: OptiMathSAT: A Tool for Optimization Modulo Theories. In: *Proc. International Conference on Computer-Aided Verification, CAV 2015* Bd. 9206, Springer, 2015
- [Sedghamiz u. a. 2017] SEDGHAMIZ, H ; CHEN, W ; RICE, M ; WHITLEY, D ; BRODERICK, G: Selecting Optimal Models Based on Efficiency and Robustness in Multi-valued Biological Networks. In: *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, oct 2017, S. 200–205
- [Sedghamiz u. a. 2018] SEDGHAMIZ, Hooman ; CHEN, Wenxiang ; RICE, Mark ; WHITLEY, Darrell ; BRODERICK, Gordon: Selecting optimal models based on efficiency and robustness in multi-valued biological networks. In: *Proceedings - 2017 IEEE 17th International Conference on Bioinformatics and Bioengineering, BIBE 2017* 2018-Janua (2018), S. 200–205. ISBN 9781538613245