User Guide of ASSA-PBN Version 3.0

Qixia Yuan

May 4, 2018

Contents

1 Introduction

ASSA-PBN is a software tool for modelling, simulation and analysis of probabilistic Boolean networks (PBNs), especially for large PBNs, which naturally arise in the domain of Systems Biology. ASSA-PBN provides several methods for analysing a PBN. This includes the basic steady-state computation, and the in-depth long-term influence and sensitivity computation, attractor detection, and parameter estimation of a PBN. We distinguish two types of PBNs: instantaneously random PBNs (IPBNs) and context-sensitive PBNs (CPBNs) [9].

2 Modules

ASSA-PBN contains three major parts (see Figure 1): modeller, simulator, and analyser. They interact with each other to provide a full support for PBN analysis. The modeller provides a simple way to construct a real-life biological system, e.g., gene regulatory network (GRN), into a PBN and supports visualisation and dynamically changing of a PBN. The simulator takes the PBN constructed in the modeller and performs simulation to produce samples. Simulation does not suffer from the state-space explosion problem even in terms of large PBNs since it is not based on the transition matrix. Based on the constructed model and generated samples, the analyser performs basic and in-depth analysis of the PBN. The analysis results can be used either to interpret the original system or to optimise the modelling of the system. Figure 1 shows the structure of ASSA-PBN. The analyser requires different input files depending on the analysis task. While simulator and analyser rely on modeller as input, the simulation and analysing results can be used to optimise modelling construction.

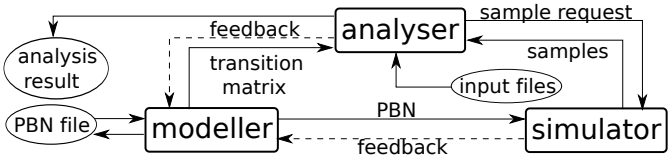


Figure 1: Structure of ASSA-PBN.

The ASSA-PBN program package can be downloaded at `http://satoss.uni.lu/software/ASSA-PBN/`. Currently, ASSA-PBN provides the following functions (those hightlighted in bold are newly added functions in version 3.0.*):

- modelling of PBNs in high-level ASSA-PBN format for **CPBN** and IPBN;
- various asynchronous updating schemes;
- random generation of PBNs;
- efficient simulation of a PBN;
- computation of steady-state probabilities of a **CPBN** and IPBN;

- parameter estimation of a **CPBN** and IPBN;
- long-run influence and sensitivity analysis of an IPBN;
- fit-score sensitivity analysis of an IPBN;
- heat map plot of the parameter estimation results;
- **attractor detection**;
- a command-line tool and a GUI.

3 Installation guide

3.1 System Requirement

1. Platform: x86 compatible 32 bit or 64 bit processor;
2. Operating system: Windows, Linux and Mac OSX (10.4 and above);
3. Required environment: Java SE Development Kit 8 or higher;
4. Required compilers (for attractor detection): flex 2.5.4 or higher, GNU bison 2.3 or higher, GNU g++ 4.0.1 or higher; Cygwin 1.5.25-14 or higher (Windows platform only); NVIDIA's CUDA Compiler (for GPU computation only).

3.2 Installation

For basic usage, ASSA-PBN does not require installation operation. For usage in Windows, the software can be run by double clip on the assa.jar file after extracting the downloaded file. For usage in Mac OS of Linux, the software can be run with the command "./xassa" in the "bin" folder. Alternatively, the command line version can be run with the command "./assa <parameters>" in the "bin" folder. More examples on running ASSA-PBN can be found in Sections 5 and 6.

Installation for attractor detection function. If the user wants to use the attractor detection function, an additional installation is required. It can be down by the following two steps.

1. (Windows platform only) Install cygwin and the packages g++, flex and bison on Windows. Detailed instructions can be found from http://www.cygwin.com/.

2. In a terminal (please use the cygwin terminal for Windows), navigate to the mcmas folder. Simply use the command make to the compile MCMAS. We now give an example in Windows. Assume the downloaded ASSA-PBN is extracted in the folder "C:\assa-pbn-3.0.1". Open the cygwin terminal, run "cd C:\assa-pbn-3.0.1" and then run "make". On Windows, the directory "bin" in the cygwin installation must be added to the system path. For example, append c:\cygwin64\bin to the system path if cygwin is installed in c:\cygwin64.

Installation for computation with GPUs. If the user wants to benefit from the parallel computation power of GPU codes, the NVIDIA's CUDA Compiler (NVCC) (https://developer.nvidia.com/cuda-downloads) are required. The parallel computation of GPU is currently only supported for Mac OS and Linux. In addition, an additional installation step (see below) is required in order to enable the GPU-based parallel computation.

- In a terminal, navigate to the ASSA-PBN folder and run `make`.

ASSA-PBN can be run either in command line or in a GUI.

4 Input Files

The newest ASSA-PBN supports definition of a PBN in a high level format, which is easier for the user to define a PBN. See section 4.1 for more details.

ASSA-PBN version 1.0 accepts five types of input files, namely the PBN definition file, the parameter definition file, and the property specification file. The PBN definition file is used to store the PBN in ASSA-PBN format; the generation definition file is used to specify the parameters for randomly generating a PBN; and the property specification file is used to specify an interested set of states. In version 2.0, ASSA-PBN accepts another two input files to support parameter estimation: the parameter specification file and the experiment data file. The parameter specification file is used to specify the nodes indices whose function selection probabilities are to be fitted and the experiment data file is used to specify the lab experiment data.

4.1 PBN definition files

In version 1.0, ASSA-PBN uses a low level PBN definition file, in which PBNs are specified with a list of numbers. In version 2.0, ASSA-PBN supports a high level PBN definition format, which is much easier for users to define a PBN. We will use an example to show the two different formats of the PBN definition file. This example PBN contains 3 nodes. Node 1 has one Boolean function and each of node 2 and 3 has two Boolean functions. The Boolean functions and their selection probabilities are shown in Table 1. The corresponding truth table is shown in Table 2. In addition, this PBN has a perturbation rate 0.001.

High level definition format. We describe this model using high level PBN definition format as in Figure 2. In this file, a line starting with "//" indicates a comment line. The high level definition format contains four parts. The first part is the header, specifying the model type, the number of nodes and the perturbation rate. ASSA-PBN supports seven kinds of update modes, including synchronous, asynchronous ROG, asynchronous RMG, asynchronous RMG-RM, asynchronous RMG-RO, asynchronous RMG-RO-RM, asynchronous AG-RO. The details of the six asynchronous modes are explained as follows.

node name	function	probability
x_0	$f_1^{(0)} = x_0 \vee x_1$	1
x_1	$f_1^{(1)} = \neg x_2 \wedge (x_1 \vee x_0)$	0.3
x_1	$f_2^{(1)} = \neg x_0 \wedge x_1 \wedge x_2$	0.7
x_2	$f_1^{(2)} = \neg x_0 \wedge x_1$	0.4
x_2	$f_2^{(2)} = x_0 \wedge x_1 \wedge \neg x_2$	0.6

Table 1: Boolean functions and their selection probabilities of a 3 nodes PBN

$x_2x_1x_0$	$f_1^{(0)}$	$f_1^{(1)}$	$f_2^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$
000	0	0	0	0	0
001	1	1	0	0	0
010	1	1	0	1	0
011	1	1	0	0	1
100	0	0	0	0	0
101	1	0	0	0	0
110	1	0	1	1	0
111	1	0	0	0	0
$c_j^{(i)}$	1	0.3	0.7	0.4	0.6

Table 2: Truth table corresponding to the predictor functions in Table 1.

- For the ROG (a random selection of one gene) mode, at each time point t ($t \in \mathbb{N}$), one gene is randomly selected for state updating and the other genes stay unchanged.
- For the RMG (a random selection of m genes with a fixed m) mode, at each time point t ($t \in \mathbb{N}$), m genes are randomly selected for state updating and the states of other genes remain the same as the current states. In this mode, m is a fixed value ranging from 1 to n.
- For the RMG-RM (a random selection of m genes with a random m) mode, at each time point t ($t \in \mathbb{N}$), m genes are randomly selected to be updated and the states of the other genes keep their values. For this mode, the only difference from RMG is that m is a random number ranging from 1 to n.
- For the RMG-RO (a random selection of m genes in a random order with a fixed m) mode, at each time point t ($t \in \mathbb{N}$), m genes are randomly chosen for state updating and the state of each of these m genes will be updated in a random order. The update of the next gene can be influenced by the states of previously updated genes. And m is a fixed number ranging from 1 to n. For the other genes, their states remain the same as their current values.

```
//The definition file always starts with the update modes:
//synchronous, rog, rmg, rmgrm, rmgro, rmgrorm, aro.
//For rmg and rmgro, m should be given as the following example:
//type=rmg m=2.
type=synchronous
//specify number of nodes
n=3
//specify perturbation rate
perturbation=0.001
//We list the names of all nodes one by one below.
//Each line specifies one node name.
//The order of the names determines their orders in ASSA-PBN.
//A node name can not start with a number.
//The following characters are not allowed for a node name:
//space, tab, *&!| ∧ / : ().
nodeNames
x0
x1
x2
endNodeNames
//If the PBN is a context−sensitive PBN, the contextDistribution
//section should be defined.  See the example in Figure 3.
//====================================
//We list the functions for each node one by one.
//Each line contains 1 Boolean function.
//It starts with a positive number denoting the selection
//probability.  The double number is separated with the
//remaining part using ":".  "0" is not allowed for probability.
//If the probabilities for a node do not sum up to 1, they will
//be normalised.
node x0
1.0 :  x0|x1
endNode
node x1
0.3 :  (!x2)&(x1|x0)
0.7 :  (!x0)&x1&x2
endNode
node x2
0.4 :  (!x0)&x1
0.6 :  x0&x1&(!x2)
endNode
npNode
x0
endNpNode
```

Figure 2: The corresponding PBN high level definition file of the PBN shown in Table 1.

- For the RMG-RO-RM (a random selection of m genes in a random order

```
//The definition file always starts with the update modes:
//synchronous, rog, rmg, rmgrm, rmgro, rmgrorm, aro.
//For rmg and rmgro, m should be given as the following example:
type=synchronous
//specify number of nodes
n=3
//specify perturbation rate
perturbation=0.001
//We list the names of all nodes one by one below.
//Each line specifies one node name.
nodeNames
x0
x1
x2
endNodeNames
//contextDistribution is used to define the selection probability for
//making the switch between different BNs
contextDistribution=0.1,0.2, 0.7
//We need switch to define the probability that a switch should be
//made at each simulation step
switch=0.001
//We list the functions for each node one by one.
//Each line contains 1 Boolean function.
//The function start with an integer denoting which BN this function
//belongs to.  If a function is shared by several BNs, all the BN
//indices should be listed with comma as separator.  The BN index
//starts with 0.  The integer numbers is separated with the remaining
//part using ":".
node x0
0,1,2 :  x0|x1
endNode
node x1
0,1 :  (!x2)&(x1|x0)
2 :  (!x0)&x1&x2
endNode
node x2
0,2 :  (!x0)&x1
1 :  x0&x1&(!x2)
endNode
npNode
x0
endNpNode
```

Figure 3: An example of context-sensitive PBN high level definition file (to be finished.

with a random m) mode, at each time point t ($t \in \mathbb{N}$), the state of randomly chosen m genes will be updated in a random order. The updating of the current gene has an impact on the update of the next gene. The

states of other genes remain unchanged. m is a random number ranging from 1 to n.

- For the AG-RO (all genes in a random order) mode, at each time point t ($t \in \mathbb{N}$), the states of all genes are updated in a random order. One gene will be randomly selected and updated until the updating runs through all genes. The updating of the current gene has an impact on the update of the next gene.

The format of the update modes are synchronous, rog, rmg, rmgrm, rmgro, rmgrorm, agro. The second part is between `nodeNames` and `endNodeNames`. It lists the names of all nodes one by one. The order of the names determines the order of the nodes in ASSA-PBN. In ASSA-PBN, each node will be assigned a unique non-negative integer as the ID of the node. The order of the nodes in fact determines the ID of the nodes in ASSA-PBN. A node name can not start with a number. The following characters are not allowed for a node name: space, tab, $*\&!|\wedge/:()$. The third part specifies the function and selection probability details for all the nodes. The functions and selection probabilities of each node are put between the keywords `node` and `endNode`. Each line is specified in the format of "probability : function". The probabilities of a node will be normalised if they do not sum up to 1. The functions can be defined using parenthesis and the four different logical operations, i.e., &, logical and; |, logical or; !, logical not, and $\wedge$, logical exclusive or. Note that the logical not operation is required to be surrounded by a pair of parenthesis, e.g., $(!x_2)$.

ASSA-PBN allows constant functions. To define a constant true function, one need to use the format "probability : TRUE" while to define a constant false function, one need to use the format "probability : FALSE". For example, if the Boolean function for node x_0 is changed to a constant function $f_1^{(0)} = true$, we would need to define node x_0 as follow:

node x_0
`1.0 :` $TRUE$
endNode

Please note that the following keywords (case-insensitive) are not allowed for a node name.

ASSA-PBN keywords: node, true, false, type, synchronous, asynchronous.

Low level definition format. Alternatively, ASSA-PBN accepts defining a PBN using low level format. In the low level format, node names are removed and only node IDs are left. Therefore, the node name x_0, x_1, x_2 will be named as 0, 1, 2 in the definition file. The Boolean functions are converted to a sequence of binary numbers, known as Boolean sequence, in the low level format. For example, the first boolean function $f_1^{(0)} = x_0 \wedge x_1$ is converted to 0 0 0 1. This function is affected by node x_0 and x_1; so its parent nodes indices are 0 and 1. The parent nodes indices will be given after the Boolean sequences in the definition file.

```
//The first line is the number of nodes in the PBN.
3
//The second line is optional.  When the file is exported from
//Matlab, this code should be added.
source=Matlab
//The third line indicates the number of Boolean functions for each
//node.
1 2 2
//The fourth line indicates the number of variables for each Boolean
//function.
2 3 3 2 3
//Boolean functions are listed below.  One function per line.
0 0 0 1
0 1 1 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0
0 0 0 1 0 0 0 0
//Then variable (parent nodes) indices of the Boolean functions are
//then listed.  One function per line.
0 1
0 1 2
0 1 2
0 1
0 1 2
//The selection probabilities for each Boolean function are listed.
//One node per line.
1
0.3 0.7
0.4 0.6
//perturbation rate.  Please put 0 if there is no perturbation.
0.001
//If you want to disable the perturbation of certain nodes, please
//provide the node indices below.
0
```

Figure 4: The corresponding PBN low level definition file of the PBN shown in Table 1.

The corresponding low level ASSA-PBN definition file of the above example is shown in Figure 4. In the definition file, comments are indicated with `//`. The comments can only be added at the beginning of a line. Note that the node index starts from 0.

In addition, ASSA-PBN also supports definition files exported from the Matlab-PBN-toolbox. In the downloaded package of this software, there is a Matlab file named `exportPBNtoASSA.m`, which can be run in Matlab to export a PBN model defined in the Matlab-PBN-toolbox to the ASSA-PBN format. For example, a PBN model can be exported to a file named `PBNfromMatlab.pbn` by running the following code in Matlab: `exportPBNtoASSA('PBNfromMatlab.pbn')`.

4.2 Parameter definition files

The parameter definition files are used to specify parameters for generating random PBNs. See Section 6.1 for a detailed explanation.

4.3 Property specification files

The property specification file defines the set(s) of states whose steady-state probability(ies) is(are) to be computed. A typical property specification file contains three lines. The first line specifies how the nodes are refereed in the file, i.e., `refer=name` means nodes are referred by their name, i.e., `refer=ID` means nodes are referred by their ID. By default, we use ID to refer the nodes. Therefore, `refer=ID` can be omitted. The second line specifies those nodes, whose values should be 1 (active); while the third line specifies those nodes, whose values should be 0 (inactive). The indices in the same line are separated with a space. Figure 5a shows an example of a property specification file for the PBN shown in Figure 2. It is specified using the node names. The last line is -1, indicating that those inactive nodes are not relevant in this property. This property file defines 4 states, i.e., 100, 101, 110, 111. Similar to the PBN definition file, property specification files recognise a line starting with "//" as comments. A same property can be specified using node IDs, as shown in Figure 5b. In most cases, ASSA-PBN only requires an input of one set of states, Starting from version 2.0.1, ASSA-PBN supports checking several properties at the same time using the same trajectory. In this case, the property specification file is required to define more than one set of states. See figure 5c for a property specification file specifying two sets of states.

Note that when the PBN is defined in a low level definition format, it is meaningless to define a property with the node names. Therefore, we use only the IDs to define a property file.

```
refer=name
x0
-1
```

(a) The value of nodes x_0 is 1 and those inactive nodes are not relevant.

```
refer=ID
0
-1
```

(b) The value of the 0th node is 1 and those inactive nodes are not relevant.

```
refer=ID
0 2
1
1
-1
```

(c) Two properties in one file.

Figure 5: Example of property specification files.

4.4 Parameter specification file

The parameter specification file defines the node indices to be estimated in the parameter estimation run. It contains a node index in each line. Figure 6 shows an example. It specifies two node indices, i.e., 80 and 91.

```
80
91
```

Figure 6: Example of the parameter specification file.

4.5 Experiment data file

The experiment data file specifies the lab experiment data which provide steady-state probabilities of certain nodes and are used to perform parameter estimation. In this file, comments are indicated by "//" at the beginning of a line. The file should start with "pbnIndex=0" to indicate that the following contents until the next appearance of "pbnIndex" are for the "0"th PBN. For each PBN, several steady-state probabilities might be provided. Each steady-state probability point is specified with four lines. The first two lines define the measured set of states in the same way as in property specification files. The third line defines the measured data for the specified set of states and the last line defines the weight of this data. It happened in the lab that different measured datas were obtained when the same experiments were performed for a second time. One can provide all the data information in the third line.w Figure 7 shows an example of the experiment data file format. Four data steady-state probabilities for 2 PBNs are specified in this file.

5 Running ASSA-PBN with GUI

ASSA-PBN provides both a graphical user interface (GUI) and a command line interface. Both use the same underlying simulator. The latter is useful for running large batches of jobs, leaving long-running steady-state analysis tasks in the background. We demonstrate how to run ASSA-PBN in a GUI in this section.

5.1 Starting the GUI of ASSA-PBN

Open a terminal and change the directory to the "bin" folder of ASSA-PBN, and run the command "./xassa". The main interface of ASSA-PBN will be launch, as shown in Figure 8. The interface is divided into three parts: the menu at the top, three panels in the middle and the status bar at the bottom. The panels are used to display PBN information and running information, which will be shown in the following examples.

5.2 Generating a random PBN

Typically, the first step to use ASSA-PBN is to load a PBN specified in ASSA-PBN. This can be down by the generating operation or by the loading operation. We first show here the steps to generate a random PBN. After launching the GUI, select menu option "Model | Generate". A window will pop up asking for

```
//define data point for the "0"th PBN
pbnindex=0
//states set 1
14
-1
//measurement for set
//multiple measurement values can be specified here
//seperated with blank
0.00 0.10 0.20
//weight for set 1
1
//states set 2
94
-1
//measurement for set 2
0.00 0.50 0.65
//weight for set 2
1
//define properties for the "1"th PBN
pbnIndex=1
//states set 1
14
-1
//measurement for set 1
0.63 0.61 0.62
//weight for set 1
1
//states set 2
95
-1
//measurement for set 2
1.00 0.95 0.90
//weight for set 2
1
```

Figure 7: Example of the parameter specification file.

details information of the model to be generated. Input the necessary parameter values, e.g., the number of nodes and perturbation rate, and click "Generate!", a PBN is then generated. Information about the generated PBN is shown in the top left panel. Except for defining parameters in the pop up window, the parameters can also be specified in a parameter definition file. Figure 29 shows an example of parameter definition file. To use the parameter definition file, click "Generate from file" and select the parameter definition file. A PBN is then

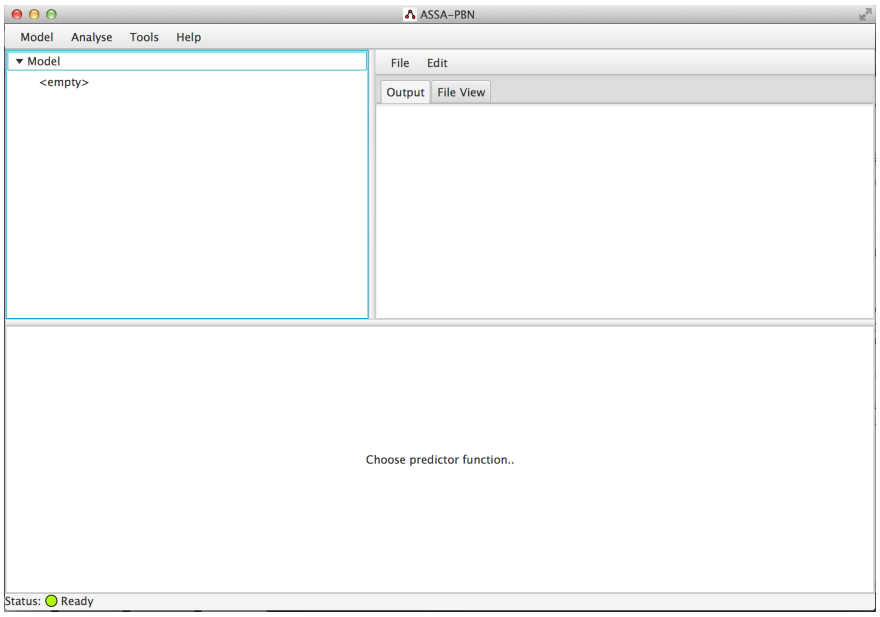


Figure 8: Screenshot of the ASSA-PBN interface after launch.

generated. To save the generated PBN, select menu option "Model | Export". The PBN can then be saved in ASSA-PBN format or in Matlab format.

5.3 Defining and Loading a PBN

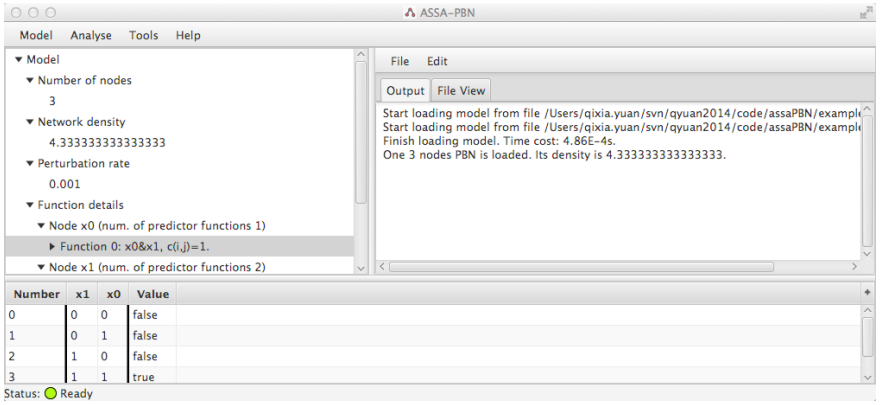


Figure 9: Screenshot of loading a PBN in ASSA-PBN.

As explained in section 4.1, ASSA-PBN provides two ways to define a PBN. We now show how to load the PBN in Figure 2. We save the PBN in a text file named dummy.pbn. Select the menu option "Model | Load", and then select the file "dummy.pbn" and click "Open". The PBN is then loaded into ASSA-PBN

as shown in Figure 9. The general information of the loaded PBN, e.g., the number of nodes, the network density, the perturbation rate, and the function details, is shown in the top left panel. The function details are shown as a tree structure in the panel. When a predictor function is selected, the function details will be shown in the bottom panel. The top right panel is used to show all the running information. In this case, the loading operation information is displayed. As mentioned in Section 4.1, ASSA-PBN supports two types of definition file. The demonstration of PBNs defined using the the two types format is slightly different in the GUI. When loading a PBN with high level PBN definition format, the node name is shown and functions are shown as expressions on the top left panel; while in the case of a low level PBN definition format, the node ID is shown and functions are shown as indices on the top left panel.

5.4 Converting a PBN from Matlab

ASSA-PBN provides a simple way to define a PBN. Alternatively, it provides a Matlab function to convert a PBN defined in optPBN Matlab toolbox to ASSA-PBN format. We demonstrate here how to convert a PBN defined in optPBN Matlab toolbox to ASSA-PBN format and load it into ASSA-PBN. The apoptosis network [8] is saved as file "apoptosis.mat" in the example folder of the downloaded package. Load it in Matlab using the command "load('apoptosis.mat')" and then run "exportPBNtoASSA('apoptosis.pbn')". With these two commands, the apoptosis network is converted into ASSA-PBN format and saved in a file named "apoptosis.pbn".

5.5 Simulate a PBN

After loading a PBN model, select the menu option "Tools | Simulate", a window as shown in Figure 10 will pop up. We set the length to 20, leave the initial state blank, check the checkbox for showing simulation graph and click the start button. The simulation process will then be launched and the result will be output in the top right panel. By default, the update mode is the one defined in the definition file. User can change it by choosing one listed below update mode. The graph view of the simulation process will be shown in a separate window as shown in Figure 11.

5.6 Computing steady-state probabilities

One major function offered by ASSA-PBN is computing steady-state probabilities of a PBN. ASSA-PBN provides two numerical methods and three statistical methods to perform this task. We demonstrate how to use the two-state Markov chain approach to compute steady-state probabilities of the previously loaded apoptosis network. Select the menu option "Analyse | Steady State Probability | Statistical | Two State Markov Chain". A window as shown in Figure 12a

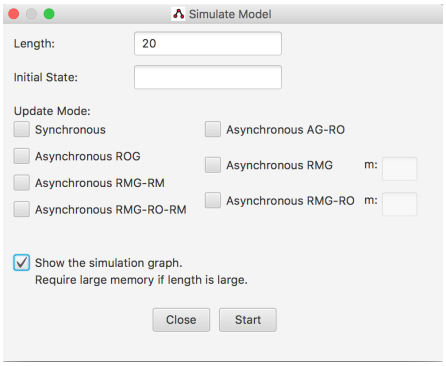


Figure 10: Screenshot of graph display in ASSA-PBN.

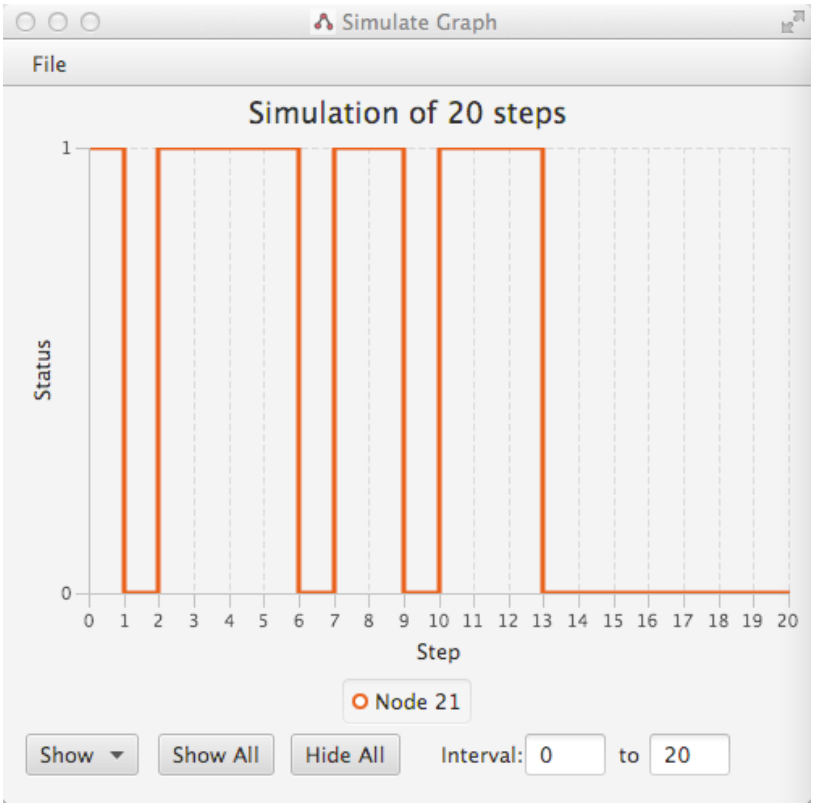
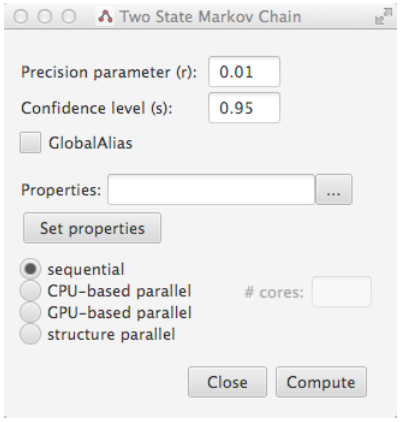
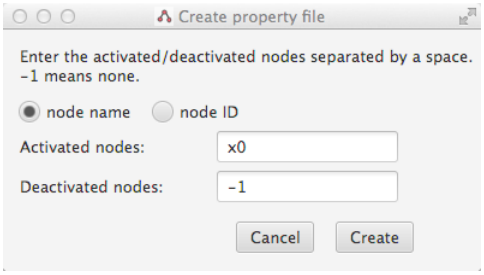


Figure 11: Screenshot of graph display in ASSA-PBN.

pops up. This pop-up window asks for 7 parameters. The precision and confidence level is used to define how precise the computed result should be. We set precision and confidence level to 0.002 and 0.95 respectively. The "GlobalAlias" defines how the alias table should be stored [4]. Checking this box could potentially increase the speed of the two-state Markov chain approach at the cost of



(a) launch the two-state MC approach



(b) create a property

Figure 12: Interfaces of the two-state MC approach.

consuming more memory. "Properties" defines the set of states whose steady-state probabilities are to be computed. It can be assigned either by clicking the three dots button, which will lead to a file selection window; or by clicking the "Set properties" button, which will lead to a window for assigning activated and deactivated nodes as shown in Figure 12b. We click the"Create" button and save the property in a file named "dummy_property.txt". The four radio selections are used to specify how the simulation should be performed. It can be either in a sequential way, or three different parallel ways. The parallel ways use different techniques to gain speedup: the CPU-based parallel makes use of multiple CPU cores; the GPU-based parallel uses multiple GPU cores and the structure parallel makes use of memory to gain speedup []. If the CPU-based parallel is selected, the gray text filed box "# cores" will become available and the text field be filled with the available number of cores in the current machine. In this demonstration, we perform the computation twice, first with the "Parallel" checkbox unchecked and then with it checked and "# of cores" set to 2. The main information, i.e., the extending times, the sample size, the probability, and the time cost of the computation, is shown in the top right panel of ASSA-PBN main interface, as shown in Figure 13. Clearly, the parallel version two-state Markov chain approach reduced the time cost from approximately 5.29s to 2.71s. More detailed information about this run can be seen by clicking the "File View" tab.

5.7 Long-run influence analyses

We demonstrate in this section how to perform long-run influence analyses of a PBN. For the formal definition of long-run influence, we refer to Section2.2 of [5]. We use the apoptosis network named as "apoptosis.pbn" in the folder "examples" of the downloaded software package. After loading the model, select

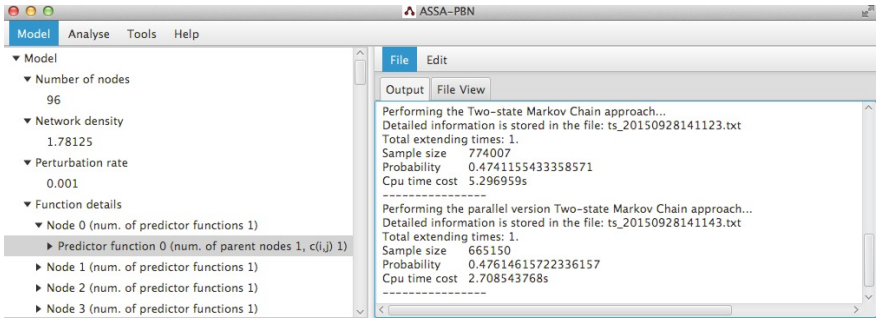
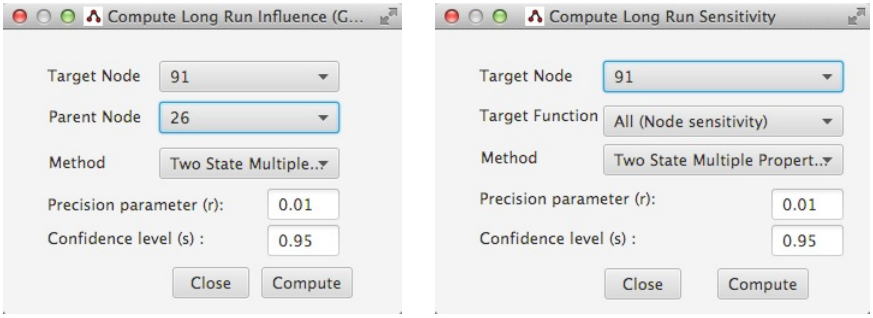


Figure 13: Screenshot of computing with the two-state MC approach.



(a) launch the long-run influence analysis

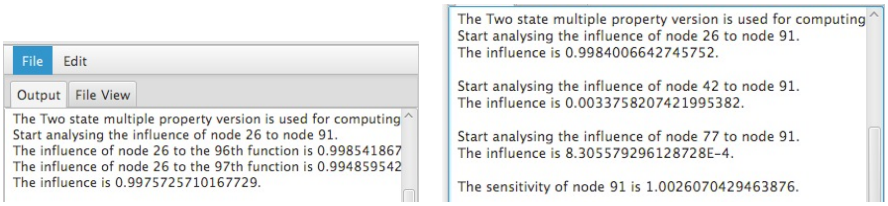
(b) launch the long-run sensitivity analysis

Figure 14: Interfaces of long-run analyses.

the menu option "Analyse | Long-run influence". The sub menu for selecting either gene level influence or function level influence is shown. We choose the gene level. A window as shown in Figure 14a pops up, asking for 5 parameters before the start of the analysis. The first two parameters, i.e., target node and parent node specify the nodes information to be analysed while the other three parameters, i.e., the method, the precision and the confidence level, specify the parameters for computing steady-state probabilities, which are required for the long-run influence analysis. Assuming we want to analyse the long-run influence of complex1 (node 26) on complex2 (node 91), we select 91 as the target node and 26 as the parent node. We keep the other parameters as default values and click "compute". The result of this analysis is then shown in the top right panel, as shown in Figure 15a. The computed influence is approximately 1.00, which means complex1 has a very strong influence on complex2 in the long-run.

5.8 Long-run sensitivity analyses

We continue to show the long-run sensitivity analysis of the dummy network. Click "Analyse", and move the cursor to "Long-run sensitivity". There are two



(a) Result of the long-run influence analysis

(b) Result of the long-run sensitivity analysis

Figure 15: Interfaces of long-run analyses.

choices in the sub menu, i.e., "Node / Function" and "w.r.t. Perturbation". The first choice corresponds to computing the sensitivity of a specific gene or function while the second choice corresponds to computing the sensitivity of a gene with respect to one-bit function perturbations or selection probability perturbations.

Long run sensitivity of a node. We demonstrate first how to compute the long-run sensitivity of node x_1. Select "Node / Function" to get the pop-up window for inputting parameters as shown in Figure 14b. There are 5 parameters in the pop-up window. We select x_1 as the target node. Since we want to analyse the sensitivity for the node, we use the default value, i.e., All (Node sensitivity) for the target function. The other three parameters specifies the parameters for computing steady-state probabilities and we use the default values. After clicking the compute button, the result will be shown in the top right panel, as shown in Figure 15b. The result contains the influence to node x_1 from its three parent nodes, i.e., node x_0, x_1 and x_2.

Long run sensitivity of a node w. r. t. one-bit function perturbation. One-bit function perturbation refers to perturb by changing only one bit in the truth table [6]. We now show how to perturb the first Boolean function of x_0 and compute the long-run sensitivity of x_0 with respect to this one-bit function perturbation. Select the menu option "Analyse | Long-run sensitivity | w.r.t. perturbation | one-bit Function Perturbation". In the pop-up window, put 0 in the top text field since the node ID of x_0 is 0. Select node x_0 and function 0 in the perturbation function area. The function details of this selected function, i.e., the 0th function of node x_0 will be shown in the table on the right part of the window. We double click on the first row of the table to change its value from false to true. See Figure 16 for demonstration. We keep other parameters as default ones. After clicking the compute button, we get the result in the top right panel. Figure 18 (first three lines) shows a screenshot of the result.

Long run sensitivity of a node w. r. t. selection probability perturbation. The long run sensitivity analysis can also be applied to a node with respect to the selection probability perturbation. We show how to perturb the selection probability of the functions of node x_1 and analyse the long-run sensitivity of node x_0 with respect to this selection probability perturbation.

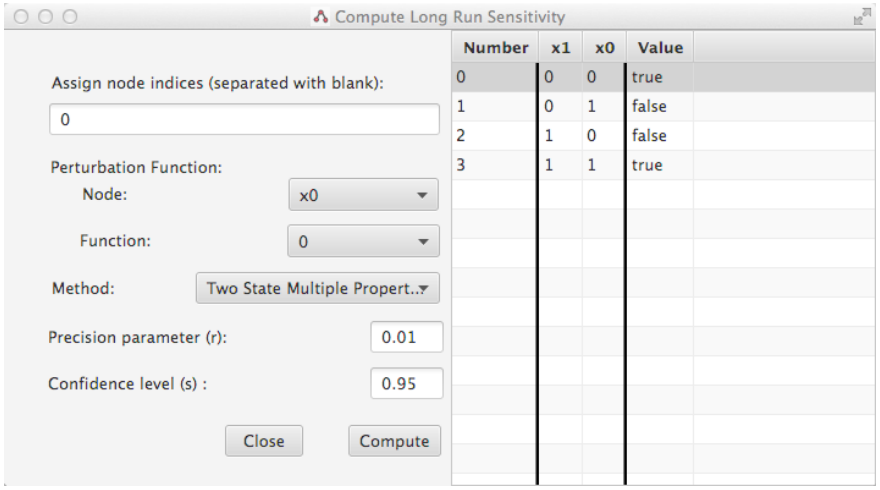


Figure 16: Interface of the long-run sensitivity w.r.t. one-bit function perturbation.

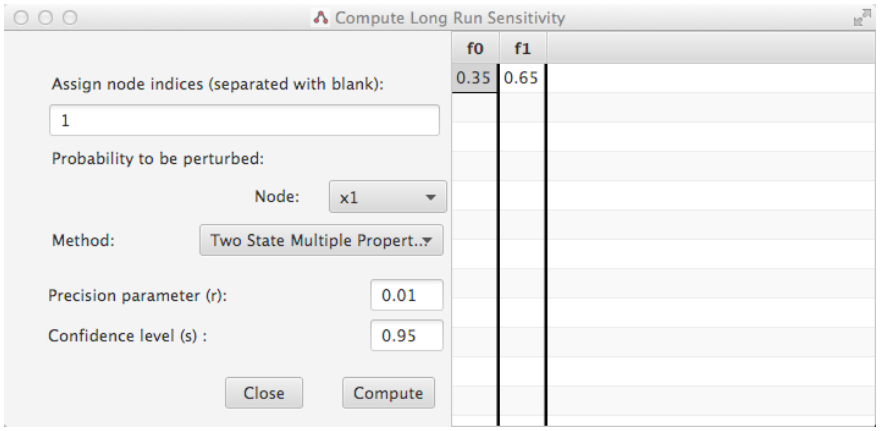


Figure 17: Interface of the long-run sensitivity w.r.t. selection probability perturbation.

Select the menu option "Analyse | Long-run sensitivity | w.r.t. perturbation | Probability Perturbation". In the pop-up window, put 1 in the top text field since the node ID of x_1 is 1. Select node x_1 in the drop down list and the selection probabilities of x_1's functions will be shown in the table in the right side of the window. We edit the selection probabilities of the first function, i.e., f0, to 0.35. After pressing the enter button, the selection probability will be changed. Meanwhile, the selection probability of the other function, i.e., f1, is changed in accordance since they have to sum up to 1. See Figure 17 for demonstration. We keep other parameters as default ones and click the compute button. The result will be shown in the top right panel of the main window as shown in the

last three lines of Figure 18.

```
The Two state multiple property version is used for computing steady-state probabilities.
Start computing the long-run sensitivity in terms of nodes 0.
The long-run sensitivity (with respect to assigned perturbations) is 0.8099383824392357.
The Two state multiple property version is used for computing steady-state probabilities.
Start computing the long-run sensitivity in terms of nodes 1.
The long-run sensitivity (with respect to assigned perturbations) is 0.003936570758858678.
```

Figure 18: Screenshot of the long-run sensitivity w.r.t. perturbation.

5.9 Parameter estimation for instantaneously random PBNs

We distinguish parameter estimation for two types of PBNs: instantaneously random and context-sensitive. In this section, we demonstrate how to perform parameter estimation for the instantaneously random PBNs. In terms of parameter estimation of a PBN, we are interested in estimating the selection probabilities of the Boolean functions of certain nodes. To reach this goal, we first perform experiments in the lab and measure the steady-state probabilities of interested nodes in different conditions, e.g., wild type and mutants. Each condition corresponds to one PBN. We then use parameter estimation techniques to find the optimal parameters which make the PBNs fit the lab experiment data best. To perform parameter estimation in ASSA-PBN, we need to specify the different PBN models under different conditions, the nodes whose selection probabilities are to be estimated, and the lab experiment data. Those information are required to be stored in specification files. We use the specification files in the folder "examples/PE" of the downloaded package to demonstrate parameter estimation of the apoptosis network.

Select menu option "Analyse | Parameter Estimation" to launch the parameter estimation window as shown in Figure 22. Click the "Add PBN files" button and add files "PBN_96_*_1.pbn" in the "examples/PE" folder. Then click the first three dots button and select the file "PBN_96_parameters.txt" to assign nodes to be fitted. Click the second three dots button and select the file "PBN_96_properties_6.txt" to assign experiment data. The parameter estimation method drop down list provides the available parameter estimation methods. There are two methods available currently: the particle swarm method and the differential evolution method. If "Start from random points." is checked, the parameter estimation will start iteration from randomly generated parameters. Otherwise, it will use the parameters specified in the first PBN (usually all the PBNs should have the same parameter values for the same node). The "Adaptive update particle" is specially used for the particle swarm method. If it is checked, the particle swarm method will use the adaptive update method for calculating the next position. See [1] for more details about the adaptive update method. If "Allow parallel evaluation?" is checked, the parameter estimation method will be run in parallel, which means in each iteration, x particles will be evaluated in parallel, where x is determined by the number of cores. If "Plot fitted versus measured value" is checked, the parameter estimation result will

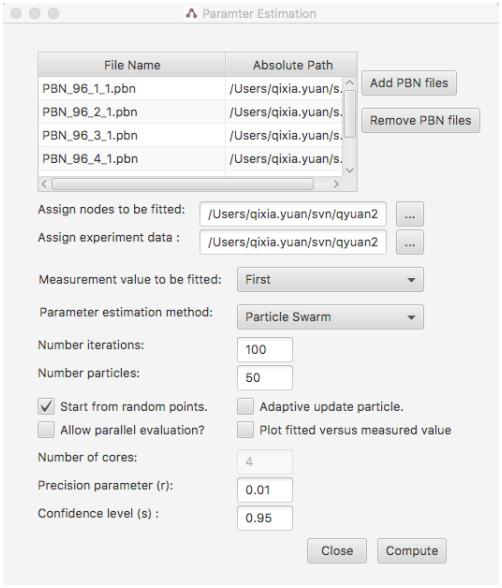


Figure 19: Interface of parameter estimation.

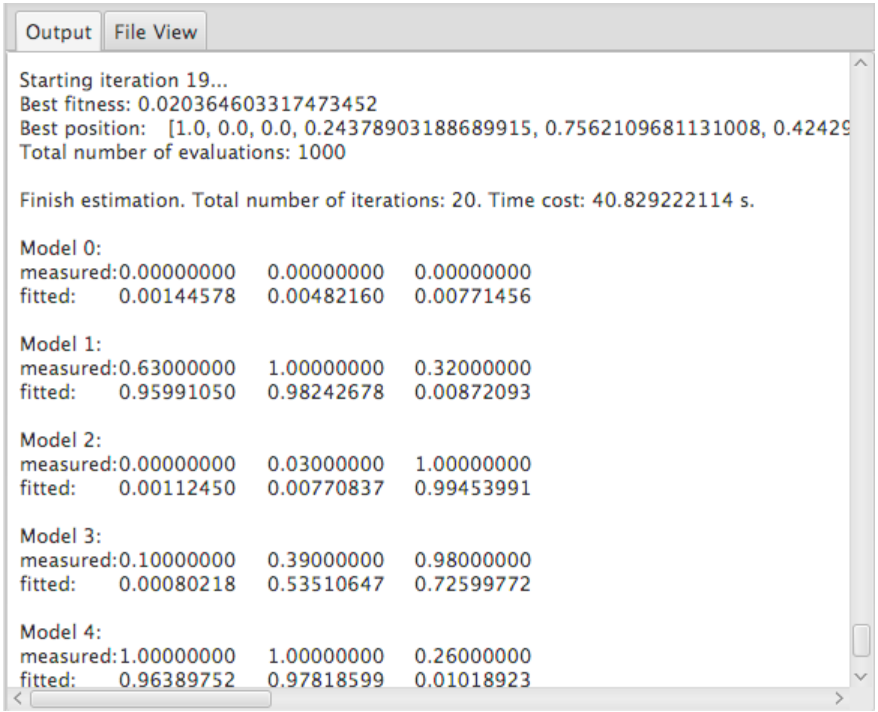


Figure 20: Screenshot of parameter estimation output.

be plotted out at the end of the computation. The last three parameters have the same meaning as shown previously. After setting all the parameters and clicking the "Compute" button, the parameter estimation is launched and the result is shown in the top right panel. The result includes detailed information of each iteration and the final estimated parameter values. A screenshot of the result is shown in Figure 23. In addition, a heat map plot of the results pops up as shown in Figure 21. The heat map is composed of columns (corresponding to

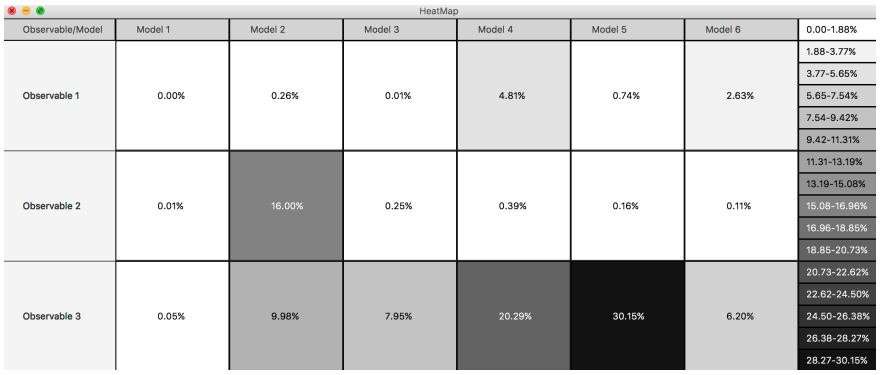


Figure 21: Screenshot of parameter estimation output.

the mutants of the PBN) and rows (corresponding to the measured steady-state probabilities). In our example, there are 6 columns and 3 rows (not including the headers). The value in each cell reflects the quality of the fitness for the corresponding observable of the mutant model. For example, the value in the cell of observable 3 and model 5 is 30.15% and it reflects that the fitness to observable 3 of the PBN (used in experiment 5) is very poor since the cell is flagged as dark and the value is the highest.

5.10 Parameter estimation for context-sensitive PBNs

When performing parameter estimation of a CPBN, we are interested in estimating the switching probabilities of the contexts of the PBN. To reach this goal, we first perform experiments in the lab and measure the steady-state probabilities of interested nodes in different conditions, e.g., wild type and mutants. Each condition corresponds to one PBN. We then use parameter estimation techniques to find the optimal parameters which make the PBNs fit the lab experimental data best. To perform parameter estimation in ASSA-PBN, we need to specify the different PBN models under different conditions, and the lab experimental data. Those information are required to be stored in specification files. We use the specification files in the folder 'examples' of the downloaded package to demonstrate parameter estimation.

To perform parameter estimation, select menu option 'Analyse | Parameter Estimation' to launch the parameter estimation window as shown in Figure 22. Click 'Add PBN files' button to add files 'example-context-sensitive-highlevel.pbn' and 'example-context-sensitive-highlevel_2.pbn' in the 'examples'

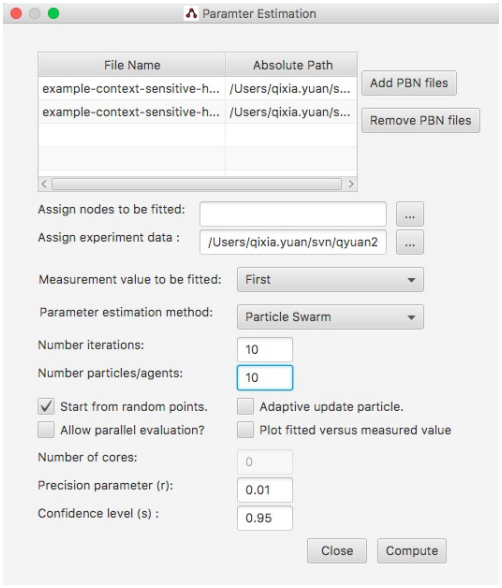


Figure 22: Interface of parameter estimation.

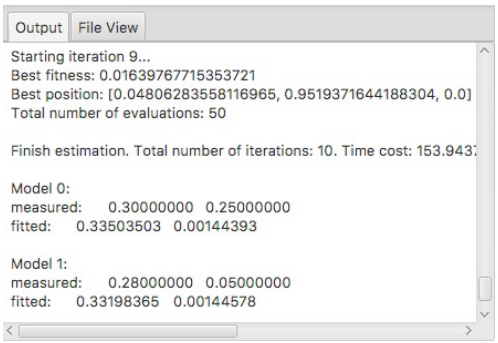


Figure 23: Screenshot of parameter estimation output.

folder. Then click the second three dots button and select the file 'context-sensitive-highlevel_properties.txt' to assign experimental data. We do not assign nodes to be fitted. This option is only valid for instantaneously random PBNs. The format of experimental data file is explained in details in the user guide. The parameter estimation method drop down list provides the available parameter estimation methods. Currently only particle swarm method is available. The following four parameters are related to the particle swarm method. We set them as shown in Figure 22. If 'Allow parallel evaluation?' is checked, the particle swarm method will be run in parallel, which means that in each iteration, n particles will be evaluated in parallel, where n is determined by the

number of cores. We refer to the user guide for the remaining 5 parameters. After setting all the parameters and clicking the 'Compute' button, the parameter estimation is launched and the result is shown in the top-right panel. The result includes detailed information of each iteration and the final estimated parameter values. A screenshot of the result is shown in Figure 23.

5.11 One-parameter profile likelihood

The one-parameter profile likelihood analysis [7] is used to check whether a parameter can be identified or not. The idea of one-parameter profile likelihood analysis is that a series of the values are assigned to the parameter and for each of the assigned value, parameter estimation for other parameters is performed to minimize the cost function. After the last assigned value has been processed, the software plot out the estimated differences for all the assigned values. The estimated difference is calculated as $\Sigma_{i=0}^{n}(\Sigma_{j=0}^{m}(x_{ij} - \bar{x}_{ij})^2)$, where n is the number of mutants, m is the number of measured data for each mutant, x_{ij} is the estimated steady-state probability for the jth data point of mutant i and $\bar{x}_{ij}$ is its corresponding lab-measured data. If the plotted cost function values do not change much with the changing of the assigned value of the parameter, then the parameter is considered as identifiable. Usually, the one-parameter profile likelihood analysis requires a series of PBN models as the case of parameter estimation. The PBN models are composed of different mutants. To perform this analysis, one need to first select one mutant and load it to ASSA-PBN. After loading the model, the menu item "Analyse | one-parameter profile likelihood" becomes available. Select the menu item and a window as shown in Figure 24 will pop up. This pop up window shares a lot of contents with the window for parameter estimation. In fact, only the following three items are different from the parameter estimation pop-up window: 1) select node for identifiability: this item is used to assign the node for one-parameter profile likelihood; 2) select function: when the node for one-parameter profile likelihood is selected, the corresponding functions will shown up in this choice box and one can then select the function to be identified here; 3) step length: the step length should be a value between 0 and 1; it determines the number of values to assign for the selected paramter (function).

In this example, we still use the 6 PBN files in the "PE" folder as for paramter estimation. We use the file "PBN_96_parameters_identifiability.txt" to assign nodes to be fitted, the file "PBN_96_properties_6.txt" to assign experiment data. The rest parameters are kept as in Figure 24. Clicking the button "Compute" and then the computation will start. At the end of the computation, a plot shown as figure 25 will pop up. In this plot, the cost value varies with the changing of the parameter values and sometimes vaires a lot. Therefore, we can conclude that this parameter is identifiable and its value should be around its original value 0.079. The computation might be time expensive since parameter estimation is performed for each of the assigned parameter values.

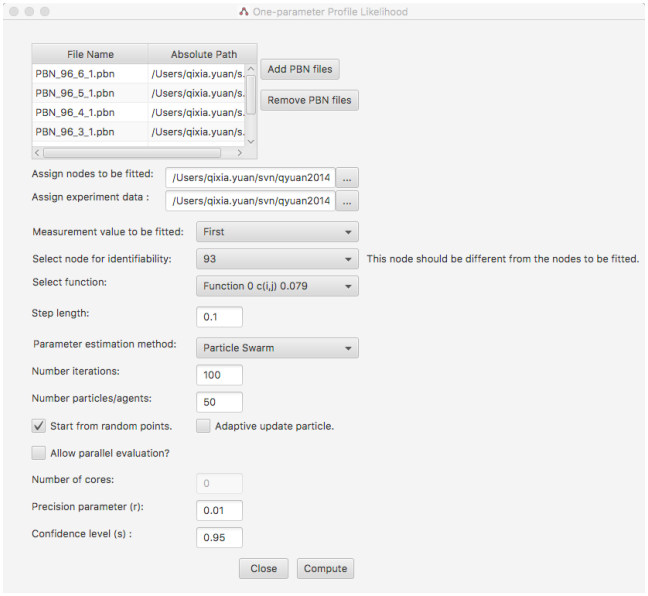


Figure 24: Screenshot of one-parameter profile likelihood window.

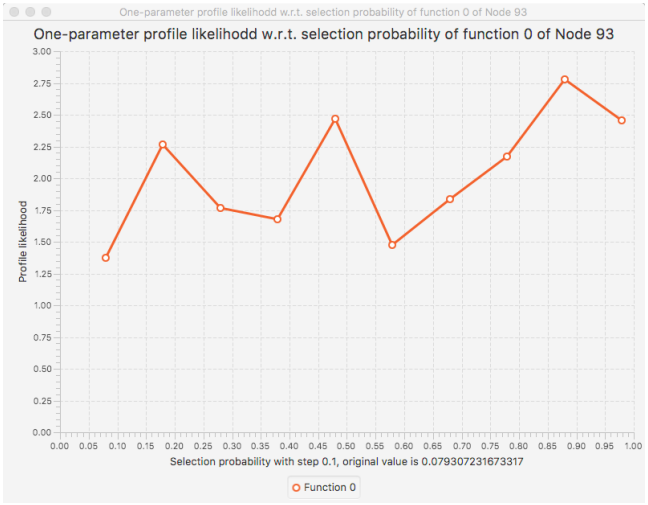


Figure 25: Screenshot of one-parameter profile likelihood analysis plot window.

5.12 Detecting attractors

Attractor detection can be performed for a context-sensitive PBN or a BN. After loading a model, select "Analyse | Attractor Detection". A pop-up window will then show up to ask for parameters used in the detection (see Figure 26). There are three methods for detecting attractors: one non-decomposition method and two decomposition method. The default one is non-decomposition method. If

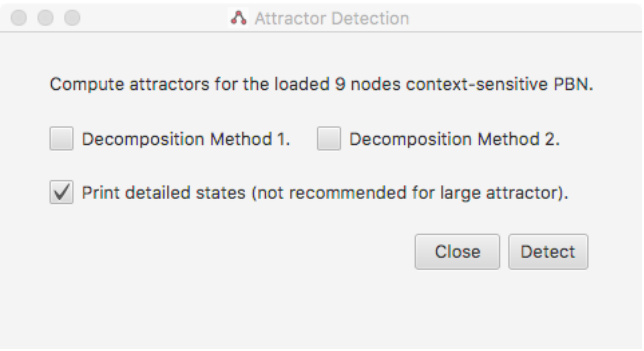


Figure 26: Screenshot of the starting window of attractor detection.

a model is small (the number of node is smaller than 20), the default one is usually enough. If a model is big, the default method may be very slow and one of the decomposition methods is needed. We recommend the decomposition method 2 for most of the cases. The decomposition method 1 refers to the technique mentioned in [10] and the decomposition method 2 refers to the technique mentioned in [3]. Note that if the loaded PBN must be in synchronous or asynchronous ROG mode and it can only be either a context-sensitive PBN or a BN. Otherwise, the "Detect" button will become grey.

5.13 Cancelling a running job

It happens that the user wants to cancel a running job. This can be done by clicking the "Cancel" button close to the status label at the left bottom part of the interface. See Figure 27 for the screenshot of the cancel button.

Figure 27: Screenshot of cancel button.

6 Running ASSA-PBN with Command Line

Below is a list of commands for ASSA-PBN. A command is indicated by a word started with "-" and there might be several words starting with "-" in one command, e.g., the 7th command may contain "-ts", "-epsilon" and "-log".

A. `-ge <number of node> <perturbation rate> <export file name> <export type> [<max function number> <min function number> <max parents node> <min parents node>]`
Generate a random PBN and export it to a file in the ASSA-PBN format or the Matlab format.
`number of node`: The number of nodes in the to be generated PBN.
`perturbation rate`: The perturbation rate in the to be generated PBN.

`export file name`: The name of the file to store the to be generated PBN.
`export type`: The exported type can be either 0, meaning the ASSA-PBN format, or 1, meaning the Matlab format (we call it Matlab format for short).
`max function number`: The maximal number of predictor functions one node can have.
`min function number`: The minimal number of predictor functions one node can have.
`max parents node`: The maximal number of parent nodes one predictor function can have.
`min parents node`: The minimal number of parent nodes one predictor function can have.

B. `-gef <parameter file name> <export file name> <export type>`
Generate a PBN with parameters given in a file and export it to a file.
`parameter file name`: The name of the file that stores the parameters.
`export file name`: The name of the file to store the generated PBN.
`export type`: The exported type can be 0, meaning the ASSA-PBN format, or 1, meaning the Matlab format.

C. `-io <import file name> <isMatlab> <export file name> <export type>`
Load a PBN from a file and export it to a file in the ASSA-PBN format or the Matlab format.
`import file name`: The name of the file that stores the PBN to be imported.
`isMatlab`: True if the model is exported from Matlab-PBN-toolbox and false otherwise. The value is false by default.
`export file name`: The name of the file to store the generated PBN.
`export type`: The exported type can be 0, meaning the ASSA-PBN format, or 1, meaning the Matlab format.

D. `-gauss <model file name> [isMatlab] <property file name> [<precision> <max iteration>]`
Perform the Gauss-Seidel method.
`model file name`: The name of the file that stores the PBN.
`isMatlab`: True if the model is exported from Matlab-PBN-toolbox and false otherwise. The value is false by default.
`property file name`: The name of the property specification file.
`precision`: The accuracy precision of the Gauss-Seidel method. By default, the precision is 10^{-6}. This parameter must be used together with the parameter `max iteration`.
`max iteration`: The maximal allowed iteration number. If the result is

not got within the maximal iteration number, the program will be stopped. By default, the max iteration number is 10,000. This parameter must be used together with the parameter `precision`.

E. `-jacobi <model file name> [isMatlab] <property file name> [<precision> <max iteration>]`
Perform the Jacobi method.
The meanings of the parameters are the same as those in the `-gauss` command.

F. `-ps <model file name> [isMatlab] <precision> <confidence level> <property file name> [-log <log file name>] [useCost]`
Perform the perfect simulation approach.
`model file name`: The name of the file that stores the PBN.
`isMatlab`: True if the model is exported from Matlab-PBN-toolbox and false otherwise. The value is false by default.
`precision`: The precision accuracy of the computed probability.
`confidence level`: The confidence level of the computed probability.
`property file name`: The name of the property specification file.
`-log <log file name>`: This option allows the user to store the log in a specified file. If not specified, the log file name will be decided by the software itself.
`useCost`: This option allows the user to turn on the cost function mode. In the cost function mode, each state is assigned with a cost value based on the property specification file. As long as the costs couple, the states are considered coupled.

G. `-ts <model file name> [isMatlab] <precision> <confidence level> [-epsilon <epsilon>] <property file name> [-log <log file name>] [useGlobalAlias]`
Perform the two-state Markov chain approach.
`-epsilon <epsilon>`: This option allows the user to change the value of ϵ used in the two-state Markov chain approach. By default, the value of ϵ is 10^{-10}. For example, `-epsilon` 10^{-12} can change the value of ϵ to 10^{-12}.
`useGlobalAlias`: The ASSA-PBN simulator can operate in two modes: 1) the *global alias mode* and 2) the *local alias mode*. By default, ASSA-PBN will use the *global alias mode* if there are enough memory. This option allows to change it to the *local alias mode*. To make this change, add `false` at the end of the command line. In the global mode, a joint probability distribution is considered on all possible combinations of predictor function selections for all the nodes and a single alias table for this distribution is constructed. In the local mode, the independence of the PBN is exploited: individual alias tables are constructed for each node of the PBN. In both cases the consecutive state is generated by updating the value of each node with the predictor function selected for this node.

However, in the global mode predictor functions for all nodes are selected simultaneously with the use of only two random numbers, while in the local mode the number of random numbers used is twice the number of nodes. In consequence, the generation of the next state is faster in the global mode, but more expensive in terms of memory usage. For large networks, the local mode is always recommended.
The meanings of the rest parameters are the same as in the `-ps` command.

H. `-tspa <model file name> [isMatlab] <precision> <confidence level> [-epsilon <epsilon>] <property file name> [-log <log file name>] <number of chains>`
Perform the multi CPU cores parallel two-state Markov chain approach. If the model file is exported from Matlab, please add true after the model file name. If you want the simulator to use global alias table for simulation, please add true at the end of the command.
`number of chains`: define how many chains (trajectories) are used in the parallel run. For more details about the meaning of the number of chains, please refer to the Gelman & Rubin method in [2].
The meanings of the rest parameters are the same as in the `-ts` command.

I. `-tspam <model file name> [isMatlab] <precision> <confidence level> [-epsilon <epsilon>] <property file name> [-log <log file name>] <number of chains>`
Perform the multi CPU cores parallel two-state Markov chain approach for checking multiple properties at the same time. If the model file is exported from Matlab, please add true after the model file name. If you want the simulator to use global alias table for simulation, please add true at the end of the command.
This command allows to check several properties using the same trajectory and therefore saving the simulation time.
The meanings of the parameters are the same as in the `-ts` command.

J. `-tsOpt <model file name> <precision> <confidence level> [-epsilon <epsilon>] <property file name> [-log <log file name>] [-m <number of chains>] [-maxe <maximum # combined predictor function>] [-maxc <maximum # combined predictor function per group>] [-sizeone <maximum # of nodes in a group for nodes with only one Boolean function>] [-psize <maximum # nodes in a group for perturbations>]`
Perform the two-state Markov chain approach based on structure-based simulation method. `-m <number of chains>`: structure-based two-state Markov chain approach simulates multiple chains as in the `-tspa` command. This parameter specifies how many chains will be used. The default value is 20.
`-maxe <maximum # combined predictor function>`: this parameter specifies the maximum number of combined predictor function used for grouping. This parameter is restricted by the memory size. The default value

is 12,288.
`-maxc <maximum # combined predictor function per group>`: this function specifies the maximum number of combined predictor function per group. The default value is 1000. A small value of this could lead to more groups but faster preparation time of the structure-based method.
`-sizeone <maximum # of nodes in a group for nodes with only one Boolean function>`: this parameter is used for grouping nodes with only one Boolean function. It specifies the maximum number of nodes in such a group. The default value is 9.
`-psize <maximum # nodes in a group for perturbations>`: the maximum number of nodes in a group for perturbations. The default value is 16.
The meanings of the rest parameters are the same as in the `-ts` command.

K. `-skart <model file name> [isMatlab] <precision> <confidence level> <property file name> [-log <log file name>] [useGlobalAlias]`
Perform the Skart method. To simulate as fast as possible, the simulated trajectory generated with this command has a maximal length of $2^{31}(2,147,483,648)$. In case of bigger trajectory size, please use the `-skartfull` command.
The meanings of parameters are the same as that in the `-ts` command.

L. `-skartfull <model file name> [isMatlab] <precision> <confidence level> <property file name> [-log <log file name>] [useGlobalAlias]`
Perform the Skart method. This command is needed for those cases that the `-skart` command cannot handle. This command can handle huge trajectory size without the maximal length limit, but the computation time is slower than that of the `-skart` command.
The meanings of the parameters are the same as those in the `-skart` command.

M. `-skartpa <model file name> [isMatlab] <precision> <confidence level> <property file name> [-log <log file name>] [useGlobalAlias]`
Perform the multi CPU core parallel Skart approach. If the model file is exported from Matlab, please add true after the model file name. If you want the simulator to use global alias table for simulation, please add true at the end of the command.
The meanings of the parameters are the same as those in the `-tspa` command.

N. `-pe <-pbns # model files, model file names separated with space> <-parameter parameter file name> <-property experiment data file name> <-precision precision> <-confidence confidence level> <-sMethod statistical method> [-parallel] [-m <number of parallel`

`unit>] <-evaluateMethod evaluation method> [-sp <starting point>] <-numParticle number particles> <-numIteration number iterations> [-stop <stop threshold>] [-adaptive] [-peMethod <parameter estimation method>]`
Perform parameter estimation.
`-pbns`: This parameter specifies the number of model file names and all the file names used in this run. The file names should be seperated with blank.
`-parameter`: This parameter is used to specify the file which stores the information of parameters to be estimated.
`-property`: This parameter specifies the file that stores the lab experiment data.
`-precision`: This option together with `-confidence` specifies the precision and confidence level used in the statistical method to compute the steady-state probabilities during the parameter estimation process.
`-sMethod`: This parameter accepts four values. 0 means the two-state Markov chain approach; 1 means the parallel version two-state Markov chain approach; 2 means the parallel version two-state Markov chain approach for checking multiple properties at the same time; 3 means the two-state Markov chain approach for checking multiple properties at the same time.
`-parallel`: When this parameter is provided, the evaluation of particles is in parallel. Otherwise, the evaluation is performed sequentially.
`-m <number of parallel unit>:` This parameter should be used together with the -sMethod option and the -parallel option. If -sMethod is given value 2, the number of parallel unit means the number of chains used in the parallel version two-state Markov chain approach; if statistical method is 3 and the -parallel opiton is given, the number of parallel unit means the number of parallel evaluations used in the particle swam algorithm. Other combinations does not gain speed up.
`-evaluationMethod:` This parameter specifies the evaluation method for calculating the cost function. Currently, it accepts only 1 value: value 0, meaning the particle swarm algorithm; value 1, meaning the differential evolution method.
`-sp <starting point>`: This option specifies the initial values of the parameters (selection probability) to be estimated. It accepts two values. 0 means random starting point; 1 means starting from the values defined in the first PBN.
`-numParticle`: This parameter specifies the number of particles in the particle swarm method or the number of agents in the differential evolution method.
`-numIteration`: This parameter specifies the maximum number of iterations the parameter estimation round.
`-stop <stop threshold>`: This parameter specifies the stop threshold of the parameter estiamtion method. Once the cost function reach the threshold, the particle swarm algorithm will be stopped.

`-adaptive`: This is used only for the particle swarm method. When this parameter is set, update each particle in an adaptive way. `-peMethod`: This option is used to specify the parameter estimation method. Currently, there are two options: 0, meaning the particle swarm method; 1, meaning the differential evolution method.

O. `-sim <model file name> [isMatlab] <output file name> <simulation steps> [-initial initialState] [-updateMode updateMode] [-m m]`
Simulate the PBN defined in `model file name` with the steps specified with `simulation steps` and output the simulated trajectory into the given file `output file name`. `m` represents the number of nodes to be updated and is only meaningful for asynchronous_rmg and asynchronous_mro. By default, the update mode is the one defined in the model definition file and m is 0. The format of the `update modes` are synchronous, rog, rmg, rrmg, mro, rmro, aro.

P. `-infN <model file name> <source node index> <target node index> [-sm <statistical method >] [precision] [confidence level]`
Perform the influence analysis of the source node on the target node.
`model file name`: This option specifies the PBN file.
`source node index`: This option specifies the node that the influence is from.
`target node index`: This option specifies the node that the influence is on.
`-sm <statistical method>:` This option allows to select the statistical methods used for computing steady-state probabilities. The values it accepts are the same as in the previous one. By default, the value is set to 3, which means the two-state Markov chain approach for checking multiple properties at the same time.
`precision`: This option together with `confidence level` specifies the precision and confidence level used in the statistical method to compute the steady-state probabilities. By default, the precision is set to 0.01 and the confidence level is set to 0.95.

Q. `-infF <model file name> <source node index> <target node index> <target function index> [-sm <statistical method >] [precision] [confidence level]`
Perform the influence analysis of the source node on the target function. The parameters are similar to those in the `-infN` option. The extra parameter <target function index> specifies the function index of the target node.

R. `-senN <model file name> <target node index> [-sm <statistical method >] [precision] [confidence level]`
Compute the sensitivity of the target node. The parameters have similar meanings as in the `-infN` option.

S. `-senF <model file name> <target node index> <target function index> [-sm <statistical method >] [precision] [confidence level]`
Compute the sensitivity of the target function. The parameters have similar meanings as in the `-infF` option.

T. `-senP <model file name> <perturbed model file name> [-sm <statistical method>] [-p <precision> <confidence level>] <list of node indices separated with blank>`
compute the long-run sensitivity with respect to 1-bit function/selection probability perturbations.

U. `-v`
Show version information.

V. `-h`
Show help information.

Note that the input and output files used in ASSA-PBN cannot be named as "true" or "false" (case insensitive).

ASSA-PBN is launched with default settings of the Java virtual machine (JVM). It also supports to change the Java heap size of the JVM when launching the program. This can be done by adding the standard Java command at the beginning of ASSA-PBN command line. It supports the following three options and allows them to be used at the same time. See Section 6.9 for examples.

- -Xms⟨size⟩ set initial Java heap size
- -Xmx⟨size⟩ set maximal Java heap size
- -Xss⟨size⟩ set Java thread stack size

All the example files used in this section can be found in the folder "examples" in the downloaded package.

6.1 Generating a PBN

We give an example in figure 28 to show how to generate a random PBN by providing parameters directly in the command line. The example generates a PBN with 8 nodes, which has a maximum of 8 predictor functions for each node and a maximum of 7 parent nodes for each function, and we store it to a file named `PBN-test1.pbn`. This can be easily done with the following command line: `"assa -ge 8 0.01 PBN-test1.pbn 0 8 1 7 1"`, where the first number `8` indicates the number of nodes in the PBN, `0.01` denotes the perturbation rate, `PBN-test1.pbn` is the file name to store the generated PBN, the number `0` denotes the exported type is in the ASSA-PBN format, the two numbers `8` and `1` define the possible maximum and minimum predictor functions of each node, and the last two numbers define the possible maximum and minimum numbers of parent nodes of the predictor functions. ASSA-PBN will choose for each node a random number between 1 and 8 as the number of predictor functions and

for each predictor function a random number between 1 and 7 as the number of parents nodes. Since a node has to have at least one predictor function and a predictor function has to have at least one parent node, their minimum values are given as 1 in case there is no explicit requirements on the minimum values. The number 0 in the command line refers to the ASSA-PBN format. It's also possible to use number 1, which means the Matlab format. The density $\mathcal{D}$ in the output information is computed with the following formula:

$$\mathcal{D} = \frac{1}{n} \sum_{i=1}^{N_{\mathcal{F}}} \omega(i),$$

where n is the number of nodes in the PBN, $N_{\mathcal{F}}$ is the total number of functions in the PBN, $\omega(i)$ is the number of parents node for the ith function.

```
./assa -ge 8 0.01 PBN-test1.pbn 0 8 1 7 1
One 8 nodes PBN is generated.  Its density is 19.0.
The 8 nodes PBN is exported to file PBN-test1.pbn in ASSA-PBN
format.
```

Figure 28: Output of generating a random PBN by providing parameters in command line.

It's also possible to generate a random PBN by providing parameters in a file. The file defining parameters should contain four lines to provide the number of nodes, the number of predictor functions for each node, the number of variables for each Boolean function and the perturbation rate. We show in Figure 29 an example of parameter definition file and in Figure 30 how to generate a random PBN using this parameter file.

```
//number of nodes
3
//number of functions for each node
1 2 2
//number of parent nodes for each function
2 1 1 2 1
//perturbation rate
0.01
```

Figure 29: Example of parameter definition file.

6.2 Converting a PBN from Matlab-PBN-toolbox format to ASSA-PBN format

For the convenience of the user, we design the function for converting models which have already been defined in the Matlab-PBN-toolbox [8] to the ASSA-PBN format. As described in 4.1, defining a PBN model in ASSA-PBN is not

```
./assa -gef parameters.txt PBNFromParameter.m 1
One 3 nodes PBN is generated.  Its density is
2.3333333333333335.
The 3 nodes PBN is exported to file PBNFromParameter.m in
Matlab format.
```

Figure 30: Output of generating a random PBN by providing parameters in a file.

a difficult task; therefore, we recommend to define the model directly in the ASSA-PBN format if it's not defined in Matlab-PBN-toolbox. We use the file `PBNFromParameter.m` generated in Section 6.1 to define a PBN in Matlab-PBN-toolbox. To export the PBN model, one needs to copy the `exportPBNtoASSA.m` file to the working directory of Matlab and run the command as shown in Figure 31. A PBN definition file named `exportedFromMatlab.pbn` will be generated and it can be analysed with ASSA-PBN.

```
exportPBNtoASSA('exportedFromMatlab.pbn')
```

Figure 31: Exporting a PBN from Matlab-PBN-toolbox.

6.3 Loading a PBN and exporting it to a file

We will load the file `exportedFromMatlab.pbn` generated in Section 6.2 and export it to a file named `exportedFromASSA.m`. See Figure 32 for the input and output for running this command in ASSA-PBN. Theoretically, the content of the exported file `exportedFromASSA.m` should be the same as that of `PBNFromParameter.m` since `exportedFromASSA.m` is got by two converting operations from `PBNFromParameter.m`. We observed a slight difference in the cij values. This difference is due to that the precision used in the tool ASSA-PBN is smaller than that in Matlab. After loading the model to Matlab, the difference will disappear.

```
./assa -io exportedFromMatlab.pbn true exportedFromASSA.m 1
Start loading model...
Finish loading model.  Time cost:  0.001635s.
Start exporting model...
The 3 nodes PBN is exported to file exportedFromASSA.m in
Matlab format.
```

Figure 32: Loading a PBN and exporting it to a file.

6.4 Simulate a PBN

We demonstrate how to simulate a PBN stored in the file "examples/PE/PBN_96_1_1.pbn" for 20 steps and export the result to file "simout.txt" with command line. Run the following command: "./assa -sim ../examples/PE/PBN_96_1_1.pbn simout.txt 20". The simulated trajectory will be shown in the terminal and stored in the file "simout.txt".

6.5 Performing the numerical methods

Currently, `ASSA-PBN` supports two numerical methods, i.e., the Gauss-Seidel method and the Jacobi method. We show in Figure 33 for the command line input and output of performing the Gauss-Seidel method. The parameter `-gauss` indicates that the analyser uses the Gauss-Seidel method, `PBN-test1.pbn` is the PBN file that we have generated in Section 6.1 and `property-1.pro` is the property specification file. Copy `property-1.pro` file from the "examples" folder to the "bin" folder. The iteration number, the time cost and the steady-state probability that we want to check are outputted. By default, the accuracy of the probability is 10^{-6}. It can be changed by adding at the end of the command the new accuracy and the max iteration numbers. It is also possible to output the steady-state distribution for all states in the PBN by changing the property specification file name to `-dis` in the command line. The Jacobi method can be called similarly by changing `-gauss` to `-jacobi` in the command line.

```
./assa -gauss PBN-test1.pbn property-1.pro
Start loading model...
Finish loading model.  Time cost:  0.004908s.
Performing the Gauss-Seidel method on the PBN from file
PBN-test1.pbn
The Gauss-Seidel method ends within 12 iterations.
CPU Time cost:  0.10797s.
The probability to check is 0.02916671121948796.
```

Figure 33: Example of running the Gauss-Seidel method.

6.6 Performing the perfect simulation approach

The perfect simulation approach can be run using the following command line: `"assa -ps PBN-test1.pbn 0.01 0.95 property-1.pro -log psOutput.log"`, where `-ps` indicates that the analyser uses the perfect simulation algorithm. The first two numbers `0.01` and `0.95` define the precision and the confidence level respectively, and the last pat `-log ps1.log` shows that the log information is stored in the file `psOutput.log`. Figure 34 shows the output of the perfect simulation algorithm on the 8 node PBN. The total simulation steps in the output are computed using the following formula:

$$2^n \sum_{i=1}^{N_s} c(i),$$

where n is the number of nodes in the PBN, N_s is the number of required samples (1046 in this example), $c(i)$ is the coupling step of the ith sample. The probability given by the perfect simulation method differs by about 0.006 with the one given by the Gauss-Seidel method in Section 6.5, which can be considered as the theoretical value. The difference, i.e., about 0.0014, is still within the precision requirement 0.01. Note that a smaller precision requirement will result in a probability closer to the theoretical value.

```
./assa -ps PBN-test1.pbn 0.01 0.95 property-1.pro -log
psOutput.log
Start loading model...
Finish loading model.  Time cost:  0.004479s.
Performing the perfect simulation algorithm on the PBN from
file PBN-test1.pbn
Detailed information is stored in the file psOutput.log.
Avg.  coupling steps 13.520076481835565
Simulation steps     3620352.0
Number of samples    1046.0
CPU time cost        0.647181s
Probability          0.027724665391969407
```

Figure 34: Example of running the perfect simulation approach.

6.7 Performing the two-state Markov chain approach

The two-state Markov chain approach can be launched using the following command line: `"assa -ts PBN-test1.pbn 0.01 0.95 property-1.pro -log ts.log"`. The parameter `ts` indicates that the analyser uses the two-state Markov chain approach. The numbers `0.01` and `0.95` define the precision and the confidence level, respectively. `PBN-test1.pbn` is the PBN file that we have generated previously. The ASSA-PBN simulator can operate in two modes: 1) the *global alias mode* and 2) the *local alias mode*. By default, ASSA-PBN will use the *global alias mode* if there is enough memory. One can change it to the *local alias mode* by adding `false` at the end of the command line. In the global mode, a joint probability distribution is considered on all possible combinations of predictor function selections for all the nodes and a single alias table for this distribution is constructed. In the local mode, the independence of the PBN is exploited: individual alias tables are constructed for each node of the PBN. In both cases the consecutive state is generated by updating the value of each node with the predictor function selected for this node. However, in the global mode predictor functions for all nodes are selected simultaneously with the use of only two random numbers, while in the local mode the number of random numbers used is twice the number of nodes. In consequence, the generation of the next

state is faster in the global mode, but more expensive in terms of memory usage. For large networks, the local mode is always recommended. Information about this execution will be shown in the command window as in Figure 35. Besides, a log file with more detailed information, e.g., the number of extensions of the two-state Markov chain approach on this particular model will also be generated and stored in the file `ts.log`.

Figure 35 shows the example to perform the two-state Markov chain approach. The two-state Markov chain approach poses an own parameter named ϵ, which is set to 10^{-10} by default in ASSA-PBN. It can be changed by adding `-epsilon <epsilon value>` after the confidence level in the command. General information about this execution will be output in the command window as shown in Figure 35. Besides, a log file with more detailed information, e.g., the number of extensions of the two-state Markov chain approach on this particular model will also be generated.

```
./assa -ts PBN-test1.pbn 0.01 0.95 property-1.pro -log ts.log
Start loading model...
Finish loading model.  Time cost:  0.004423s.
Performing the two-state Markov chain approach on the PBN from
file PBN-test1.pbn.
Detailed information is stored in the file ts.log.
Sample size   1262
Probability   0.027156549520766772
CPU time cost 0.09973s
```

Figure 35: Example of running the two-state Markov chain approach.

6.8 Performing the Skart method

Similar to the two-state Markov chain approach, the Skart method can be launched using the following command line: `"assa -skart PBN-test1.pbn 0.01 0.95 property-1.pro -log skart1.log"`. The parameter `-skart` indicates that the analyser uses the Skart method. There are also an output in the command window and a more detailed output stored in a log file `skart1.log` for the Skart method. Figure 36 shows the output in the command window. The Skart method will compute a confidence interval which satisfies the precision requirement. The confidence interval corresponds to the `CI half-length` in the output. In this case, the confidence interval is [0.02952587482158517, 0.03424365642841483]. If the analysis requires a trajectory size bigger than 2^{31}, the `-skartfull` command is needed. The usage of the `-skartfull` command is the same as the `-skart` command.

6.9 Changing the default Java heap size

ASSA-PBN allows to change the default Java heap size when launching the program by adding the standard Java command at the beginning of ASSA-PBN

```
./assa -skart PBN-test1.pbn 0.01 0.95 property-1.pro -log
skart1.log
Start loading model...
Finish loading model.  Time cost:  0.005912s.
Performing the Skart method on the PBN from file
PBN-test1.pbn.
Detailed information is stored in the file skart1.log.
Sample size   20480
Probability   0.031884765625
CI half-length0.00235889080341483
CPU time cost 0.302707s
```

Figure 36: Output of the Skart method.

command line. We show an example for changing the initial Java heap size to 256M and the maximum Java heap size to 2G in Figure 37.

```
./assa -Xms256M -Xmx2G -gauss PBN-test1.pbn property-1.pro
Start loading model...
Finish loading model.  Time cost:  0.004588s.
Performing the Gauss-Seidel method on the PBN from file
PBN-test1.pbn.
The Gauss-Seidel method ends within 12 iterations.
CPU Time cost:  0.105934s.
The probability to check is 0.02916671121948796.
```

Figure 37: Changing default Java heap size.

6.10 Performing the two-state Markov chain approach in parallel

Starting from the version 1.0.4, ASSA-PBN supports to analysing a steady-state property of a PBN in a parallel way using multiple CPU cores. Figure 38 shows an example to check a single property using the two-state Markov chain approach in a parallel way. ASSA-PBN also supports to check multiple properties at the same time using the two-state Markov chain approach in a parallel way. This function can be launched using the command `-tspam`. The multiple properties should be stored in a single file. See Section 4.3 for how to define multiple properties in one file.

6.11 Performing the long-term influence analysis

The long-term influence analysis shown in section 5.7 can be launched in command line with the command shown in Figure 39. The output of this run is shown as well.

```
./assa -tspa ../examples/PBN_50.pbn 0.001 0.95 property-1.pro
-log parallelrun.txt 40
Start loading model...
Finish loading model.  Time cost:  2.349563s.
Run the two-state approach in parallel for computing PBN_50.pbn
for property property-1.pro...
number of processors:  4
Required number of chains is 40, the program sets it to 4 (the
number of processors in the computer).

Detailed information is stored in parallelrun.txt
Total extending times:  0

m=4, n=81920, burnIn=81920, Nmax=1360, each chain
length=163840, fetch samples every 1 step(s).
precision=0.001, probability=3.41796875E-4, time
cost=4.213332899s.
```

Figure 38: Checking a single property using the two-state Markov chain approach in a parallel way.

```
./assa -infN ../examples/PE/PBN_96_1_1.pbn 26 91
Start loading model from file model/PBN_96_6_1.pbn.
This is a model exported from Matlab.
Finish loading model.  Time cost:  0.013777s.
The Two state multiple property version is used for computing
steady-state probabilities.
Start analysing the influence of node 26 to node 91.
The influence of node 26 to the 96th function is
0.999023087997855.
The influence of node 26 to the 97th function is
0.9956496066492052.
The influence is 0.9981350879879183.
```

Figure 39: Long-term influence analysis in command line.

6.12 Performing the long-term sensitivity analysis

The long-term sensitivity analysis shown in section 5.8 can be launched in command line with the command shown in Figure 40. The output of this run is shown as well.

6.13 Performing parameter estimation

The parameter estimation shown in section 5.9 can be launched in command line with the command shown in Figure 41.

```
./assa -senN ../examples/PE/PBN_96_1_1.pbn 91
Start loading model from file model/PBN_96_6_1.pbn.
This is a model exported from Matlab.
Finish loading model.  Time cost:  0.015003s.
The Two state multiple property version is used for computing
steady-state probabilities.
Start analysing the influence of node 26 to node 91.
The influence is 0.9965752309079929.

Start analysing the influence of node 42 to node 91.
The influence is 0.0032458966230724823.

Start analysing the influence of node 77 to node 91.
The influence is 9.457500518534018E-4.

The sensitivity of node 91 is 1.0007668775829188.
```

Figure 40: Long-term influence analysis in command line.

```
./assa -pe -pbns 6 ../examples/PE/PBN_96_1_1.pbn
../examples/PE/PBN_96_2_1.pbn ../examples/PE/PBN_96_3_1.pbn
../examples/PE/PBN_96_4_1.pbn ../examples/PE/PBN_96_5_1.pbn
../examples/PE/PBN_96_6_1.pbn -parameter
../examples/PE/PBN_96_parameters.txt -property
../examples/PE/PBN_96_properties_6.txt -precision 0.01
-confidence 0.95 -numParticle 10 -numIteration 5
```

Figure 41: Parameter estimation in command line.

7 Update log of ASSA-PBN

A. Version 3.0.1 Updated on: 03/05/2017.
Main changes:

   (a) add support for Windows OS.

B. Version 3.0.0. Updated on: 20/10/2017.
Main changes:

   (a) add attractor detection function;

   (b) add support for CPBNs, this includes the high-level definition language for CPBN, simulation, steady-state computation, parameter estimation, and attractor detection.

C. Version 2.0.4. Updated on: 01/04/2017.
Main changes:

   1) add support for various asynchronous updating schemes;

2) add identifiability analysis;

3) add heat map plot for parameter estiamtion;

4) add comparative plot (fitted value vs measured value) for parameter estimation results;

5) add support for multiple measurment values in parameter estiamtion;

6) add the differential evolution method for parameter estimation;

7) add support for comments in property specification files.

D. Version 2.0.3. Updated on: 07/12/2016.
Main changes:
1) allow constant functions for high level PBN definition files;
2) add display for updating mode (synchronous or asynchronous).

E. Version 2.0.2. Updated on: 29/04/2016.
Main changes:
1) add support for GPU computation;
2) add support for level PBN definition format.

F. Version 2.0.1. Updated on: 9/10/2015.
Main changes:
1) add GUI;
2) add long-term influence analysis;
3) add long-term sensitivity analysis;
4) add parameter estimation.

G. Version 1.0.4. Updated on: 25/07/2015.
Main changes:
1) add multiple CPU implementations for the two-state Markov chain approach and the Skart method;
2) fix the bug for losing the order information of the parent nodes indices when loading a model definition file;
3) removing unnecessary libraries.

H. Version 1.0.3. Updated on: 30/01/2015.
Main changes:
1) add command line parameters for defining Java heap size;
2) optimize the Jacobi method by storing the transition matrix in memory instead of repeated on-the-fly generating.

I. Version 1.0.2. Updated on: 30/11/2014.
Main changes: 1) fix the bug for not being able to define whether the model is converted from Matlab when running the Skart method.

J. Version 1.0.1. Updated on: 07/11/2014.
Main changes:
1) add optimization for the initial trajectory size used in the two-state Markov chain approach; 2) add output for loading models.

References

[1] Alireza Alfi and Hamidreza Modares. System identification and control using adaptive particle swarm optimization. *Applied Mathematical Modelling*, 35(3):1210–1221, 2011.

[2] A. Gelman and D.B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992.

[3] A. Mizera, J. Pang, H. Qu, and Q. Yuan. Taming asynchrony for attractor detection in large Boolean networks (technical report). Available online at `http://arxiv.org/abs/1704.06530`, 2017.

[4] A. Mizera, J. Pang, and Q. Yuan. ASSA-PBN: a tool for approximate steady-state analysis of large probabilistic Boolean networks. In *Proc. 13th International Symposium on Automated Technology for Verification and Analysis*, volume 9364 of *LNCS*, pages 214–220. Springer, 2015. Available at `http://satoss.uni.lu/software/ASSA-PBN/`.

[5] A. Mizera, J. Pang, and Q. Yuan. Reviving the two-state markov chain approach (technical report). *IEEE/ACM Transactions on Computational Biology and Bioinformatics (special issue of APBC 2017)*, pages xx–xx, 2017. to appear.

[6] Xiaoning Qian and Edward R Dougherty. On the long-run sensitivity of probabilistic Boolean networks. *Journal of Theoretical Biology*, 257(4):560–577, 2009.

[7] A. Raue, C. Kreutz, T. Maiwald, J. Bachmann, M. Schilling, U. Klingmüller, and J. Timmer. Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. *Bioinformatics*, 25(15):1923, 2009.

[8] P. Trairatphisan, A. Mizera, J. Pang, A.-A. Tantar, and T. Sauter. optPBN: An optimisation toolbox for probabilistic boolean networks. *PLOS ONE*, 9(7):e98001, 2014.

[9] Panuwat Trairatphisan, Andrzej Mizera, Jun Pang, Alexandru Adrian Tantar, Jochen Schneider, and Thomas Sauter. Recent development and biomedical applications of probabilistic boolean networks. *Cell Communication and Signaling*, 4(6):1–25, 2013.

[10] Qixia Yuan, Hongyang Qu, , Jun Pang, and Andrzej Mizera. Improving BDD-based attractor detection for synchronous Boolean networks. *Science China Information Sciences*, 59(8):080101, 2016.