User Guide of ASSA-PBN Version 1.0

July 27, 2015

Contents

1 Introduction

ASSA-PBN is a tool specially designed for approximate steady-state analysis of large probabilistic Boolean networks (PBNs). The approximate steady-state analysis is crucial for large PBNs, which naturally arise in the domain of Systems Biology. We refer to [5] for the theoretical background of the steady-state analysis of PBNs. ASSA-PBN provides different solutions for different size PBNs. For small PBNs, ASSA-PBN supports with two numerical methods, i.e., the Jacobi method and the Gauss-Seidel method, and one statistical method, i.e., the perfect simulation approach [6]. For large PBNs, ASSA-PBN provides the two-state Markov chain approach [1, 2] and the Skart method [3].

2 Modules

ASSA-PBN contains three major parts (see Figure 1): a PBN constructor, a PBN simulator, and a PBN analyser. Based on the specified parameters or model file, the constructor can build a PBN. The simulator takes a PBN generated by the constructor as input and performs simulation of the PBN efficiently to produce trajectories. The key function of ASSA-PBN is to compute the steady-state probability for a set of states of the PBN which is defined in a property file. This is achieved by the analyser in either a numerical manner (for small PBNs) or a statistical manner (for large PBNs). The implemented numerical methods require the transition matrix of a PBN as input, which is supplied by the constructor; while the implemented statistical methods require simulated trajectories of the PBN as input, which are supplied by the simulator. 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. The ASSA-PBN program package can be downloaded at `http://satoss.uni.lu/software/ASSA-PBN/assa-pbn-1.0.zip`. The cur-

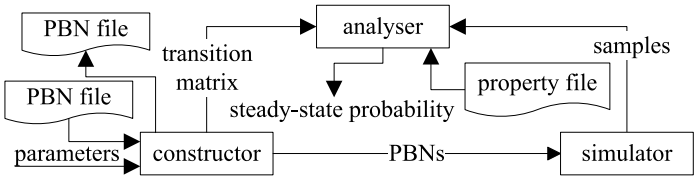

Figure 1: Structure of ASSA-PBN.

rent version ASSA-PBN provides the following 4 functions:

1. generating a random PBN model;
2. loading a PBN model defined in the ASSA-PBN format or the Matlab-PBN-toolbox (optPBN toolbox[1]) format [4];

[1]In the later part of the user guide, we will only mention the Matlab-PBN-toolbox. But we consider the Matlab-PBN-toolbox also covers the optPBN toolbox since the optPBN toolbox is an extension of the Matlab-PBN-toolbox and it uses the same format as Matlab-PBN-toolbox.

3. exporting a PBN model from ASSA-PBN to Matlab-PBN-toolbox;

4. computing the steady-state probabilities of a PBN using different methods, e.g., the Jacobi method, the Gauss-Seidel, the perfect simulation approach, the two-state Markov chain approach, and the Skart method;

See Section 5 for the detailed instructions for running ASSA-PBN.

3 Installation

ASSA-PBN does not require installation. It can be run in command line after extracted from the downloaded package. Figure 2 shows an example of extracting and running ASSA-PBN. ASSA-PBN can be run on Mac OS, Linux and

```
unzip assa-pbn-1.0.zip
./assa -v
```

Figure 2: Extracting and running ASSA-PBN.

Windows 7 or later. It requires Java 6 (also called version 1.6) or later. To run ASSA-PBN in Linux or Mac OS, use the command `./assa <parameters>`. For Windows, use the command `assa <parameters>`.

4 Input Files

ASSA-PBN accepts three types of input files, namely the PBN definition file, the parameter definition file, and the property specification file. ASSA-PBN does not require the file name to have a special suffix, but the ".pbn" is recommended for the PBN definition files, the ".par" is recommended for parameter definition files and the ".pro" is recommended for the property specification files.

4.1 PBN definition files

We will use an example to show the format 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 ASSA-PBN, the node of a PBN is defined by non-negative integers. 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 ASSA-PBN. For example, the first boolean function $f_1^{(0)} = x_0 \wedge x_1$ is converted to 0 1 1 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.

node name	function	probability
x_0	$f_1^{(0)} = x_0 \wedge x_1$	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.

The corresponding ASSA-PBN definition file is shown in Figure 3. 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.

ASSA-PBN accepts two types of PBN definition files. One is shown above; the other one is 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 5.1 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. It contains even number of lines and every two lines define one set of states. For each two lines, the first line specifies those indices of the nodes, whose values should be 1 (active); while the second

```
//The first line is the number of nodes in the PBN.
3
//The second line indicates the number of Boolean functions for each
//node.
1 2 2
//The third 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 1 1 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 3: The corresponding PBN definition file of the PBN shown in Table 1.

line specifies those indices of the nodes, whose values should be 0 (inactive). The indices in the same line are separated with a space. Figure 4 shows two examples of a PBN with three nodes. In the first example, we are interested in those states that nodes 0 and 2 are active and node 1 is inactive. In fact, there is only one interested state, i.e., 101. In the second example, we are interested in those states that node 1 is active. The second line of Figure 4b is -1, indicating that those inactive nodes are not relevant in this property. This property file defines 4 states, i.e., 010, 011, 110, 111. In most cases, ASSA-PBN only requires an input of one set of states, which means that the property specification file contains only two lines. 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. Each set of states is defined with two lines in the above mentioned way. For example, if we put the two sets of states defined in Figure 4 and 4b

in one file, we form a new property specification file which defines two sets of states. See figure 4c for this property specification file.

```
0 2
1
```

(a) The value of nodes 0 and 2 is 1 and the value of node 1 is 0.

```
1
-1
```

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

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

(c) Two properties in one file.

Figure 4: Example of property specification files.

5 Running ASSA-PBN with Command Line

ASSA-PBN provides a command line version at the moment. Below is a list of commands for ASSA-PBN.

1. `-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.

2. `-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.

3. `-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.

4. `-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`.

5. `-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.

6. `-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.

7. `-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.

8. `-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 [**?**].
   The meanings of the rest parameters are the same as in the `-ts` command.

9. `-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.

10. `-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.

11. `-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.

12. `-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.

13. `-v`
Show version information.

14. `-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 5.8 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.

5.1 Example 1: generating a PBN

We give an example in figure 5 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.

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 6 an example of parameter definition file and in Figure 7 how to generate a random PBN using this parameter file.

```
./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 5: Output of generating a random PBN by providing parameters in command line.

```
//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 6: Example of parameter definition file.

```
./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 7: Output of generating a random PBN by providing parameters in a file.

5.2 Example 2: loading a PBN from Matlab-PBN-toolbox format

For the convenience of the user, we design the function for converting models which have already been defined in the Matlab-PBN-toolbox [4] to the ASSA-PBN format. As described in 4.1, defining a PBN model in ASSA-PBN is not 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 5.1 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 8. A PBN definition file named `exportedFromMatlab.pbn` will be generated and it can be analysed with ASSA-PBN.

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

Figure 8: Exporting a PBN from Matlab-PBN-toolbox.

5.3 Example 3: loading a PBN and exporting it to a file

We will load the file `exportedFromMatlab.pbn` generated in Section 5.2 and export it to a file named `exportedFromASSA.m`. See Figure 9 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 9: Loading a PBN and exporting it to a file.

5.4 Example 4: 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 10 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 5.1 and `property-1.pro` is the property specification file. 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 10: Example of running the Gauss-Seidel method.

5.5 Example 5: 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 11 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 5.4, 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 11: Example of running the perfect simulation approach.

5.6 Example 6: 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 12. 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 12 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 12. 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.txt -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 12: Example of running the two-state Markov chain approach.

5.7 Example 7: 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 13 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.

```
./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 13: Output of the Skart method.

5.8 Example 8: 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 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 14.

```
./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 14: Changing default Java heap size.

5.9 Example 9: performing the two-state Markov chain approach in parallel

Starting from the version 2.0.1, ASSA-PBN supports to analysing a steady-state property of a PBN in a parallel way using multiple CPU cores. Figure 15 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.

```
./assa -tspa 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 15: Checking a single property using the two-state Markov chain approach in a parallel way.

6 Update log of ASSA-PBN

1. 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.

2. 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.

3. 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.

4. 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] A. Mizera, J. Pang, and Q. Yuan. Reviving the two-state markov chain approach (technical report). Available online at `http://arxiv.org/abs/1501.01779`, 2015.

[2] A.E. Raftery and S. Lewis. How many iterations in the Gibbs sampler? *Bayesian Statistics*, 4:763–773, 1992.

[3] A. Tafazzoli, J.R. Wilson, E.K. Lada, and N.M. Steiger. Skart: A skewness- and autoregression-adjusted batch-means procedure for simulation analysis. In *Proc. of the 2008 Winter Simulation Conference*, pages 387–395, 2008.

[4] 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.

[5] 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.

[6] J.-M. Vincent and C. Marchand. On the exact simulation of functionals of stationary Markov chains. *Linear Algebra and its Applications.*, 385:285–310, 2004.