Computational Methods for Analysing Long-run Dynamics of Large Biological Networks

Qixia YUAN

Supervisor:

Prof. Dr. Sjouke Mauw (University of Luxembourg)

Co-supervisor:

Prof. Dr. Thomas Sauter (University of Luxembourg)

Daily advisors:

Dr. Jun Pang (University of Luxembourg)
Dr. Andrzej Mizera (University of Luxembourg)

The author was employed at the University of Luxembourg and supported by the Fonds National de la Recherche Luxembourg (FNR) in the project “New Approaches to Parameter Estimation of Gene Regulatory Networks” (reference 7814267).

UNIVERSITÉ DU LUXEMBOURG

PhD-FSTC-2017-65
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 29/11/2017 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG
EN INFORMATIQUE

by

Qixia YUAN

Born on 28 December 1986 in Rizhao (China)

COMPUTATIONAL METHODS FOR ANALYSING LONG-RUN DYNAMICS OF LARGE BIOLOGICAL NETWORKS

Dissertation defence committee

Dr. Thomas Sauter, Chairman
Professor, Université du Luxembourg

Dr. Jun Pang, Vice-chairman
Université du Luxembourg

Dr. Sjouke Mauw, Dissertation supervisor
Professor, Université du Luxembourg

Dr. Jaco van de Pol
Professor, University of Twente

Dr. Ion Petre
Professor, Turku Centre for Computer Science, Åbo Akademi University

Dr. Andrzej Mizera
Luxembourg Institute of Health

Summary

Systems biology combines developments in the fields of computer science, mathematics, engineering, statistics, and biology to study biological networks from a holistic point of view in order to provide a comprehensive, system level understanding of the underlying system. Recent developments in biological laboratory techniques have led to a slew of increasingly complex and large biological networks. This poses a challenge for formal representation and analysis of those large networks efficiently.

To understand biology at the system level, the focus should be on understanding the structure and dynamics of cellular and organismal function, rather than on the characteristics of isolated parts of a cell or organism. One of the most important focuses is the long-run dynamics of a network, as they often correspond to the functional states, such as proliferation, apoptosis, and differentiation. In this thesis, we concentrate on how to analyse long-run dynamics of biological networks. In particular, we examine situations where the networks in question are very large.

In the literature, quite a few mathematical models, such as ordinary differential equations, Petri nets, and Boolean networks (BNs), have been proposed for representing biological networks. These models provide different levels of details and have different advantages. Since we are interested in large networks and their long-run dynamics, we need to use "coarse-grained" level models that focus on the system behaviour of the network while neglecting molecular details. In particular, we use probabilistic Boolean networks (PBNs) to describe biological networks. By focusing on the wiring of a network, a PBN not only simplifies the representation of the network, but it also captures the important characteristics of the dynamics of the network.

Within the framework of PBNs, the analysis of long-run dynamics of a biological network can be performed with regard to two aspects. The first aspect lies in the identification of the so-called *attractors* of the constituent BNs of a PBN. An attractor of a BN is a set of states, inside which the network will stay forever once it goes in; thus capturing the network's long-term behaviour. A few methods have been discussed for computing attractors in the literature. For example, the binary decision diagram based approach [ZYL$^+$13] and the satisfiability based approach [DT11]. These methods, however, are either restricted by the network size, or can only be applied to synchronous networks where all the elements in the network are updated synchronously at each time step. To overcome these issues, we propose a decomposition-based method. The method works in three steps: we decompose a large network into small sub-networks, detect attractors in sub-networks, and recover the attractors of the original network using the attractors of the sub-networks. Our methods can be applied to both asynchronous networks, where only one element in the network is updated at each time step, and synchronous networks. Experimental results show that our proposed method is significantly faster than the state-of-the-art methods.

The second aspect lies in the computation of steady-state probabilities of a PBN with perturbations. The perturbations of a PBN allow for a random, with a small probability, alteration of the current state of the PBN. In a PBN with perturbations, the long-run dynamics is characterised by the steady-state probability of being in a certain set of states. Various methods for computing steady-state probabilities can be applied to small networks. However, for large networks, the simulation-based statistical methods remain the only viable choice. A crucial issue for such methods is the efficiency. The long-run analysis of large networks requires the computation of steady-state probabilities to be finished as soon as possible. To reach this goal, we apply various techniques. First, we revive an efficient Monte Carlo simulation method called the *two-state Markov chain approach* for making the computations. We identify an initialisation problem, which may lead to biased results of this method, and propose several heuristics to avoid this problem. Secondly, we develop several techniques to speed up the simulation of PBNs. These techniques include the multiple central processing unit based parallelisation, the multiple graphic processing unit based parallelisation, and the structure-based parallelisation. Experimental results show that these techniques can lead to speedups from ten times to several hundreds of times.

Lastly, we have implemented the above mentioned techniques for identification of attractors and the computation of steady-state probabilities in a tool called ASSA-PBN. A case-study for analysing an apoptosis network with this tool is provided.

Acknowledgments

First of all, I would like to thank Prof. Dr. Sjouke Mauw for providing me the with the opportunity to join the SaToSS research group as a PhD student at the University of Luxembourg.

Secondly, I want to thank Prof. Dr. Thomas Sauter for co-supervising me in the field of systems biology.

I thank my daily supervisors Dr. Jun Pang and Dr. Andrzej Mizera for leading me into the world of research and the field of computational systems biology. Without their ongoing support, this work would be undoubtedly impossible.

I thank my collaborator Dr. Hongyang Qu. The successful collaboration with him has lead to several publications and a lot of contributions to this thesis.

I thank Prof. Dr. Ion Petre and Prof. Dr. Jaco van de Pol for being the external reviewer of my thesis and joining my thesis defence committee.

I thank my colleague Zachary Smith for correcting the English grammar mistakes in the thesis.

I thank my friends and colleagues: Xihui Chen, Wei Dou, Olga Gadyatskaya, Haiqin Huang, Ravi Jhawar, Hugo Jonker, Barbara Kordy, Piotr Kordy, Artsiom Kushniarou, Li Li, Karim Lounis, Guozhu Meng, Samir Ouchani, Soumya Paul, Aleksandr Pilgun, Yunior Ramirez Cruz, Rolando Trujillo Rasua, Marco Rocchetto, Cui Su, Jorge Toro Pozo, Chunhui Wang, Jun Wang, Zhe Liu, Yang Zhang, and Lu Zhou for their inspiring and informative discussions.

I would like to thank my parents for their continuous support during the past four years.

Qixia Yuan
December 11, 2017

Contents

List of Figures

List of Tables

1

Introduction

> *"Progress in the study of biological networks such as the heart, brain, and liver will require computer scientists to work closely with life scientists and mathematicians. Computer science will play a key role in shaping the new discipline of systems biology and addressing the significant computational challenges it poses."*
>
> – Anthony Finkelstein et al., *Computational challenges of systems biology*

Systems biology is a scientific field that analyses complex biological networks in a computational and mathematical way. It combines developments in the fields of computer science, mathematics, engineering, statistics, and biology to study biological networks from a holistic point of view. The result of the study is a comprehensive, system-level understanding of the behaviours of the underlying design principles and mechanisms [Gat10, IGH01, Kit02]. Recent developments in biological laboratory techniques have provided a large amount of data on biological networks. Indeed, in the last few years there has been a rapid increase in not only the quantity but also the quality of biological network data [HK09]. MetaCore [Ana17], an integrated software suite for functional analysis of many biological networks, has provided more than 1.6 million molecular interactions and more than 1,600 pathway maps. This rapid increase facilitates more realistic computational modelling of biological networks, but at the same time poses significant challenges with respect to the size of the state space of the resulting computational models that needs to be considered. Hence, to ensure a profound understanding of biological networks, developments of new methods are required to provide means for formal analysis and reasoning about large networks.

In this thesis, we take this challenge and propose a few computational methods for analysing large biological networks. In particular, we are interested in analysing the long-run dynamics of such networks. We explain in details why the long-run dynamics are our focus by introducing a vital concept of *attractors* in Section 1.1. With this focus in mind, we formulate the research objective in terms of two research problems in Section 1.2. We then discuss several mathematical models for describing biological networks and explain the reason why we use *probabilistic Boolean networks* (PBNs) as the modelling framework for describing large biological networks and performing long-run analyses of them. Finally, we provide an overview of this thesis in Section 1.5.

1.1 Attractors in Systems Biology

Originally, the concept of attractors comes from dynamical systems theory, where the whole system is considered to evolve towards a set of preferred states called an attractor.

More formally, an attractor is a set of states inside which the system will stay forever once entered. In biology, this concept dates back to the 1950s when the British developmental biologist Conrad H. Waddington demonstrated his famous "epigenetic landscape" as a conceptual picture of development of cell fate [Wad57]. From the view of Waddington, development takes place like a ball rolling down a sloping landscape that contains multiple "valleys" and "ridges". The valleys describe stable cellular states (cell types) and the ridges act as barriers. Different cell states are maintained by epigenetic barriers that can be overcome by sufficient perturbations. According to C. Waddington, "*this landscape presents, in the form of a visual model, a description of the general properties of a complicated developing system in which the course of events is controlled by many different processes that interact in such a way that they tend to balance each other*." [Wad57]. This suggests that cell types might be reflected by the balanced states of an underlying regulatory network, which is astonishingly similar to the mathematical notion of attractors of dynamical systems [MML09].

The idea of attractors in the context of biological networks has gained a lot of attention and shapes a new direction towards the understanding of these networks. Attractors are originally hypothesised to characterise cellular phenotypes [Kau69a, Kau69b, Kau93]. Later, another complementary conjecture is that attractors correspond to functional cellular states such as *proliferation*, *apoptosis*, or *differentiation* [Hua99, Hua01]. These interpretations can cast new light on the understanding of cellular homeostasis and cancer progression [SDZ02b]. Notably, Shinya Yamanaka, the laureate of the 2012 Nobel Prize in Physiology or Medicine, explains using Waddington's epigenetic landscape the effect that an effective stimulus is able to push a cell from a lineage-committed (stable) state back to a pluripotent (unstable) state. Yamanaka treats the pluripotent states as the ridges, and the lineage-committed states as the valleys which are interpreted as attractors. He shows that a lineage-committed state can be pushed up to a pluripotent state with a competent stimulus, resulting in a higher differentiation potency of the cell [Shi09]. In addition to the usage in understanding biological networks, attractors also play an important role in the development of new drugs. According to [OALH06, Hop08], the number of new drugs reaching the market stage has dramatically decreased since the 1980s. A simple explanation from the viewpoint of attractors could be: the modification to a node located in the internal position of a network can be immediately or quickly counteracted by the feedback relations and therefore, the network cannot be easily modified by pharmacological intervention [TMD+11].

Attractors reflect the long-run dynamics of a biological network; and an understanding of attractors is closely linked with an understanding of the related network. Inspired by this, we concentrate on the analysis of the long-run dynamics of biological processes in this thesis. In particular, we are interested in analysing the long-run dynamics of large computational models, which often arise in the study of biological networks.

1.2 Research Problems

An important way for analysing the long-run dynamics of a biological network is to identify the attractors of this network. For small networks, their attractors can be quickly identified with various methods like enumeration. With the use of techniques like binary decision diagrams (BDDs) and satisfiability (SAT) solver, attractors of medium

networks can also be found efficiently [DT11, ZYL+13]. The BDD-based method usually encodes the corresponding transition relation of a biological network with BDDs and takes advantages of the efficient BDD operations. Although the symbolic encoding of BDDs is efficient, their efficiency is severely hampered when the network becomes huge, e.g., a network with over 10^{30} states. The SAT-based methods transform a biological network into a satisfiability problem. The attractor identification can then be solved by finding a valid assignment of the satisfiability problem. Due to the efficient implementation of SAT solver, they can deal with larger networks within shorter time comparing to BDD-based methods. However, their application is restricted to a special type of networks where the dynamics is deterministic. In addition, a few approximation methods [KBS15, NRTC11] have been proposed to deal with large networks. However, those methods cannot guarantee to identify all the attractors as they are only approximation. This observation leads to the formulation of our first research problem.

Research Problem 1. How to efficiently identify attractors of large biological networks?

The attractors of a network characterise its long-run behaviour. However, if we incorporate random perturbations, the dynamic of the network may evolve out of its attractor. For example, in a gene regulatory network, we can introduce perturbations by allowing the values of each gene to be flipped from an expressed (ON) state to an unexpressed (OFF) state and vice versa with a certain probability. The network can then evolve out of an attractor with certain probability due to the flip of genes. In other words, attractors do not exist in such networks any more. Their long-run behaviour is rather characterised by the probabilities of the network to be in certain states. If the chance for perturbations is very small, then with a high probability, the network will stay in its attractor cycles for a large majority of the time. Therefore, the attractor states can still carry most of the probability mass in the long run. Hence, the probability of being in an attractor in the long run is of vital importance. In the networks whose dynamics can be treated as an ergodic Markov chain (we refer to Chapter 2 for the formal definition of an ergodic Markov chain), such a probability is referred as *steady-state* or long-run probability. Not only are the steady-state probabilities important to reveal the long-run behaviour of a network, but also they constitute the foundation for further in-depth analysis of this network. For instance, in the context of a gene regulatory network, such probabilities can provide answers to questions of the following types: "what is the influence of one gene on another in the long-run?" or "how sensitive is the network with respect to perturbations of a given gene?" Moreover, they could help to formulate new biology hypotheses and, in consequence, to extend and to improve current biological knowledge.

Considering the above reasons, it is easy to conclude that computing the steady-state probabilities is an import task for long-run analysis of a biological network. There are various ways to make such computations. For example, they can be computed via iterative methods like Jacobi [BMW14] or Gauss-Seidel [BMW06]. These iterative methods require the transition matrix of the corresponding Markov chain of a biological network as input. Starting from an arbitrary initial distribution, they update the distribution by multiplying it with the transition matrix in each iteration. When the difference between the new distribution and the previous distribution is smaller than a pre-defined threshold, the iteration finishes and the last calculated distribution is considered as the steady-state distribution of the biological network. This distribution contains the steady-state probabilities for all the states and the steady-state probability for a particular set of states

can be obtained by summing up the probabilities of each state in the set. The Jacobi and Gauss-Seidel methods differ a little in the way of updating the distribution in each iteration. The Jacobi method always uses the old distribution to update all the values in the new distribution while the Gauss-Seidel method makes use of the partially updated distribution to update the remaining values. Since the iterative methods require the transition matrix, which is exponential to the number of elements in a network, such methods only work for smaller networks.

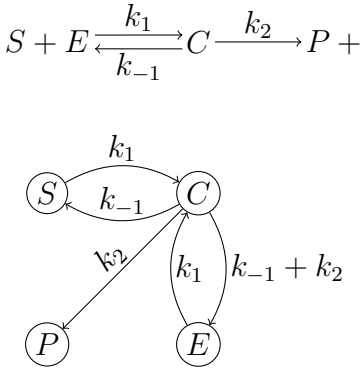
For large networks, estimating the probabilities via simulation-based statistical methods remains the only viable choice. A key issue of these methods is their efficiency, which involves two problems to be considered. The first is to determine a suitable method of estimating the probabilities via sampling finite trajectories; and the second is to make trajectory simulation as fast as possible. A prominent technique explored for the first problem is the method of statistical model checking [YS02, SVA05], which is a simulation-based approach using hypothesis testing to infer whether a stochastic system satisfies a property. Statistical model checking is quite successful in verifying bounded properties, where the estimation is made based on finite executions of the underlying system. The estimation of steady-state probabilities is, however, related to unbounded properties, which is a property reflecting infinite length paths. Approaches like the Skart method [TWLS08] and the perfect simulation algorithm [EP09] have been explored for statistical model checking of "unbounded until" properties. The Skart method has a sound mathematical background and enables estimating confidence intervals with a relatively smaller size of trajectories. However, it requires to store all the simulated states which is memory inefficient. Moreover, the state space of large networks is so huge that the simulated states cannot be directly used by the Skart method for the reason of efficiency. Abstraction of the simulated states is needed. The perfect simulation algorithm uses even less samples to make an estimation. However, it requires the underlying network to be monotonic. The monotonicity assigns order information to the states of a network and a state can only transit to another state which has a higher order. This strict requirement restricted the application of the perfect simulation algorithm in analysing biological networks as a biological network may not be monotonic. Since the efficiency is vital for computing the steady-state probabilities of a biological network, especially in terms of in-depth analysis, methods to handle the above two problems of efficiency are required. Hence, our second research problem focuses on the efficiency of the computation of steady-state probabilities. We formulate it as follows.

Research Problem 2. How to efficiently compute the steady-state probability of being in a set of states of a large biological network?

1.3 Modelling of Biological Networks

To perform a formal analysis for a biological network, the first step is to represent the network in the form of a mathematical/computational model. A number of mathematical/computational frameworks have been proposed for modelling and analysing biological networks. We now briefly review seven popular frameworks. For reviews of other frameworks, we refer to [FH07, BL16].

Ordinary differential equations (ODEs) is a mathematical framework that has been widely applied in modelling and analysing all kinds of biological networks, e.g., in gene



$$\frac{dS}{dt} = -k_1SE + k_{-1}C$$
$$\frac{dE}{dt} = -k_1SE + (k_{-1} + k_2)C$$
$$\frac{dC}{dt} = k_1SE - (k_{-1} + k_2)C$$
$$\frac{dP}{dt} = k_2C$$

Figure 1.1: An example of ODE models demonstrating enzyme catalysed reactions. Left top: Enzyme catalysed reactions where E stands for enzyme, S stands for substrate, C stands for complex, and P stands for product. Left below: The corresponding graph showing the reactions. Right: The corresponding ODEs.

regulatory networks [CQZ12]. ODE models use rate equations to describe the reaction rates of interactions in a biological network. Figure 1.1 shows an example of using ODEs to describe enzyme catalysed reactions. The set of equations can be solved to reflect the concentration of molecular species over time. Using continuous time variables, ODEs can capture time series information of a network and are suitable for quantitative analysis. Another prominent advantage of ODEs is that they have profound mathematical roots which can be used for understanding the underlying networks and analysing their properties such as robustness [FHL$^+$04]. Moreover, there are rich softwares available for ODEs. These include the standard ODE tools like Matlab and Mathematica, as well as the customised ones for biological networks like COPASI [HSG$^+$06], CellDesigner [FMKT03], and CellWare [DMS$^+$04]. Developing an ODE model requires information of kinetic reaction rates, which describes the reactions and numerical values of the kinetic parameters associated with the reactions [dJR06]. Although large amounts of data of network interactions are revealed, the exact reaction rates information is unfortunately rarely available [IM04]. Therefore, modelling with ODEs faces the problem of lacking biological information. In addition, although simple ODEs can be exactly solved mathematically, it becomes too complex for large ODEs. Hence, ODEs are not suitable for large networks [Bor05].

Bayesian networks model a network with two mathematical areas: probability and graph theory [Pea14]. Given $X = \{x_1, x_2, \cdots, x_n\}$, the components (variables) of a network, a Bayesian network models this network as a pair $B = (G, \Theta)$, where G is a directed acyclic graph (DAG) whose nodes represent variables in X and Θ is a set of local conditional probability distributions for each variable in X to qualify the network. A Bayesian network represents a joint probability distribution, which can be decomposed into a product of the local conditional probabilities with the following formula:

$$P\{x_1, x_2, \cdots, x_n\} = \Pi_{i=1}^{n} P\{x_i | Pa(x_i)\},$$

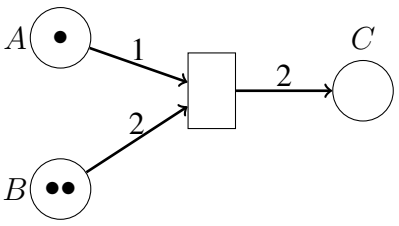
where $Pa(x_i)$ is the values of the parents of x_i. Figure 1.2 (left) shows an example of a Bayesian network with four nodes. Since the graph in a Bayesian network is a DAG, the Bayesian network approaches cannot model cyclic networks. To capture cyclic interactions, Bayesian networks are extended to dynamic Bayesian networks. Essentially, a dynamic Bayesian network represents the joint probability distribution over all possible

Figure 1.2: Left: An example of a Bayesian network consisting of four nodes. The parents of node C are A and B; C is independent of D. Thus, $P\{C|A,B,D\} = P\{C|A,B\}$. Right: An example of a Boolean network consisting of four nodes. An arrow "$\rightarrow$" represents activation while an arrow with a bar ending ("$\dashv$") represents inhibition. Boolean functions are not shown in this figure. This network structure contains a loop ($A \rightarrow C \rightarrow B \rightarrow A$) while the Bayesian network on the left does not contain any loop.

time series of variables in X. It is defined by a pair of Bayesian networks (B_0, B_1). B_0 works as an initial Bayesian network and it defines the joint distribution of the variables in $X(0)$, where $X(t)$ represents the variables in X at time step t and in this case $t = 0$. B_1 is a transition Bayesian network which specifies the transition probabilities $P\{X(t)|X(t-1)\}$ for all t. Dynamic Bayesian networks can capture the time-series data [OGP02] and can be used for inferring genetic regulatory networks from gene expression data [KIM03, ZC04]. However, the applications are limited for small size networks due to the excessive computational cost [VCCW12].

Boolean networks were first introduced by Stuart Kaffman in 1969 as a class of simple models for the analysis of the dynamical properties of gene regulatory networks [Kau69b], where projection of gene states to an ON/OFF pattern of binary states was considered. This model idea fits naturally with gene regulatory networks and signalling networks where each component can represent active and inactive states. The relationship between different components are described with Boolean functions. We show in Figure 1.2 (right) an example BN with four nodes. The Boolean functions in this figure are not shown, but the relationships between different nodes are reflected with arrows. Although BNs are simple, they can provide insights into the dynamics of the modelled biological networks. BNs have also been extended to probabilistic Boolean networks (PBNs) in 2002 by Schmulevich et al. to deal with uncertainty [SD10, TMP+13]. Not only can a PBN incorporate rule-based dependencies between genes and allow the systematic study of global network dynamics, but also it is capable of dealing with uncertainty, which naturally occurs at different levels in the study of biological networks. The limitations of PBNs are that they cannot capture the reaction details and time information. Also, it is difficult to construct a large model from smaller blocks using Boolean models [SHF07].

Petri nets were originally developed to model asynchronous distributed systems in 1962 [PR08]. A Petri net is a graph consisting of place nodes, transition nodes, and edges. Places are usually drawn as circles and represent the resources of the network. Transitions are usually drawn as boxes and represent the events that can change the state of the resources. Each place can be marked by a number of tokens, which represent the states of the place. An edge connecting a place to a transition shows that the transition depends on the state of the place; an edge connecting a transition to a place shows that the outcome of the transition will result in a change of the state of the place. An edge



$$A \stackrel{\text{def}}{=} (\alpha, 1) \downarrow A$$
$$B \stackrel{\text{def}}{=} (\alpha, 2) \downarrow B$$
$$C \stackrel{\text{def}}{=} (\alpha, 2) \uparrow C$$

Figure 1.3: Left: An example of a Petri net describing the reaction $A + 2B \rightarrow 2C$. The dots inside a place node are the marking tokens of that node. Node A is marked with one token; node B is marked with two tokens; node C has no token. Right: An example showing how the reaction $A + 2B \rightarrow 2C$ is described in Bio-PEPA. In this example, α is a label for this reaction. In the first line, $(\alpha, 1) \downarrow A$ means that A participates as a reactant ($\downarrow$) in this reaction with stoichiometry 1. The meanings of the remaining two lines are similar to the first line.

can be labelled with a number to reflect the number of tokens required to "fire" the transition. Petri nets are visual and can be designed and analysed by a range of tools [FH07]. They have been applied to the analysis of metabolic networks [ZOS03, KJH04], gene regulatory networks [SBSW06, KBSK09] and signalling networks [SHK06, LSG$^+$06]. In addition to the standard Petri nets, there are several extended frameworks of Petri nets providing more possibilities for modelling. For example, coloured Petri nets introduce the distinction between tokens to allow the multiple possible values for each place [Jen87]. Another example is the stochastic Petri nets, which add probabilities to the different choices of transitions [BK96]. Similar to Boolean models, Petri nets are also discrete models. The formulation of Petri nets is simple. However, comparing to Boolean models, it is still complex [WMG08].

Process algebras (or process calculi) are a family of formal languages that provide formal specifications of concurrent processes. They have been intensively applied for modelling and analysing biological networks recently [PRSS01, CGH06, DPR08, LMP$^+$14, SNC$^+$17]. They treat the components (e.g., molecules) of a network as "agent" or "process", and describe the interactions between components via reaction channels. Figure 1.3 (right) shows an example for describing a biological reaction with process algebra Bio-PEPA [CH09]. One advantage of process algebras over BNs and Petri nets is their compositionality. This provides means for modelling a network by composing from its sub-components. A notable advantage of process algebras is their close relationship with the technique of model checking, which is a method for formally verifying finite-state concurrent systems. Biological networks described with process algebras can be directly analysed via model checking to verify certain properties, e.g., an analysis of fibroblast growth factor signalling pathway with probabilistic model checking was presented in [HKN$^+$08]. A recent review of applications of process algebras in biology can be found in [GPPQ09]. One drawback of process algebras is that, they are often too abstract and not much intuitive for modelling biological networks as they are not designed to describe biological networks in the beginning. Therefore, additional extensions could be considered for better support of modelling biological networks.

Statecharts was introduced by Harel [Har87] for modelling complex reactive systems. It provides a natural way to model the dynamics of a biological network by specifying the sequence of the states characterizing its behaviours [FH10]. We show in Figure 1.4 (left) an example of a statechart. Similar to process algebras, statecharts have the ad-

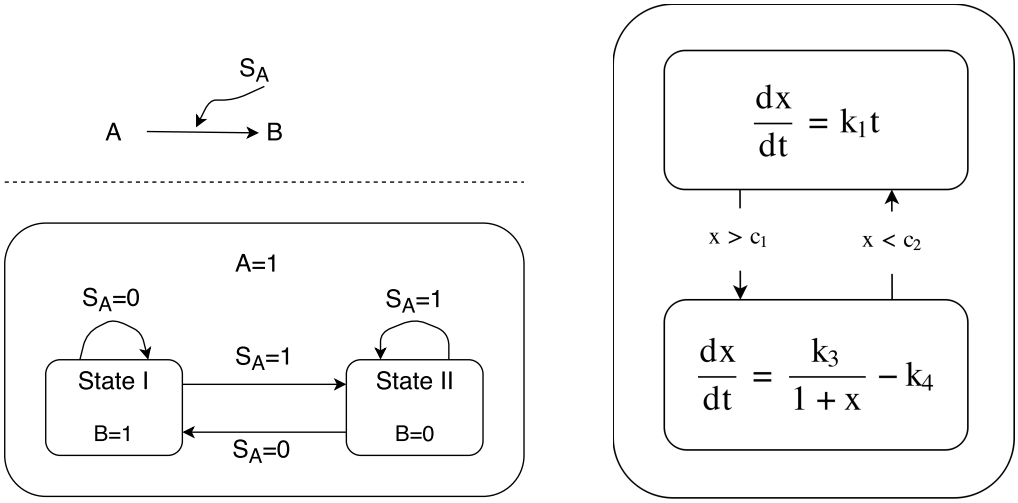


Figure 1.4: Left: An example of a statechart. The graph above the dashed line shows that the presence of A and S_A can work together to control the presence of B. The graph below the dashed line is the corresponding statechart of the graph above. The presence of an element is represented as value 1 and the absence of an element is shown as value 0. This example is modified based on Figure 2 in [SN10]. Right: An example of a hybrid automaton. It describes the changes of the concentration of x. The changes of x is governed by either the equations in the box above or the equations in the box below. The switch of the two boxes is the concentration of x itself.

vantage of compositionality and modularity. They offer a hierarchy structure to handle networks with different levels of detail. Using a hierarchy of states with transitions, events, and conditions, statecharts can describe state changes at a microlevel as well as the state changes at a macro-level, which is a single state change caused by several micro steps but can be described in a more abstract view of the system. Examples of using statecharts to analyse biological networks can be found in [KCH01, CH07]. One major disadvantage of statecharts lies in the fact that a distinct state requires the specification of all possible combination of parameters and this easily leads to an explosion of the number of states [BL16]. Hence, it becomes unrealistic to determine the states and manage the transitions between them when statecharts are used to model large and complex systems [SN10].

Hybrid automata combine discrete and continuous variables into a single dynamical framework [ACHH93]. It is quite natural to model a biological network as a hybrid model since many biological scenarios often involve both discrete and continuous events. For example, the concentration of certain proteins often determines the expression of a gene, which in turns affect the dynamical change of the concentration of proteins. An example demonstrating this process is shown in Figure 1.4 (right). Due to this appealing feature, hybrid automata have been applied a lot in analysing biological networks, e.g., [GTT03, DFTdJV06, BCB$^+$16]. The continuous part of a hybrid automaton is often described with differential equations. As a result, exact quantitative data are required in order to adjust the equations. The drawbacks of ODE models therefore also exist for hybrid automata.

Each of these different frameworks has corresponding advantages and disadvantages. Perhaps there is no perfect framework which bypasses others in all aspects. Selecting a framework should be made in accordance with the actual usage. Our selection of frame-

works follows two simple rules, i.e., suitable for long-run analysis and able to handle large networks. Following the two rules, we first exclude fine-grained models which attempt to model the precise details of the underlying network. Fine-grained models like ODEs are able to reflect detailed information on a biological network; however their applicability is severely hampered due to a number of reasons when it comes to the modelling of large networks. For example, experimental data for large genetic systems are often incomplete and hence it is not possible to provide the whole set of kinetic-like weights for quantifying the relations between different elements. In addition, the standard differential equations model for a single elementary block of the network (e.g., a gene) becomes prohibitively complex when applied to the whole network. Therefore, utilising coarse-grained models, which focus on the wiring information of the underlying networks, becomes the only feasible solution for large biological networks. In fact, these coarse-grained formalisms have been proved to posses a lof of predictive power in many systems biology studies [AO03], especially in the cases where the exact reaction rates are not the main focus. For instance, the study in [LLL+04] shows that dynamical attractors of the genetic network that controls the yeast cell cycle seem to depend on the circuit wiring rather than the details of the kinetic constants. In this sense modelling biological systems with more abstract formalisms that use a more high-level view has certain unquestionable advantages. Among the coarse-grained models, Boolean models are one of the "simplest" types of models as each element in such a model can have only one of two states, i.e. ON or OFF, also referred to as 1 or 0 in a computational model, respectively [HK09]. Other models like Petri nets are more flexible than Boolean models; however, since our focus is long-run dynamics analysis, the simplest Boolean models are already "complex" enough to provide insights into the long-run dynamics of biological networks [Bor05]. Hence, in this thesis, we focus on the framework of Boolean models.

1.4 Addressing Research Problems with Boolean Models

After selecting the modelling framework, we now discuss how to handle the two research problems under the framework of Boolean models.

1.4.1 Attractor Detection in Large Boolean Models

Although BNs and PBNs are all Boolean models, they should be distinguished in terms of attractor detection. The definition of attractors described in Section 1.1 can be directly applied to the framework of BNs. The attractors in PBNs, however, need to be redefined due to the special dynamics of PBNs. Two types of PBNs are introduced in the literature and we discuss the attractors for the two types separately. The first type comes from the original definition of a PBN, which is known as an *instantaneously random* PBN. In an instantaneously random PBN, a node may be associated with a set of functions and at any time point one of the functions in the set is selected to determine the value of the node. The other type, referred as a *context-sensitive* PBN, is used to capture the uncertainty of latent factors outside a network model, whose behaviours influence regulation within the network. A context-sensitive PBN consists of a list of constituent BNs, each known as a context, and switches between these contexts in a stochastic way. At each time step, a context-sensitive PBN can either remain in a context, or switch to a new context

with a probability and evolve accordingly. When a context-sensitive PBN is in a certain context, i.e., a BN, it will eventually settle into one of the attractors of this context. Since a context is usually kept for a period of time, the chances for a context-sensitive PBN to evolve in an attractor of a context is still high in the long-run. Therefore, it is meaningful to identify the attractors of the constituent BNs of a context-sensitive PBN. Hence, in this thesis, the attractor detection in large Boolean models is performed in a BN, and in each of the constituent BNs of a context-sensitive PBN.

In general, there are two updating schemes for Boolean models: synchronous and asynchronous. In BNs with the synchronous updating scheme, the values of all the nodes are updated simultaneously at each time step; while in BNs with the asynchronous updating scheme, the value of one randomly selected node is updated at each time step. The two updating schemes correspond to different biological scenarios; hence both of them should be handled. Since the two schemes pose different properties to a BN, we treat the attractor detection in synchronous BNs and asynchronous BNs separately to gain an optimal detection speed. As mentioned in Section 1.2, attractor detection in smaller size networks can be easily handled with various methods. The challenge lies in large networks where existing methods are hampered by the state space explosion problem. Our solution for this challenge is to use the idea of "divide-and-conquer". Based on this idea, we design two decomposition methods: one for synchronous BNs and one for asynchronous BNs. The two methods follow the same pattern for detecting attractors: given a large BN, we divide it into several small sub-networks, detect attractors in sub-networks and recover the attractors in the original network. However, they make use of the different properties provided by the synchronous or asynchronous updating scheme to reach an optimal detection speed for the two types of BNs.

1.4.2 Steady-state Probabilities Computation in Large Boolean Models.

From a mathematical point of view, the steady-state probabilities are only meaningful in a network whose dynamics can be treated as an ergodic Markov chain. If a BN or a PBN is incorporated with perturbations, it can evolve to any state from an arbitrary state in the network. Therefore, the dynamics of such a network can be treated as an ergodic Markov chain. We refer to Section 2.3 for a formal definition of BNs/PBNs with perturbations and for their relationships with ergodic Markov chains. Due to this mathematical view, we consider BNs or PBNs with perturbations when computing the steady-state probabilities. For simplification, we will only mention PBNs in the rest of this thesis when we discuss steady-state probabilities computation as BNs are special cases of PBNs.

As discussed in Section 1.2, two issues need to be considered for efficiently computing the steady-state probabilities in large Boolean models: one is to determine a suitable method of estimating the probabilities via sampling finite trajectories, and the other is to make the simulation as fast as possible. To address the first issue, we explore a method called the two-state Markov chain approach [RL92]. This method has been proposed for computing the steady-state probabilities in [SD10]. Its idea of abstracting a huge state space into two meta states is exactly suitable for the purpose of steady-state probabilities computation in large PBNs. However, there is an initialisation problem which may lead to biased results of this method. Probably due to lack of statistical experiments, this problem was not observed before. We identify the initialisation problem and pro-

pose several heuristics to avoid this problem. Moreover, we have made a comparative study for comparing the efficiency of the two-state Markov chain approach with another state-of-the-art method, i.e., the Skart method, in terms of computing steady-state probabilities of a large PBN. We manage to handle the large memory requirement for the Skart method and successfully apply it in computing the steady-state probabilities of a large PBN. Our experiments show that the two-state Markov chain approach is better than the Skart method in terms of efficiency.

We address the second issue with various techniques. Firstly, we apply the technique called the Alias method [Wal77] to maximize the speed for selecting a context. The alias method allows to make a selection of the context of a PBN within constant time, irrespective of the number of contexts. Secondly, we consider parallel simulation techniques with multiple cores. The current hardware techniques provide possibilities for making calculations with multiple central processing units (CPUs) as well as multiple graphical processing units (GPUs). To make use of these techniques in the computation of steady-state probabilities of a PBN, we design a method combining the two-state Markov chain approach and the Gelman & Rubin method [GR92]. This combination allows us to use samples obtained from different cores to calculate steady-state probabilities. Thirdly, we focus on the structure of a PBN and make use of this to speedup the simulation process. The developments in computer hardware provide not only more CPU cores and GPU cores, but also more memory. This gives us the possibility to use large memory for speeding up computations. By analysing the structure of a PBN, we group and merge nodes, and restore them in different data structure. This process requires additional memory usage, but can lead to faster simulation speed.

In addition to addressing the two research problems with theoretical algorithms and methods, it is also vital to make them applicable. Hence, we design a software tool called ASSA-PBN for modelling, simulating, and analysing PBNs. We integrate all the techniques mentioned above in the tool. Moreover, we further implement several in-depth analysis functions, e.g., parameter estimation of PBNs. We give an overview of this thesis in the next section to explain how the research problems are addressed in each chapter.

1.5 Thesis Overview

This thesis is composed of three main parts. Part I concentrates on the first research problem and discusses computational techniques of attractor identification in both asynchronous (Chapter 3) and synchronous networks (Chapter 4). Part II focuses on the second research problem and discusses in Chapters 5, 6, and 7 several methods for efficient steady-state probabilities computation. The last part presents in Chapter 8 a software tool called ASSA-PBN which includes the techniques and algorithms discussed in the first two parts for performing long-run analyses of PBNs. A detailed description of each chapter is given below.

- **Chapter 2. Preliminaries**
  In this chapter, we introduce fundamental concepts used throughout this thesis, e.g., Boolean networks and probabilistic Boolean networks.
- **Chapter 3. Attractor Detection in Asynchronous Networks**
  In this chapter, we discuss the problem of attractor detection in asynchronous

networks, where only one element in the network is updated at each time step. The asynchronous network can either be an asynchronous BN or an asynchronous *context-sensitive* PBN. We will introduce the concept of context-sensitive PBNs later in Chapter 2.
This chapter is based on the work [MPQY18], which has been accepted in the 16th Asia Pacific Bioinformatics Conference (APBC'18).

- **Chapter 4: Attractor Detection in Synchronous Networks**
  In this chapter, we focus on methods for attractor detection in synchronous networks, where the values of all the elements are updated synchronously. We propose a strongly connected component based decomposition method for attractor detection and we prove its correctness.
  This chapter is based on the work [YQPM16, MQPY17], which were respectively published in the journal of Science China Information Science and in the proceedings of the 3rd International Symposium on Dependable Software Engineering: The Theories, Tools, and Applications (SETTA'17).
- **Chapter 5. Efficient Steady-state Computation**
  Starting from this chapter, we deal with the second research problem. We discuss several different methods for computing steady-state probabilities of a biological network. Specifically, we focus on a method called the two-state Markov chain approach [RL92]. We discuss situations that may lead to biased results obtained with this method and we propose a remedy for it.
  This chapter is based on the work [MPY17], which is published in the journal of IEEE/ACM Transactions on Computational Biology and Bioinformatics.
- **Chapter 6. Multiple-core Based Parallel Steady-state Computation**
  In this chapter, we discuss a multiple-core based parallel technique for speeding up the steady-state probabilities computation. We propose to combine the two-state Markov chain approach with the Gelman & Rubin method [GR92] for this purpose. By doing this combination, we are able to use samples from different trajectories to make the computation. Therefore, we can use multiple cores, which can be either CPU cores or GPU cores. When it comes to GPU cores, we need to take special care of the GPU memory usage. Special data structures and data optimization methods are provided for fast operations in GPU cores.
  This chapter is based on the work [MPY16d, MPY16c], which were respectively published in the proceedings of the 31st ACM Symposium on Applied Computing (SAC'16), and in the proceedings of the 2nd International Symposium on Dependable Software Engineering: The Theories, Tools, and Applications (SETTA'16).
- **Chapter 7. Structure-based Parallel Steady-state Computation**
  In this chapter, we discuss a technique called structure-based parallelisation for speeding up the steady-state probabilities computation. Instead of using more cores, this technique uses more memory and applies the idea of gaining speed by sacrificing memory.
  This chapter is based on the work [MPY16b], which was published in the proceedings of the 14th International Conference on Computational Methods in Systems Biology (CMSB'16).
- **Chapter 8. ASSA-PBN: a software tool for probabilistic Boolean networks.**
  In this chapter, we introduce the tool ASSA-PBN which is designed for approximate steady-state analysis of probabilistic Boolean networks. The tool implements the above discussed methods and techniques and is able to perform both

attractor detection and steady-state probabilities computation. Moreover, the tool provides several in-depth network analysis functions. As a case-study, we demonstrate how parameter estimation can be performed using our tool ASSA-PBN.
This chapter is based on two published work and one under-review work submitted to the journal of Transactions on Computational Biology and Bioinformatics. The two published work are [MPY15, MPY16a], which were respectively published in the proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA'15), and in the proceedings of the 14th International Conference on Computational Methods in Systems Biology (CMSB'16).

2

Preliminaries

To fully understand this thesis, some preliminary knowledge is required. In this chapter, we give a brief introduction of it. We first describe the finite discrete-time Markov chains (DTMCs). Then, we present Boolean networks (BNs) and probabilistic Boolean networks (PBNs).

2.1 Finite discrete-time Markov chains (DTMCs)

Let S be a finite set of states. A (first-order) DTMC is an S-valued stochastic process $\{X_t\}_{t\in\mathbb{N}}$ with the property that the next state is independent of the past states given the present state. Formally, $\mathbb{P}(X_{t+1} = s_{t+1} \,|\, X_t = s_t, X_{t-1} = s_{t-1}, \ldots, X_0 = s_0) = \mathbb{P}(X_{t+1} = s_{t+1} \,|\, X_t = s_t)$ for all $s_{t+1}, s_t, \ldots, s_0 \in S$. Here, we consider *time-homogenous* Markov chains, i.e., chains where $\mathbb{P}(X_{t+1} = s' \,|\, X_t = s)$, denoted $P_{s,s'}$, is independent of t for any states $s, s' \in S$. The transition matrix $P = (P_{s,s'})_{s,s'\in S}$ satisfies $P_{s,s'} \geqslant 0$ and $\sum_{s'\in S} P_{s,s'} = 1$ for all $s \in S$. Formally, the definition of a DTMC is given below.

Definition 2.1.1 (Discrete-time Markov chain)**.** *A Discrete-time Markov chain $\mathcal{D}$ is a 3-tuple $\langle S, S_0, P\rangle$ where S is a finite set of states, $S_0 \subseteq S$ is the initial set of states, and $P : S \times S \to [0, 1]$ is the transition probability matrix where $\Sigma_{s'\in S} P(s, s') = 1$ for all $s \in S$. When $S = S_0$, we write $\langle S, P\rangle$.*

We denote by π a probability distribution on S. If $\pi = \pi P$, then π is a *stationary distribution* of the DTMC (also referred to as a *invariant distribution*). A path of length n is a sequence $s_1 \to s_2 \to \cdots \to s_n$ such that $P_{s_i,s_{i+1}} > 0$ and $s_i \in S$ for $i \in \{1, 2, \ldots, n\}$. State $q \in S$ is *reachable* from state $p \in S$ if there exists a path such that $s_1 = p$ and $s_n = q$. A DTMC is *irreducible* if any two states are reachable from each other. The period of a state is defined as the greatest common divisor of the lengths of all paths that start and end in the state. A DTMC is *aperiodic* if all states in S are of period 1. A finite state DTMC is called *ergodic* if it is irreducible and aperiodic. By the famous ergodic theorem for DTMCs [Nor98], an ergodic chain has a unique stationary distribution being its *limiting distribution* (also referred to as the *steady-state distribution*) given by $\lim_{n\to\infty} \pi_0 P^n$, where π_0 is any initial probability distribution on S. In consequence, the limiting distribution for an ergodic chain is independent of the choice of π_0. The steady-state distribution can be estimated from any initial distribution by iteratively multiplying it by P.

The evolution of a first-order DTMC can be described by a stochastic recurrence sequence $X_{t+1} = \phi(X_t, U_{t+1})$, where $\{U_t\}_{t\in\mathbb{N}}$ is an independent sequence of uniformly distributed real random variables over $[0, 1]$ and the transition function $\phi : S\times[0, 1] \to S$

satisfies the property that $\mathbb{P}(\phi(s,U) = s') = P_{s,s'}$ for any states $s, s' \in S$ and for any U, a real random variable uniformly distributed over $[0,1]$. When S is partially ordered and the transition function $\phi(\cdot, u)$ is monotonic, then the chain is said to be *monotone* [PW96].

If we ignore the transition probability matrix in a DTMC and concentrate on the transition relation between states, we have the concept of *state transition system* as follows:

Definition 2.1.2 (State transition system)**.** *A state transition system $\mathcal{T}$ is a 3-tuple $\langle S, S_0, T\rangle$ where S is a finite set of states, $S_0 \subseteq S$ is the initial set of states, and $T \subseteq S \times S$ is the transition relation. When $S = S_0$, we write $\langle S, T\rangle$.*

In the state transition system, we define path and reachability as follows.

Definition 2.1.3 (Path and reachability)**.** *In a state transition system $\mathcal{T} = \langle S, S_0, T\rangle$, a path of length k ($k \geq 2$) is a serial $s_1 \to s_2 \to \cdots \to s_k$ of states in S such that there exists a transition between any consecutive two states x_i and x_{i+1}, where $i \in [1, k-1]$. A state s_j is reachable from s_i if there is a path from s_i to s_j.*

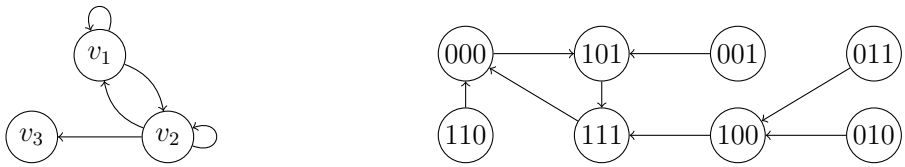
2.2 Boolean Networks

A Boolean network (BN) is composed of two elements: binary-valued nodes, which represent elements of a biological system, and Boolean functions, which represent interactions between the elements. The concept of BNs was first introduced in 1969 by S. Kauffman for analysing the dynamical properties of GRNs [Kau69a], where each gene was assumed to be in only one of two possible states: ON/OFF.

Definition 2.2.1 (Boolean network)**.** *A* Boolean network $G(V, \boldsymbol{f})$ *consists of a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$, also referred to as genes, and a vector of Boolean functions $\boldsymbol{f} = (f_1, f_2, \ldots, f_n)$, where f_i is a predictor function associated with node v_i ($i = 1, 2, \ldots, n$). For each node v_i, its predictor function f_i is defined with respect to a subset of nodes $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{k(i)}}\}$, referred to as the set of* parent nodes *of v_i, where $k(i)$ is the number of parent nodes and $1 \leq i_1 < i_2 < \cdots < i_{k(i)} \leq n$. A state of the network is given by a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, where $x_i \in \{0,1\}$ is a value assigned to node v_i.*

Since the nodes are binary, the state space of a BN is exponential in the number of nodes. Starting from an initial state, the BN evolves in time by transiting from one state to another. The state of the network at a discrete time point t ($t = 0, 1, 2, \ldots$) is given by a vector $\boldsymbol{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$, where $x_i(t)$ is a binary-valued variable that determines the value of node v_i at time point t. The value of node v_i at time point $t+1$ is given by the predictor function f_i applied to the values of the parent nodes of v_i at time t, i.e., $x_i(t+1) = f_i(x_{i_1}(t), x_{i_2}(t), \ldots, x_{i_{k(i)}}(t))$. For simplicity, with slight abuse of notation, we use $f_i(x_{i_1}, x_{i_2}, \ldots, x_{i_{k(i)}})$ to denote the value of node v_i at the next time step. For any $j \in [1, k(i)]$, node v_{i_j} is called a *parent node* of v_i and v_i is called a *child node* of v_{i_j}.

In general, the Boolean predictor functions can be formed by combinations of any logical operators, e.g., logical AND $\wedge$, OR $\vee$, and NEGATION $\neg$, applied to variables associated with the respective parent nodes. The BNs are divided into two types based on the time evolution of their states, i.e., *synchronous* and *asynchronous*.



(a) A BN with 3 nodes. (b) Synchronous transition graph of the BN in Example 2.2.1.

Figure 2.1: The Boolean network in Example 2.2.1 and its state transition graph.

- **Synchronous BNs.**
  In synchronous BNs, values of all the variables are updated simultaneously. The transition relation of a synchronous BN is given by

$$T\left(\boldsymbol{x}(t), \boldsymbol{x}(t+1)\right) \;=\; \bigwedge_{i=1}^{n} \Big(x_i(t+1) \leftrightarrow f_i(x_{i_1}(t), x_{i_2}(t), \cdots, x_{i_{k_i}}(t))\Big). \quad (2.1)$$

  It states that in every step, all the nodes are updated synchronously according to their Boolean functions.
- **Asynchronous BNs.**
  In asynchronous BNs, one variable at a time is randomly selected for update. The transition relation of an asynchronous BN is given by

$$T\left(\boldsymbol{x}(t), \boldsymbol{x}(t+1)\right) = \exists i \left(\Big(x_i(t+1) \leftrightarrow f_i(x_{i_1}(t), x_{i_2}(t), \cdots, x_{i_{k_i}}(t))\Big) \bigwedge_{j=1, j\neq i}^{n} (x_j(t+1) \leftrightarrow x_j(t)) \right). \quad (2.2)$$

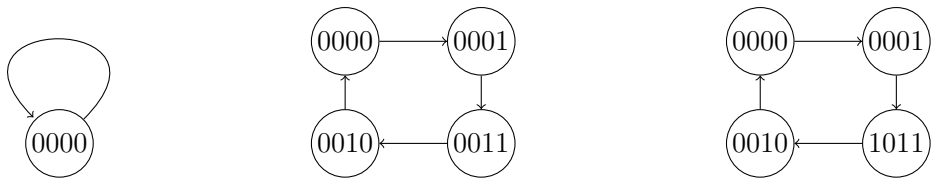
  It states that node v_i is updated by its Boolean function and other nodes are kept unchanged. Each node has a chance to be updated by its Boolean function, therefore there are n outgoing transitions in maximum from any state.

In many cases, a BN $G(V, \boldsymbol{f})$ is studied as a state transition system. A BN can be easily modelled as a state transition system: the set S is just the state space of the BN, so there are 2^n states for a BN with n nodes; the initial set of states S_0 is the same as S; the transition relation T is given by either Equation 2.1 (when the BN is synchronous) or Equation 2.2 (when the BN is asynchronous); finally the label function L can be just a function mapping s to its value.

Example 2.2.1. *A synchronous BN with 3 nodes is shown in Figure 2.1a. Its Boolean functions are given as: $f_1 = \neg(x_1 \wedge x_2)$, $f_2 = x_1 \wedge \neg x_2$, and $f_3 = \neg x_2$. In Figure 2.1a, the three circles v_1, v_2, and v_3 represent the three nodes of the BN. The edges between nodes represent the interactions between nodes. Applying the transition relation to each of the states, we can get the corresponding state transition system. For better understanding, we demonstrate the state transition system as a state transition graph in this thesis. The corresponding state transition graph of this example is shown in Figure 2.1b.*

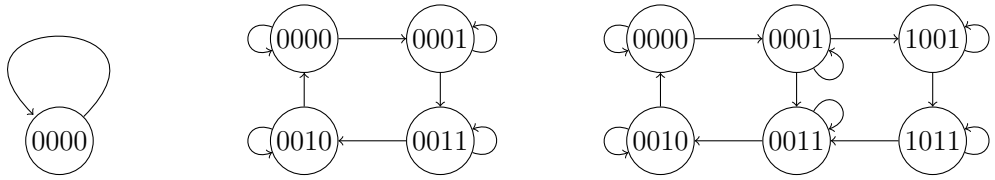
In the transition graph of Figure 2.1b, the three states $(000), (1*1)$[1] can be reached from each other but no other state can be reached from any of them. This forms an *attractor* of the BN. The formal definition of an *attractor* is given as follows.

[1] We use $*$ to denote that the bit can have value either 1 or 0, so $(1*1)$ actually denotes two states: 101 and 111.



(a) A selfloop. (b) A simple loop (type II). (c) A simple loop (type III).

Figure 2.2: Three types of attractor systems in a synchronous BN.



(a) A selfloop. (b) A simple loop. (c) A complex loop.

Figure 2.3: Three types of attractor systems in an asynchronous BN.

Definition 2.2.2 (Attractor of a BN)**.** *An* attractor *of a BN is a set of states satisfying that any state in this set can be reached from any other state in this set and no state in this set can reach any other state that is not in this set.*

Example 2.2.2. *The BN given in Example 2.2.1 has one attractor, i.e.,* $\{(000), (1 * 1)\}$.

When analysing an attractor, we often need to identify transition relations between the attractor states. We call an attractor together with its state transition relation as an *attractor system* (AS). The states constituting an attractor are called *attractor states*. The attractors of a BN characterise its long-run behaviour [SD10] and are of particular interest due to their biological interpretation.

For synchronous BNs, each state of the network can only have at most one outgoing transition. Therefore, the transition graph of an attractor in a synchronous BN is simply a loop. By detecting all the loops in a synchronous BN, one can identify all its attractors. The attractor systems can be divided into three different types. Type I: selfloops. An attractor composed of only one state as shown in Figure 2.2a. Type II: simple loops where the Hamming distance between two consecutive states is 1 as shown in Figure 2.2b. Type III: simple loops where the maximum Hamming distance between two consecutive states is greater than 1 as shown in Figure 2.2c.

For asynchronous BNs, a state may have multiple outgoing transitions. Therefore, an attractor may not be a loop any more. The attractor systems in an asynchronous BN can also be divided into three types. Type I: selfloops. This type of attractors is the same as in a synchronous network. An example is shown in Figure 2.3a. Type II: simple loops. This type of attractors is quite similar to the type II attractor in a synchronous network. The maximum Hamming distance between two consecutive states is also 1 but there are selfloops due to the asynchronous update mode. See Figure 2.3b for an example. Type III: complex loops. This type of attractors only exists in asynchronous networks since a state may have more than one outgoing transitions leading to two or more different states. Figure 2.3c shows an example of this kind of loop.

A type I attractor or a type II attractor in a synchronous BN will be present as type

I attractor and type II attractor in its corresponding asynchronous BN and vice verse. However, a type III attractor in a synchronous BN is not necessarily present in its corresponding asynchronous BN and vice verse.

2.3 Probabilistic Boolean Networks (PBNs)

PBNs were introduced to reflect the indeterministic of a biological system [SDZ02b, SDKZ02, SDZ02a], originally as a model for gene regulatory networks. It allows a node to have more than one Boolean function and the value of a node is updated based on a selected Boolean function each time. Formally, the definition of PBNs is given as:

Definition 2.3.1 (Probabilistic Boolean network)**.** *A PBN $G(V, \mathcal{F})$ consists of a set of binary-valued nodes $V = \{v_1, v_2, \ldots, v_n\}$ and a list of sets $\mathcal{F} = (F_1, F_2, \ldots, F_n)$. For each $i \in \{1, 2, \ldots, n\}$ the set $F_i = \{f_1^{(i)}, f_2^{(i)}, \ldots, f_{l(i)}^{(i)}\}$ is a collection of predictor functions for node v_i, where $l(i)$ is the number of predictor functions for v_i. Each $f_j^{(i)} \in F_i$ is a Boolean function defined with respect to a subset of nodes referred to as parent nodes of $f_j^{(i)}$. The union of all the parent nodes of $f_j^{(i)} \in F_i$ is the parent nodes of v_i.*

The above definition does not include probability distributions. Two different probability distributions will be introduced later on for two different types of PBNs. We denote by $x_i(t)$ the value of node v_i at time point $t \in \mathbb{N}$. The state space of the PBN is $S = \{0, 1\}^n$ and it is of size 2^n. The state of the PBN at time t is determined by $\boldsymbol{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$. The dynamics of the PBN is given by the sequence $(\boldsymbol{x}(t))_{t=0}^{\infty}$. Each node x_i has $l(i)$ possible predictor functions. A *realisation* of a PBN at a given time is a function vector where the ith function of the vector is selected from the predictor functions of node x_i. For a PBN with N realisations, there are N possible network transition functions $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_N$ of the form $\boldsymbol{f}_k = (f_{k_1}^{(1)}, f_{k_2}^{(2)}, \ldots, f_{k_n}^{(n)}), k = 1, 2, \ldots, N, 1 \leqslant k_j \leqslant l(j), f_{k_j}^{(j)} \in F_j$, and $j = 1, 2, \ldots, n$. Each network function $\boldsymbol{f}_k$ defines a *constituent* Boolean network, or a *context*, of the PBN.

At each time point of the PBN's evolution, a decision is made whether to switch the constituent network. This is modelled with a binary random variable ξ: if $\xi = 0$, then the current constituent network is preserved; if $\xi = 1$, then a context is randomly selected from all the constituent networks in accordance with the probability distribution of $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_N$. Notice that this definition implies that there are two mutually exclusive ways in which the context may remain unchanged: 1) either $\xi = 0$ or 2) $\xi = 1$ and the current network is reselected. The functional switching probability $q = Pr(\xi = 1)$ is a system parameter. Two cases are distinguished in the literature: if $q = 1$, then a switch is made at each updating epoch; if $q < 1$, then the PBN's evolution in consecutive time points proceeds in accordance with a given constituent BN until the random variable ξ calls for a switch. If $q = 1$, as originally introduced in [SDKZ02], the PBN is said to be *instantaneously random*; if $q < 1$, it is said to be *context-sensitive*. In this thesis, when we consider an instantaneously random PBN, we restrict it to be an *independent* PBN where predictor functions for different nodes are selected independently of each other; while when we consider a context-sensitive PBN, the PBN is dependent by definition.

In instantaneously random PBNs, there is a probability distribution $C^{(i)} = (c_1^{(i)}, c_2^{(i)}, \ldots, c_{l(i)}^{(i)})$ on each $F_i \in \mathcal{F}$, where $c_j^{(i)}$ for $j \in [1, l(i)]$ is the probability of selecting $f_j^{(i)} \in F_i$

as the next predictor for v_i and it holds that $\sum_{j=1}^{l(i)} c_j^{(i)} = 1$. A realisation selected at time t is referred to as $\boldsymbol{F}_t$. Due to independence, the probability distribution on constituent BNs is given by $\mathbb{P}(\boldsymbol{f}_k) = \mathbb{P}(\boldsymbol{F}_t = \boldsymbol{f}_k) = \prod_{i=1}^{n} c_{k_i}^{(i)}$. In this way, the instantaneously random PBN can be viewed as a time-homogeneous DTMC: the state space of the DTMC is S, i.e., the state-space of the PBN; the transition probability between two states $\boldsymbol{x}$ and $\boldsymbol{x}'$ is given by $P_{\boldsymbol{x},\boldsymbol{x}'} = \sum_{k=1}^{N} \mathbb{1}_{[\boldsymbol{f}_k(\boldsymbol{x})=\boldsymbol{x}']} \mathbb{P}(\boldsymbol{f}_k)$, where $\mathbb{1}$ is the indicator function.

In the context-sensitive PBNs, there is no probability distribution on each $F_i \in \mathcal{F}$. Instead, there is a probability distribution $D = (d_1, d_2, \ldots, d_N)$ on constituent BNs, where d_k for $k \in [1, N]$ is the probability that context $\boldsymbol{f}_k$ is selected when a switch of context is made ($\xi = 1$). Let $\boldsymbol{F}_t$ be the context at time point t, then the probability that context $\boldsymbol{f}_k$ is selected at time point $t+1$ is that $\mathbb{P}(\boldsymbol{F}_t = \boldsymbol{f}_k) = (1-q)\mathbb{1}_{[\boldsymbol{F}(t+1)=\boldsymbol{F}(t)]} + qd_k$. Therefore, we cannot view the states of a context-sensitive PBN as a time-homogeneous DTMC. However, if we combine the states of a context-sensitive PBN (represented by $\boldsymbol{x}$) with the contexts $\boldsymbol{f}$ in the PBN to form new states, we can view the new states represented by $(\boldsymbol{x}, \boldsymbol{f})$ as a time-homogeneous DTMC.

Similarly to Boolean networks, there are also two updating schemes for PBNs: synchronous and asynchronous. Therefore, PBNs can be divided into the following four types.

- **Instantaneously Random Synchronous PBNs.**

  In the instantaneously random synchronous PBNs, the transition from $\boldsymbol{x}(t)$ to $\boldsymbol{x}(t+1)$ is conducted by randomly selecting a predictor function for each node v_i from F_i and by synchronously updating the node values in accordance with the selected functions. A realisation selected at time t is referred to as $\boldsymbol{F}_t$. The transition relation of a instantaneously random synchronous PBN can then be denoted as

  $$T(\boldsymbol{x}(t), \boldsymbol{x}(t+1)) = \boldsymbol{F}_t(\boldsymbol{x}(t)). \tag{2.3}$$

- **Instantaneously Random Asynchronous PBNs.**
  In the instantaneously random asynchronous PBNs, the transition from $\boldsymbol{x}(t)$ to $\boldsymbol{x}(t+1)$ is conducted by randomly selecting a node v_i, randomly selecting a predictor function for the node v_i from F_i and updating the node value in accordance with the selected function. Let $f_i(t)$ be the randomly selected function of the randomly selected node v_i at time point t and $s_i(t)$ be the parent nodes of function $f_i(t)$. The transition relation of an instantaneously random asynchronous PBN can then be denoted as $T(\boldsymbol{x}(t), \boldsymbol{x}(t+1)) =$

  $$(x_1(t), x_2(t), \ldots, x_{i-1}(t), f_i(t)(s_i(t)), x_{i+1}(t), \ldots, x_n(t)). \tag{2.4}$$

- **Context-sensitive Synchronous PBNs.**
  In the context-sensitive synchronous PBNs, the transition from $\boldsymbol{x}(t)$ to $\boldsymbol{x}(t+1)$ is conducted by synchronously updating the node values in accordance with Boolean functions of the current constituent BN which is either the same as its previous context (with probability $1-q$) or randomly selected from all the constituent BNs (with probability q). Similarly to the instantaneously random synchronous PBNs, we refer the context of Boolean functions governing the update of nodes values at time t as $\boldsymbol{F}_t$. Then the transition relation of a context-sensitive synchronous PBN can be denoted using Equation 2.3 as well.

- **Context-sensitive Asynchronous PBNs.**
  In the context-sensitive asynchronous PBNs, the transition from $\boldsymbol{x}(t)$ to $\boldsymbol{x}(t+1)$ is conducted by randomly selecting a node v_i, and by updating the node value in accordance with its Boolean function of the current constituent BN which is either the same as its previous context (with probability $1-q$) or randomly selected from all the constituent BNs (with probability q). Let $f_i(t)$ be the Boolean function of the randomly selected node v_i in the current constituent BN at time point t and $s_i(t)$ be the parent nodes of function $f_i(t)$. The transition relation of a context-sensitive asynchronous PBN can then be denoted using Equation 2.4 as well.

In a PBN with *perturbations*, a perturbation rate $p \in (0,1)$ is introduced and the dynamics of a PBN is guided with either perturbations or predictor functions: at each time point t, the value of each node v_i is flipped with probability p; and if no flip happens, either the value of each node v_i is updated with selected predictor functions synchronously in the synchronous update mode or the value of a randomly selected node is updated with the selected predictor function in the asynchronous update mode. Let $\boldsymbol{\gamma}(t) = (\gamma_1(t), \gamma_2(t), \ldots, \gamma_n(t))$ be a perturbation vector, where each element is a Bernoulli distributed random variable with parameter p, i.e., $\gamma_i(t) \in \{0,1\}$ and $\mathbb{P}(\gamma_i(t)=1) = p$ for all t and $i \in \{1,2,\ldots,n\}$. By extending Equation 2.3, the transition relation in synchronous PBNs with perturbations is given by

$$T(\boldsymbol{x}(t), \boldsymbol{x}(t+1)) = \begin{cases} \boldsymbol{x}(t) \oplus \gamma(t) & \text{if } \gamma(t) \neq 0 \\ \boldsymbol{F}_t(\boldsymbol{x}(t)) & \text{otherwise,} \end{cases} \tag{2.5}$$

where $\oplus$ is the element-wise exclusive or operator for vectors. By extending Equation 2.4, the transition relation in asynchronous PBNs with perturbations is given as $T(\boldsymbol{x}(t), \boldsymbol{x}(t+1)) =$

$$\begin{cases} \boldsymbol{x}(t) \oplus \gamma(t) & \text{if } \gamma(t) \neq 0 \\ (x_1(t), x_2(t), \ldots, x_{i-1}(t), f_i(t)(s_i(t)), x_{i+1}(t), \ldots, x_n(t)) & \text{otherwise.} \end{cases} \tag{2.6}$$

The perturbations, by the latter update Equations 2.5 and 2.6, allow the system to move from any state to any other state in one single transition, hence render the underlying Markov chain irreducible and aperiodic. Therefore, the dynamics of a PBN with perturbations can be viewed as an ergodic DTMC [SD10]. The transition matrix[2] is given by $P_{\boldsymbol{x},\boldsymbol{x}'} = (1-p)^n \sum_{k=1}^{N} \mathbb{1}_{[\boldsymbol{f}_k(\boldsymbol{x})=\boldsymbol{x}']} \mathbb{P}(\boldsymbol{f}_k) + (1-(1-p)^n) p^{\eta(\boldsymbol{x},\boldsymbol{x}')} (1-p)^{n-\eta(\boldsymbol{x},\boldsymbol{x}')}$, where $\mathbb{1}$ is the indicator function and $\eta(\boldsymbol{x},\boldsymbol{x}')$ is the Hamming distance between states $\boldsymbol{x}, \boldsymbol{x}' \in S$. According to the ergodic theory, adding perturbations to any PBN assures that the long-run dynamics of the resulting PBN is governed by a unique limiting distribution, convergence to which is independent of the choice of the initial state. However, the perturbation probability value should be chosen carefully, not to dilute the behaviour of the original PBN. In this way the 'mathematical trick', although introduces some noise to the original system, allows to significantly simplify the analysis of the steady-state behaviour.

The density of a PBN is measured with its function number and parent nodes number. For a PBN G, its density is defined as $\mathcal{D}(G) = \frac{1}{n} \sum_{i=1}^{N_F} \omega(i)$, where n is the number of nodes in G, N_F is the total number of predictor functions in G, and $\omega(i)$ is the number of parent nodes for the ith predictor function.

[2]This is the transition matrix for instantaneously random PBNs. For context-sensitive PBNs, the transition matrix is different since the state also includes the current context as mentioned above.

Part I

Attractor Detection

3

Attractor Detection in Asynchronous Networks

3.1 Introduction

In this chapter, we consider attractor detection in asynchronous networks, in particular, asynchronous BNs without perturbations and asynchronous PBNs without perturbations. Perturbations are not introduced since they will make the underlying Markov chain of the network ergodic and hence no attractors exist any more. In a PBN without perturbations, attractors exist in its constituent BNs and the network remains in an attractor as long as it does not switch the context. So when detecting attractors of a PBN, we in fact detect the attractors of all its constituent BNs. Therefore, we will use asynchronous BNs to discuss attractor detection in the remaining part of this chapter. Usually in an instantaneously random PBN, the number of constituent BNs is relatively large and an attractor has a high probability to be escaped since the probability for switching a context is high. Hence, attractor detection is more performed on context-sensitive PBNs.

Attractor detection of a BN is non-trivial since attractors are determined based on the BN's states, the number of which is exponential in the number of nodes. In this chapter, we tackle the challenge of attractor detection for asynchronous BNs, especially for large ones, and we propose a strongly connected component (SCC) based decomposition method: decompose a BN into sub-networks called *blocks* according to the SCCs in the BN and recover attractors of the original BN based on attractors of the blocks. Since the *decomposition is performed on the BN structure, not in the state space*, the decomposition time cost is linear in the number of nodes and the state space of each block is exponentially smaller in comparison to that of the original BN. The asynchrony poses two main challenges for the decomposition methods: one is to take care of the dependency relations between different blocks; the other is to strictly comply with the asynchronous updating scheme when recovering attractors from different blocks. To overcome these difficulties, we order the blocks according to their dependency relations and detect attractors of each block with consideration of the block that it depends on. In this way, our method is *top-down*, starting with elementary blocks which do not depend on others. The result of this chapter is arranged as follows. We review the related work in attractor detection in Section 3.2. We prove that our proposed method can correctly detect all the attractors of a BN (Section 3.3), and we implement it using efficient BDD techniques (Section 3.4). Evaluation results show that our method can effectively detect attractors of two real-life biological networks (Section 3.5).

3.2 Related Work

A lot of efforts have been put in the development of attractor detection algorithms and tools. The simplest way to detect attractors is to enumerate all the possible states and to run simulation from each one until an attractor is reached [SG01]. This method ensures that all the attractors are detected but it has exponential time complexity and its applicability is highly restricted by the network size. Another approach is to take a sample from the whole state space and simulate from it until an attractor is found [Luc02]. However, this technique cannot guarantee finding all the attractors of a BN. Later, Irons proposed a method by analysing partial states involving parts of the nodes [Iro06]. This method can reduce the computational complexity of attractor detection from exponential time to polynomial time; however, it is highly dependent on the topology of the underlying network and the network size manageable by this method is restricted to 50.

Next, the efficiency and scalability of attractor detection techniques are further improved with the integration of two techniques. This first technique is based on Binary Decision Diagram (BDD), a compact data structure for representing Boolean functions. Algorithms proposed in [DTM05, GXMD07, GDCX$^+$08] explore BDDs to encode the Boolean functions in BNs, use BDD operations to capture the dynamics of the networks, and to build their corresponding transition systems. The efficient operations of BDDs are used to compute the forward and backward reachable states. Attractor detection is then reduced to finding self-loops or simple cycles in the transition systems, which highly relies on the computation of forward and backward reachable states. Garg *et al.* proposed a method for detecting attractors for both synchronous and asynchronous BNs [GXMD07]. Later in [GDCX$^+$08], the method was further improved for attractor detection of asynchronous BNs. In a recent work [ZYL$^+$13], Zheng *et al.* developed an algorithm based on reduced-order BDD (ROBDD) data structure, which further speeds up the computation time of attractor detection. These BDD-based solutions only work for GRNs of a hundred of nodes and suffer from the infamous state explosion problem, as the size of the BDD depends both on the regulatory functions and the number of nodes in the GRNs.

The other technique represents attractor detection in BNs as a satisfiability (SAT) problem [DT11]. The main idea is inspired by SAT-based bounded model checking: the transition relation of the GRN is unfolded into a bounded number of steps in order to construct a propositional formula which encodes attractors and which is then solved by a SAT solver. In every unfolding step a new variable is required to represent a state of a node in the GRN. It is clear that the efficiency of these algorithms largely depends on the number of unfolding steps required and the number of nodes in the GRN.

Recently, decomposition based algorithms have been developed for dealing with large BNs. Zhao *et at.* proposed an aggregation algorithm to deal with large BNs. Their idea is to decompose a large BN into several sub-networks and detect attractors of each sub-network [ZKF13]. By merging the attractors of all the sub-networks, their algorithm can reveal the attractors of the original BN. In [GYW$^+$14], Guo *et al.* developed an SCC (strongly connected component)-based decomposition method. Their method divides a BN into several sub-networks according to the SCCs in the BN and assigns each sub-network a credit. The attractor detection in each sub-network is performed one by one according to their credits. Unlike the algorithm in [ZKF13], when detecting attractors of a sub-network, the method of Guo *et al.* considers the attractor information of other sub-

networks whose credits are smaller. In this way, it reveals the attractors of the original BN by detecting attractors of the last sub-network. However, it is worth to point out that the algorithm designed by Guo in fact leads to wrong results in certain cases. An example showing this error is demonstrated in Example 4.2.2 of Chapter 4.

Remark. The above mentioned methods are mainly designed for BNs with the synchronous updating scheme. In synchronous BNs, an attractor is either a single state selfloop or a cycle since there is exactly one outgoing transition for each state. Under the asynchronous updating scheme, each state may have multiple outgoing transitions. Therefore, an attractor in general is a bottom strongly connected component (BSCC) [1] in the corresponding state transition system. The potentially complex attractor structure renders SAT-based methods ineffective as the respective SAT formulas become prohibitively large. Besides, the decomposition methods [ZKF13, GYW+14, YQPM16] are also prohibited by the asynchronous updating requirement. Moreover, BDD-based methods face the state-space explosion problem even in the synchronous updating scheme. In the asynchronous updating scheme, the problem gets even worse as the number of edges in the state transition system increases multiple times.

3.3 An SCC-based Decomposition Method

In this section, we describe in details our SCC-based decomposition method for detecting attractors of large asynchronous BNs and prove its correctness. The method consists of three main steps. First, we divide a BN into sub-networks called *blocks*. This step is performed based on the BN network structure and therefore it can be executed efficiently. Second, we detect attractors of each block. This step is performed on the constructed state transition system of the blocks. Finally, we recover attractors of the original BN by merging the detected attractors of the blocks.

3.3.1 Decomposing a BN into Blocks

We start a detailed presentation of our approach by giving the formal definition of a block.

Definition 3.3.1 (Block)**.** *Given a BN $G(V, \boldsymbol{f})$ with $V = \{v_1, v_2, \ldots, v_n\}$ and $\boldsymbol{f} = \{f_1, f_2, \ldots, f_n\}$, a* block *$B(V^B, \boldsymbol{f^B})$ is a subset of the network, where $V^B \subseteq V$ and $\boldsymbol{f^B}$ is a list of Boolean functions for nodes in V^B: for any node $v_i \in V^B$, if B contains all the parent nodes of v_i, its Boolean function in B remains the same as in G, i.e., f_i; otherwise, the Boolean function is undetermined, meaning that additional information is required to determine the value of v_i in B. We call the nodes with undetermined Boolean functions as* undetermined nodes*. We refer to a block as an* elementary block *if it contains no undetermined nodes.*

We consider asynchronous networks in this chapter and therefore a block is also under the asynchronous updating scheme, i.e., only one node in the block can be updated at any given time point no matter this node is undetermined or not.

We now introduce a method to construct blocks using SCC-based decomposition. Formally, the standard graph-theoretical definition of an SCC is as follows.

[1] It is also referred as *loose attractor* in the literature [WSA12].

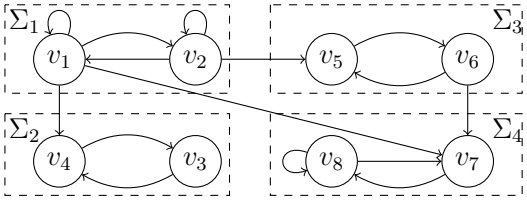
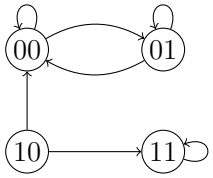
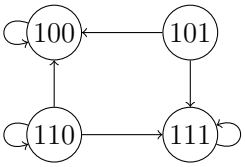


Figure 3.1: SCC decomposition of a BN.



(a) Transition graph of block B_1.



(b) Fulfilment 1 of Example 3.3.2.

Figure 3.2: Two transition graphs.

Definition 3.3.2 (SCC)**.** *Let $\mathcal{G}$ be a directed graph and $\mathcal{V}$ be its vertices. A strongly connected component (SCC) of $\mathcal{G}$ is a maximal set of vertices $C \subseteq \mathcal{V}$ such that for every pair of vertices u and v in C, there is a directed path from u to v and vice versa.*

We first decompose a given BN, its network structure, into SCCs. Figure 3.1 shows the decomposition of a BN into four SCCs: Σ_1, Σ_2, Σ_3, and Σ_4. A node outside an SCC that is a parent to a node in the SCC is referred to as a *control node* of this SCC. In Figure 3.1, node v_1 is a control node of Σ_2 and Σ_4; node v_2 is a control node of Σ_3; and node v_6 is a control node of Σ_4. The SCC Σ_1 does not have any control node.

Definition 3.3.3 (Parent SCC, Ancestor SCC)**.** *An SCC Σ_i is called a* parent SCC *(or* parent *for short) of another SCC Σ_j if Σ_i contains at least one control node of Σ_j. Denote $P(\Sigma_i)$ the set of parent SCCs of Σ_i. An SCC Σ_k is called an* ancestor SCC *(or* ancestor *for short) of an SCC Σ_j if and only if either (1) Σ_k is a parent of Σ_j or (2) Σ_k is a parent of Σ_j's ancestor. Denote $\Omega(\Sigma_j)$ the set of ancestor SCCs of Σ_j.*

An SCC together with its control nodes forms a *block*. For example, in Figure 3.1, Σ_2 and its control node v_1 form one block B_2. Σ_1 itself is a block, denoted as B_1, since the SCC it contains does not have any control node. If a control node in a block B_i is a determined node in another block B_j, block B_j is called a *parent* of block B_i and B_i is a child of B_j. The concepts of parent and ancestor are naturally extended to blocks.

By adding directed edges from all parent blocks to all their child blocks, we form a directed acyclic graph (DAG) of the blocks as the blocks are formed from SCCs. We notice here that in our decomposition approach, as long as the block graph is guaranteed to be a DAG, other strategies to form blocks can be used.

Two blocks can be merged into one larger block. For example, the above mentioned two blocks B_1 and B_2 can be merged to form a larger block $B_{1,2}$ which contains nodes v_1, v_2, v_3 and v_4. In the merged block $B_{1,2}$, there are no undetermined nodes since the parent nodes of all the nodes in $B_{1,2}$ are included in $B_{1,2}$.

A state of a block is a binary vector of length equal to the size of the block which determines the values of all the nodes in the block. In this thesis, we use a number of operations on the states of a BN and its blocks. Their definitions are given below.

Definition 3.3.4 (Projection map, Compressed state, Mirror states)**.** *For a BN G and its block B, where the set of nodes in B is $V^B = \{v_1, v_2, \ldots, v_m\}$ and the set of nodes in G is $V = \{v_1, v_2, \ldots, v_m, v_{m+1}, \ldots, v_n\}$, the* projection map $\delta_B : X \to X^B$ *is given by $\boldsymbol{x} = (x_1, x_2, \ldots, x_m, x_{m+1}, \ldots, x_n) \mapsto \delta_B(\boldsymbol{x}) = (x_1, x_2, \ldots, x_m)$. For any set of states $S \subseteq X$, we define $\delta_B(S) = \{\delta_B(\boldsymbol{x}) : \boldsymbol{x} \in S\}$. The projected state $\delta_B(\boldsymbol{x})$ is called a* compressed state *of $\boldsymbol{x}$. For any state $\boldsymbol{x}^B \in X^B$, we define its set of* mirror states *in G as $\mathcal{M}_G(\boldsymbol{x}^B) = \{\boldsymbol{x} \mid \delta_B(\boldsymbol{x}) = \boldsymbol{x}^B\}$. For any set of states $S^B \subseteq X^B$, its set of mirror states is $\mathcal{M}_G(S^B) = \{\boldsymbol{x} \mid \delta_B(\boldsymbol{x}) \in S^B\}$.*

The concept of the projection map can be extended to blocks. Given a block with nodes $V^B = \{v_1, v_2, \ldots, v_m\}$, let $V^{B'} = \{v_1, v_2, \ldots, v_j\} \subseteq V^B$. We can define $\delta_{B'} : X^B \to X^{B'}$ as $\boldsymbol{x}^B = (x_1, x_2, \ldots, x_m) \mapsto \delta_{B'}(\boldsymbol{x}^B) = (x_1, x_2, \ldots, x_j)$ and for a set of states $S^B \subseteq X^B$, we define $\delta_{B'}(S^B) = \{\delta_{B'}(\boldsymbol{x}^B) : \boldsymbol{x}^B \in S^B\}$.

Definition 3.3.5 (Path, Hyper-path)**.** *Given a BN G of n nodes and its state space $X = \{0,1\}^n$, a* path *of length k ($k \geqslant 2$) in G is a serial $\boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ of states in X such that there exists a transition between any consecutive two states $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$, where $i \in [1, k-1]$. A* hyper-path *of length k ($k \geqslant 2$) in G is a serial $\boldsymbol{x}_1 \dashrightarrow \boldsymbol{x}_2 \dashrightarrow \cdots \dashrightarrow \boldsymbol{x}_k$ of states in X such that at least one of the two conditions is satisfied: 1) there is a transition from $\boldsymbol{x}_i$ to $\boldsymbol{x}_{i+1}$, 2) $\boldsymbol{x}_i = \boldsymbol{x}_{i+1}$, where $i \in [1, k-1]$.*

The concepts of a path and a hyper-path in a BN can be naturally extended to elementary blocks. Notice that for any two consecutive states $\boldsymbol{x}_i$, $\boldsymbol{x}_{i+1}$ in a path $\boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ in a BN, $k \geqslant 2$ and $i \in [1, k-1]$, if the transition between these two states is due to the updating of a node in an elementary block B, then there is a transition from $\delta_B(\boldsymbol{x}_i)$ to $\delta_B(\boldsymbol{x}_{i+1})$; otherwise, $\delta_B(\boldsymbol{x}_i) = \delta_B(\boldsymbol{x}_{i+1})$. Therefore, the projection of all the states in the path $\boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ on block B actually forms a hyper-path $\delta_B(\boldsymbol{x}_1) \dashrightarrow \delta_B(\boldsymbol{x}_2) \dashrightarrow \cdots \dashrightarrow \delta_B(\boldsymbol{x}_k)$ in block B. The following lemma follows immediately from the definitions of path and hyper-path.

Lemma 3.3.1. *Let $\boldsymbol{x}_1 \dashrightarrow \boldsymbol{x}_2 \dashrightarrow \cdots \dashrightarrow \boldsymbol{x}_k$ be a hyper-path in a BN of length k. At least one of the two statements holds. 1) There is a path from $\boldsymbol{x}_1$ to $\boldsymbol{x}_k$ in the BN and this path contains all the states in the hyper-path. 2) $\boldsymbol{x}_1 = \boldsymbol{x}_2 = \cdots = \boldsymbol{x}_k$.*

3.3.2 Detecting Attractors in Blocks

An elementary block does not depend on any other block while a non-elementary block does. Therefore, they should be treated separately. We first consider the case of elementary blocks. An elementary block is in fact a BN; therefore, the notion of attractors of an elementary block is given by the definition of attractors of a BN. Next, we introduce the following concept.

Definition 3.3.6 (Preservation of attractors)**.** *Given a BN G and an elementary block B in G, let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be the set of attractors of G and $\mathcal{A}^B = \{A_1^B, A_2^B, \ldots, A_{m'}^B\}$ be the set of attractors of B. We say that B* preserves the attractors *of G if for any $k \in [1, m]$, there is an attractor $A_{k'}^B \in \mathcal{A}^B$ such that $\delta_B(A_k) \subseteq A_{k'}^B$.*

Example 3.3.1. *Consider the Boolean network G shown in Figure 3.1. The Boolean functions of this network are given as follows:*

$$\begin{cases} f_1 = x_1 \wedge x_2, & f_2 = x_1 \vee \neg x_2, \\ f_3 = \neg x_4, & f_4 = x_1 \wedge \neg x_3, \\ f_5 = x_2 \wedge x_6, & f_6 = x_5, \\ f_7 = (x_1 \vee x_6) \wedge x_8, & f_8 = x_7 \vee x_8. \end{cases}$$

It has 10 attractors, i.e., $\mathcal{A} = \{\{(0*100000)\}, \{(0*100001)\}, \{(11010000)\}, \{(11010011)\}, \{(11011100)\}, \{(11011111)\}, \{(11100000)\}, \{(11100011)\}, \{(11101100)\}, \{(11101111)\}\}$ ($*$ means either 0 or 1). Nodes v_1 and v_2 form an elementary block B_1. Since B_1 is an elementary block, it can be viewed as a BN. The transition graph of this block is shown in Figure 3.2a. Its set of attractors is $\mathcal{A}^{B_1} = \{\{(0*)\}, \{(11)\}\}$ (nodes are arranged as v_1, v_2). We have $\delta_{B_1}(\{(0*100000)\}) = \{(0*)\} \in \mathcal{A}^{B_1}$ and $\delta_{B_1}(\{(0*100001)\}) = \{(0*)\} \in \mathcal{A}^{B_1}$. For the remaining 8 attractors of G, their compressed set of state is always $\{(11)\}$, which belongs to $\mathcal{A}^{B_1}$. Hence, block B_1 preserves the attractors of the original BN G.

With Definition 3.3.6, we have the following lemma and theorem.

Lemma 3.3.2. *Given a BN G and an elementary block B in G, let Φ be the set of attractor states of G and Φ^B be the set of attractor states of B. If B preserves the attractors of G, then $\Phi \subseteq \mathcal{M}_G(\Phi^B)$.*

Proof. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be the set of attractors of G and $\mathcal{A}^B = \{A_1^B, A_2^B, \ldots, A_{m'}^B\}$ be the set of attractors of B. Since B preserves the attractors of G, for any $k \in [1, m]$, there exists a $k' \in [1, m']$ such that $\delta_B(A_k) \subseteq A_{k'}^B$. Therefore, $\delta_B(\Phi) = \cup_{i=1}^{m} \delta_B(A_i) \subseteq \cup_{i=1}^{m'} A_i^B = \Phi^B$. By Definition 3.3.4, we have that $\Phi \subseteq \mathcal{M}_G(\delta_B(\Phi))$. Hence, $\Phi \subseteq \mathcal{M}_G(\Phi^B)$. □

Theorem 3.3.1. *Given a BN G, let B be an elementary block in G. B preserves the attractors of G.*

Proof. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be the set of attractors of G. For any $i \in [1, m]$, let $L = \boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ be a path containing all the states in A_i and let $\boldsymbol{x}_1 = \boldsymbol{x}_k$. According to Definition 3.3.5, $\delta_B(\boldsymbol{x}_1) \dashrightarrow \delta_B(\boldsymbol{x}_2) \dashrightarrow \cdots \dashrightarrow \delta_B(\boldsymbol{x}_k)$ is a hyper-path in B. We denote this hyper-path as L^B. Therefore, one of the following two conditions must hold: 1) there exists a path L' from $\delta_B(\boldsymbol{x}_1)$ to $\delta_B(\boldsymbol{x}_k)$ in B; 2) $\delta_B(\boldsymbol{x}_1) = \delta_B(\boldsymbol{x}_2) = \cdots = \delta_B(\boldsymbol{x}_k)$. Given that the choice of the attractor A_i is arbitrary, the claim holds if we can prove that states in the hyper-path L^B form an attractor of B under both conditions. We will prove them one by one.

Condition 1: Given the arbitrary choice of the path, when the first condition holds, the states in this path can reach each other. Now we only need to prove that the states in this path cannot reach any other state that is not in this path. We prove by contradiction. Assume a state $\delta_B(\boldsymbol{x}_i)$ in path L' can reach state $\delta_B(\boldsymbol{x}_i')$ by applying the Boolean function of some node v_p and $\delta_B(\boldsymbol{x}_i')$ is not in L'. Hence there is a transition from $\boldsymbol{x}_i$ to $\boldsymbol{x}_i'$ in G. Since L contains all the states in A_i and A_i is an attractor, necessarily $\boldsymbol{x}_i'$ is contained by L. Therefore, $\delta_B(\boldsymbol{x}_i')$ is one of the states in the hyper-path L^B. According to Lemma 3.3.1, all states in L^B are contained by L', in particular $\delta_B(\boldsymbol{x}_i')$. This is contradictory to the assumption. It follows that states of L^B form an attractor of B.

Condition 2: This condition holds only when all transitions in path L are performed by applying Boolean functions of nodes that are not in block B. For any $j \in [1, k-1]$, let $\boldsymbol{x}_{j'}$ be any state reachable from $\boldsymbol{x}_j$ by one transition. We have $\boldsymbol{x}_{j'} \in A_i$ and therefore L contains $\boldsymbol{x}_{j'}$. Hence L^B contains $\delta_B(\boldsymbol{x}_{j'})$ and $\delta_B(\boldsymbol{x}_{j'}) = \delta_B(\boldsymbol{x}_1) = \delta_B(\boldsymbol{x}_2) = \cdots = \delta_B(\boldsymbol{x}_k)$. Given the choice of $\boldsymbol{x}_j$ and $\boldsymbol{x}_{j'}$ is arbitrary, $\delta_B(A_1) = \{\delta_B(\boldsymbol{x}_1)\}$, which is a singleton attractor in B. □

For an elementary block B in a BN G, the mirror states of its attractor states cover all G's attractor states according to Lemma 3.3.2 and Theorem 3.3.1. Therefore, by searching from the mirror states only instead of the whole state space, we can detect all the attractor states of G.

We now proceed to consider the case of non-elementary blocks. For an SCC Σ_j, if it has no parent SCC, then this SCC forms an elementary block; if it has at least one parent, then it must have an ancestor that has no parent, and all its ancestors $\Omega(\Sigma_j)$ together can form an elementary block, which is also a BN. The SCC-based decomposition will result in at least one elementary block and usually one or more non-elementary blocks. Moreover, for each non-elementary block we can construct by merging all its predecessor blocks a single parent elementary block. We detect the attractors of the elementary blocks and use the detected attractors to guide the values of the control nodes of their child blocks. The guidance is achieved by considering *fulfilment* of the dynamics of a child block with respect to the attractors of its parent elementary block. In some cases, a fulfilment of a block is simply obtained by assigning new Boolean functions to the control nodes of the block. However, in many cases, it is not this simple and a fulfilment of a block is obtained by explicitly constructing a transition system of this block corresponding to the considered attractor of the elementary parent block. Since the parent block of a non-elementary block may have more than one attractor, a block may have more than one fulfilment.

By the following two definitions, we explain in details what fulfilments are. We first introduce the concept of crossability and cross operations in Definition 3.3.7. The concept of crossability specifies a special relation between states of a non-elementary block and of its parent blocks, while the cross operations are used for merging attractors of two blocks when recovering the attractors of the original BN.

Definition 3.3.7 (Crossability, Cross operations)**.** *Let G be a BN and let B_i be a non-elementary block in G with the set of nodes $V^{B_i} = \{v_{p_1}, v_{p_2}, \ldots, v_{p_s}, v_{q_1}, v_{q_2}, \ldots, v_{q_t}\}$, where q_k ($k \in [1,t]$) are the indices of the control nodes also contained in B_i's parent block B_j and p_k ($k \in [1,s]$) are the indices of the remaining nodes. We denote the set of nodes in B_j as $V^{B_j} = \{v_{q_1}, v_{q_2}, \ldots, v_{q_t}, v_{r_1}, v_{r_2}, \ldots, v_{r_u}\}$, where r_k ($k \in [1,u]$) are the indices of the non-control nodes in B_j. Let further $\boldsymbol{x}^{B_i} = (x_1, x_2, \ldots, x_s, y_1^i, y_2^i, \ldots, y_t^i)$ be a state of B_i and $\boldsymbol{x}^{B_j} = (y_1^j, y_2^j, \ldots, y_t^j, z_1, z_2, \ldots, z_u)$ be a state of B_j. States $\boldsymbol{x}^{B_i}$ and $\boldsymbol{x}^{B_j}$ are said to be* crossable*, denoted as $\boldsymbol{x}^{B_i}$ $\mathcal{C}$ $\boldsymbol{x}^{B_j}$, if the values of their common nodes are the same, i.e., $y_k^i = y_k^j$ for all $k \in [1,t]$. The cross operation of two crossable states $\boldsymbol{x}^{B_i}$ and $\boldsymbol{x}^{B_j}$ is defined as $\Pi(\boldsymbol{x}^{B_i}, \boldsymbol{x}^{B_j}) = (x_1, x_2, \ldots, x_s, y_1^i, y_2^i, \ldots, y_t^i, z_1, z_2, \ldots, z_u)$. The notion of crossability naturally extends to two elementary blocks; any two states of any two elementary blocks are always crossable.*

We say a set of states $S^{B_i} \subseteq X^{B_i}$ and a set of states $S^{B_j} \subseteq X^{B_j}$ are crossable, denoted as S^{B_i} $\mathcal{C}$ S^{B_j}, if at least one of the sets is empty or the following two conditions hold: 1) for any state $\boldsymbol{x}^{B_i} \in S^{B_i}$, there always exists a state $\boldsymbol{x}^{B_j} \in S^{B_j}$ such that $\boldsymbol{x}^{B_i}$ and

$\boldsymbol{x}^{B_j}$ are crossable; 2) vice versa. The cross operation of two crossable non-empty sets of states S^{B_i} and S^{B_j} are defined as $\Pi(S^{B_i}, S^{B_j}) = \{\Pi(\boldsymbol{x}^{B_i}, \boldsymbol{x}^{B_j}) \mid \boldsymbol{x}^{B_i} \in S^{B_i}, \boldsymbol{x}^{B_j} \in S^{B_j}$ and $\boldsymbol{x}^{B_i} \mathcal{C} \boldsymbol{x}^{B_j}\}$. When one of the two sets is empty, the cross operation simply returns the other set, i.e., $\Pi(S^{B_i}, S^{B_j}) = S^{B_i}$ if $S^{B_j} = \emptyset$ and $\Pi(S^{B_i}, S^{B_j}) = S^{B_j}$ if $S^{B_i} = \emptyset$. Let $\mathcal{S}^{B_i} = \{S^{B_i} \mid S^{B_i} \subseteq X^{B_i}\}$ be a set of states set in B_i and $\mathcal{S}^{B_j} = \{S^{B_j} \mid S^{B_j} \subseteq X^{B_j}\}$ be a set of states set in B_j. We say $\mathcal{S}^{B_i}$ and $\mathcal{S}^{B_j}$ are crossable, denoted as $\mathcal{S}^{B_i} \mathcal{C} \mathcal{S}^{B_j}$ if for any states set $S^{B_i} \in \mathcal{S}^{B_i}$, there always exists a states set $S^{B_j} \in \mathcal{S}^{B_j}$ such that S^{B_i} and S^{B_j} are crossable; 2) vice versa. The cross operation of two crossable sets of states sets $\mathcal{S}^{B_i}$ and $\mathcal{S}^{B_j}$ are defined as $\Pi(\mathcal{S}^{B_i}, \mathcal{S}^{B_j}) = \{\Pi(S_i, S_j) \mid S_i \in \mathcal{S}^{B_i}, S_j \in \mathcal{S}^{B_j}$ and $S_i \mathcal{C} S_j\}$.

The crossability is similar to the join operation in relational database. With the crossability defined, the definition of a fulfilment is now given as follows.

Definition 3.3.8 (Fulfilment of a block)**.** *Let B_i be a non-elementary block formed by merging an SCC with its control nodes. Let nodes $u_1, u_2, \ldots, u_r$ be all the control nodes of B_i which are also contained by its single and elementary parent block B_j (we can always merge all B_i's ancestor blocks to form B_j if B_i has more than one parent block or has a non-elementary parent block). Let $A_1^{B_j}, A_2^{B_j}, \ldots, A_t^{B_j}$ be the AS' of B_j. For any $k \in [1, t]$, a* fulfilment *of block B_i with respect to $A_k^{B_j}$ is a state transition system such that*

1. *a state of the system is a vector of the values of all the nodes in the block;*
2. *the state space of this fulfilment is crossable with $A_k^{B_j}$;*
3. *for any transition $\boldsymbol{x}^{B_i} \to \tilde{\boldsymbol{x}}^{B_i}$ in this fulfilment, if this transition is caused by a non-control node, the transition should be regulated by the Boolean function of this node; if this transition is caused by the updating of a control node, one can always find two states $\boldsymbol{x}^{B_j}$ and $\tilde{\boldsymbol{x}}^{B_j}$ in $A_k^{B_j}$ such that there is a transition from $\boldsymbol{x}^{B_j}$ to $\tilde{\boldsymbol{x}}^{B_j}$ in $A_k^{B_j}$, $\boldsymbol{x}^{B_i} \mathcal{C} \boldsymbol{x}^{B_j}$ and $\tilde{\boldsymbol{x}}^{B_i} \mathcal{C} \tilde{\boldsymbol{x}}^{B_j}$;*
4. *for any transition $\boldsymbol{x}^{B_j} \to \tilde{\boldsymbol{x}}^{B_j}$ in $A_k^{B_j}$, one can always find a transition $\boldsymbol{x}^{B_i} \to \tilde{\boldsymbol{x}}^{B_i}$ in this fulfilment such that $\boldsymbol{x}^{B_i} \mathcal{C} \boldsymbol{x}^{B_j}$ and $\tilde{\boldsymbol{x}}^{B_i} \mathcal{C} \tilde{\boldsymbol{x}}^{B_j}$.*

Constructing fulfilments for a non-elementary block is the key process for obtaining its attractors. For each fulfilment, the construction process requires the knowledge of all the transitions in the corresponding attractor of the parent block. In Section 3.4, we explain in details how to implement it with BDDs.

Example 3.3.2. *Consider the BN shown in Figure 3.1. The network contains four SCCs $\Sigma_1, \Sigma_2, \Sigma_3$ and Σ_4. For any Σ_i ($i \in [1, 4]$), we form a block B_i by merging Σ_i with its control nodes. Block B_1 is an elementary block and its transition graph is shown in Figure 3.2a. Block B_1 has two attractors, i.e., $\{(11)\}$ and $\{(0*)\}$. Regarding the first attractor, block B_3 has a fulfilment by setting node v_2 to contain only the transition $\{(1) \to (1)\}$. Its transition graph is shown in Figure 3.2b. Regarding the second attractor, block B_3 has a fulfilment by setting node v_2 to contain the following transitions $\{(0) \to (*), (1) \to (*)\}$. The transition graph of this fulfilment is shown in Figure 3.3.*

Lemma 3.3.3. *Let B_j be a single, elementary parent block of a non-elementary block B_i in a BN G. Let A^{B_j} be an attractor of B_j and let A^{B_i} be an attractor in the fulfilment of B_i with respect to A^{B_j}. Then $A^{B_i} \mathcal{C} A^{B_j}$.*

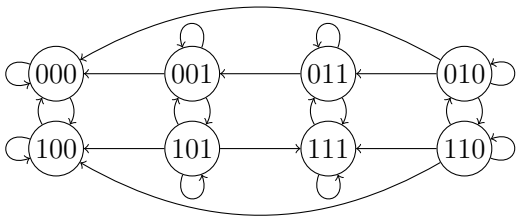


Figure 3.3: Fulfilment 2 of Example 3.3.2.

Proof. By the definition of fulfilment we have that for any state $\boldsymbol{x}^{B_i} \in A^{B_i}$, there exists a state $\boldsymbol{x}^{B_j} \in A^{B_j}$ such that $\boldsymbol{x}^{B_j}\ \mathcal{C}\ \boldsymbol{x}^{B_i}$.

Let us denote the set of control nodes of B_i with Z, the set of the remaining nodes in B_i with V, and use $\boldsymbol{zv}$ to represent a state of block B_i where $\boldsymbol{z}$ are the values for the nodes in Z and $\boldsymbol{v}$ are the values for the nodes in V. Let L^{B_j} be a closed path, i.e., the first and the last state are the same, in the transition system of B_j which contains all the states in A^{B_j}. Let $\boldsymbol{x}^{B_i}$ be any state in A^{B_i}. Due to the asynchronous updating scheme and the fact that the nodes in Z are independent of the nodes in V, one obtains that $\boldsymbol{z}\delta_V(\boldsymbol{x}^{B_i}) \in A^{B_i}$ for any $\boldsymbol{z} \in \delta_Z(A^{B_j}) = \delta_Z(L^{B_j})$. For this it is enough to observe that any of these states can be reached from $\boldsymbol{x}^{B_i}$ by following the corresponding sequence of transitions in the hyper-path obtained by projecting L^{B_j} on Z. In consequence, for any state $\boldsymbol{x}^{B_j} \in A^{B_j}$ we have that $\boldsymbol{x}^{B_j}\ \mathcal{C}\ \delta_Z(\boldsymbol{x}^{B_j})\delta_V(\boldsymbol{x}^{B_i})$ and $\delta_Z(\boldsymbol{x}^{B_j})\delta_V(\boldsymbol{x}^{B_i}) \in A^{B_i}$. Hence, $A^{B_i}\ \mathcal{C}\ A^{B_j}$. □

A fulfilment of a block takes care of the dynamics of the undetermined nodes and instantiates a transition system of the block. Therefore, we can extend the attractor definition to fulfilments and to non-elementary blocks as follows.

Definition 3.3.9 (Attractors of a non-elementary block)**.** *An attractor of a fulfilment of a non-elementary block is a set of states satisfying that any state in this set can be reached from any other state in this set and no state in this set can reach any other state that is not in this set. The attractors of a non-elementary block is the union of the attractors of all fulfilments of the block.*

With the definition of attractors of non-elementary blocks, we can relax Definition 3.3.8 by allowing B_j to be a single and either elementary or non-elementary parent block with known attractors. This is due to the fact that when forming the fulfilments of a non-elementary block, we only need the attractors of its parent block that contains all its control nodes, no matter whether this parent block is elementary or not. In other words, computing attractors for non-elementary blocks requires the knowledge of the attractors of its parent block that contains all its control nodes. Therefore, we need to consider blocks in a specific order which guarantees that when computing attractors for block B_i, the attractors of its parent block that contains all B_i's control nodes are already available. To facilitate this, we introduce the concept of a credit as follows.

Definition 3.3.10 (Credit)**.** *Given a BN G, an elementary block B_i of G has a credit of 0, denoted as $\mathcal{P}(B_i) = 0$. Let B_j be a non-elementary block and $B_{j_1}, \ldots, B_{j_{p(j)}}$ be all its parent blocks. The credit of B_j is $\mathcal{P}(B_j) = max_{k=1}^{p(j)}(\mathcal{P}(B_{j_k})) + 1$.*

3.3.3 Recovering Attractors of the Original BN

After identifying attractors for all the blocks, we need to recover attractors for the original BN. This is achievable by the following theorem for recovering the attractors of two blocks.

Theorem 3.3.2. *Given a BN G with B_i and B_j being its two blocks, let $\mathcal{A}^{B_i}$ and $\mathcal{A}^{B_j}$ be the set of attractors for B_i and B_j, respectively. Let $B_{i,j}$ be the block got by merging the nodes in B_i and B_j. If B_i and B_j are both elementary blocks or B_i is an elementary and single parent block of B_j, then it holds that $\mathcal{A}^{B_i} \mathcal{C} \mathcal{A}^{B_j}$ and $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$ is the set of attractors of $B_{i,j}$.*

Proof. We first prove that $\mathcal{A}^{B_i} \mathcal{C} \mathcal{A}^{B_j}$. If B_i and B_j are two elementary blocks, they do not share common nodes. Then it holds by definition that $\mathcal{A}^{B_i} \mathcal{C} \mathcal{A}^{B_j}$. Now, let B_i be the only elementary parent block of B_j. By definition, the attractors of B_j is the set of the attractors of all fulfilments of B_j. Due to this definition, for any attractor $A^{B_i} \in \mathcal{A}^{B_i}$, one can always find an attractor $A^{B_j} \in \mathcal{A}^{B_j}$ such that $A^{B_i} \mathcal{C} A^{B_j}$. For this it is enough to consider the fulfilment of B_j with respect to A^{B_i} and to take as A^{B_j} one of the attractors of this fulfilment. By Lemma 3.3.3 we have that $A^{B_i} \mathcal{C} A^{B_j}$. Further, again by Lemma 3.3.3, for any attractor $A^{B_j} \in \mathcal{A}^{B_j}$, there is an attractor $A^{B_i} \in \mathcal{A}^{B_i}$ such that $A^{B_i} \mathcal{C} A^{B_j}$, i.e., the one that gives rise to the fulfilment of which A^{B_j} is an attractor in B_j. Therefore, $\mathcal{A}^{B_i} \mathcal{C} \mathcal{A}^{B_j}$.

We now prove that $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$ is the set of attractors of $B_{i,j}$. This is equivalent to showing the following two statements: 1) for any $A \in \Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$, A is an attractor of $B_{i,j}$; 2) any attractor of $B_{i,j}$ is contained in $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$. We prove them one by one.

Statement 1: Let A be any set of states in $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$. Then there exist $A^{B_i} \in \mathcal{A}^{B_i}$ and $A^{B_j} \in \mathcal{A}^{B_j}$ such that $A = \Pi(A^{B_i}, A^{B_j})$ and $A^{B_i} \mathcal{C} A^{B_j}$. We first prove that $\boldsymbol{x} = \Pi(\delta_{B_i}(\boldsymbol{x}), \delta_{B_j}(\boldsymbol{x}))$, where $\delta_{B_i}(\boldsymbol{x}) \in A^{B_i}$ and $\delta_{B_j}(\boldsymbol{x}) \in A^{B_j}$, cannot reach any state that is not in A by contradiction. Assume that $\boldsymbol{x}$ can reach a state $\boldsymbol{y}$ by one transition and $\boldsymbol{y} \notin A$. Due to asynchronous updating mode, the transition from $\boldsymbol{x}$ to $\boldsymbol{y}$ is caused by updating one node. There are three possibilities: 1) the updated node is in B_i and it is not a control node of B_j; 2) the updated node is in B_j and it is not a control node; 3) the updated node is a control node of B_j. In the first case, there is a transition from $\delta_{B_i}(\boldsymbol{x})$ to $\delta_{B_i}(\boldsymbol{y})$ in the elementary block B_i and since $\delta_{B_i}(\boldsymbol{x})$ belongs to attractor A^{B_i}, it follows that $\delta_{B_i}(\boldsymbol{y}) \in A^{B_i}$. In addition, we have $\delta_{B_j}(\boldsymbol{y}) = \delta_{B_j}(\boldsymbol{x})$. Then $\boldsymbol{y} = \Pi(\delta_{B_i}(\boldsymbol{y}), \delta_{B_j}(\boldsymbol{x}))$ and $\boldsymbol{y} \in A$. Similarly in the second case, there is a transition from $\delta_{B_j}(\boldsymbol{x})$ to $\delta_{B_j}(\boldsymbol{y})$ within the attractor system A^{B_j}, so $\delta_{B_j}(\boldsymbol{y}) \in A^{B_j}$ and $\boldsymbol{y} = \Pi(\delta_{B_i}(\boldsymbol{x}), \delta_{B_j}(\boldsymbol{y})) \in A$. In the third case, there is a transition from $\delta_{B_i}(\boldsymbol{x})$ to $\delta_{B_i}(\boldsymbol{y})$ in the elementary block B_i and, as in the first case, we have that $\delta_{B_i}(\boldsymbol{y}) \in A^{B_i}$. Since $A^{B_i} \mathcal{C} A^{B_j}$, there exists $\boldsymbol{s} \in A^{B_j}$ such that $\delta_{B_i}(\boldsymbol{y}) \mathcal{C} \boldsymbol{s}$. Let us denote the set of control nodes of B_j with Z, the set of the remaining nodes in B_j with V, and use $\boldsymbol{zv}$ to represent a state of block B_j where $\boldsymbol{z}$ are the values for the nodes in Z and $\boldsymbol{v}$ are the values for the nodes in V. Now, $\boldsymbol{s} = \delta_Z(\boldsymbol{s})\delta_V(\boldsymbol{s})$ and there is a path from $\delta_{B_j}(\boldsymbol{x})$ to $\boldsymbol{s}$ in the attractor system A^{B_j} as both states belong to A^{B_j}. Since at each step of this path the value of only a single node is updated and the the control nodes in Z are updated independently of the nodes in V, it follows that starting from $\delta_{B_j}(\boldsymbol{x})$ and by following only the updates related to the control nodes in Z in the path from $\delta_{B_j}(\boldsymbol{x})$ to $\boldsymbol{s}$, there is a path in the attractor system A^{B_j} from $\delta_{B_j}(\boldsymbol{x})$ to $\delta_Z(\boldsymbol{s})\delta_V(\boldsymbol{x}) = \delta_{B_j}(\boldsymbol{y})$. Hence, $\delta_{B_j}(\boldsymbol{y}) \in A^{B_j}$ and we have that

$\boldsymbol{y} = \Pi(\delta_{B_i}(\boldsymbol{y}), \delta_{B_j}(\boldsymbol{y})) \in A$. In all three cases we reach a contradiction.

We now show that for any two states $\boldsymbol{a}, \boldsymbol{x} \in A = \Pi(A^{B_i}, A^{B_j})$, $\boldsymbol{x}$ is reachable from $\boldsymbol{a}$ only via states in A. We have $\delta_{B_i}(\boldsymbol{a})$, $\delta_{B_i}(\boldsymbol{x}) \in A^{B_i}$ and there is a path L^{B_i} from $\delta_{B_i}(\boldsymbol{a})$ to $\delta_{B_i}(\boldsymbol{x})$ in A^{B_i}. Similarly, there is a path L^{B_j} from $\delta_{B_j}(\boldsymbol{a})$ to $\delta_{B_j}(\boldsymbol{x})$ in the attractor system of A^{B_j}. Following the same updating rules as in the path L^{B_i}, there is a path $L_1^{B_{i,j}}$ in $B_{i,j}$ from state $\boldsymbol{a}$ to state $\boldsymbol{y}$ such that $\delta_{B_i}(\boldsymbol{y}) = \delta_{B_i}(\boldsymbol{x})$ and the non-control nodes of B_j in $\boldsymbol{y}$ have the same values as in $\boldsymbol{a}$. The claim holds if we can prove that there is a path $\overline{L^{B_j}}$ in the attractor system of A^{B_j} from state $\delta_{B_j}(\boldsymbol{y})$ to $\delta_{B_j}(\boldsymbol{x})$ since following the same updating rules as in the path $\overline{L^{B_j}}$, there is a path $L_2^{B_{i,j}}$ in $B_{i,j}$ from $\boldsymbol{y}$ to $\boldsymbol{x}$ and hence $\boldsymbol{x}$ is reachable from $\boldsymbol{a}$. We prove this in the following two cases. The first case is when B_i and B_j are both elementary blocks. In this case, the merged block $B_{i,j}$ is in fact a BN and we have $\delta_{B_j}(\boldsymbol{a}) = \delta_{B_j}(\boldsymbol{y})$. Therefore, the path L^{B_j} is in fact $\overline{L^{B_j}}$. We now consider the second case where B_i is a parent of B_j. Using the notation introduced above, we show that the state $\delta_{B_j}(\boldsymbol{y}) = \delta_Z(\boldsymbol{y})\delta_V(\boldsymbol{a}) \in A^{B_j}$. This follows from applying the corresponding argumentation for node update possibilities one or three presented above at each step of the path L^{B_i}. Now, since both $\delta_{B_j}(\boldsymbol{y})$ and $\delta_{B_j}(\boldsymbol{x})$ belong to A^{B_j}, there is a path from $\delta_{B_j}(\boldsymbol{y})$ to $\delta_{B_j}(\boldsymbol{x})$ in the attractor system of A^{B_j}. This path is exactly the searched path $\overline{L^{B_j}}$. Given the choice of $\boldsymbol{a}$ and $\boldsymbol{x}$ is arbitrary, we can claim that any two states in A are reachable from each other. Moreover, since a state in A cannot reach any state outside A as shown above, the two states in A are reachable from each other via states only in A. Hence, Statement 1 follows.

Statement 2: We prove that $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$ contains all the attractors of $B_{i,j}$. Let $A^{B_{i,j}}$ be an attractor in $B_{i,j}$. Since the nodes in B_i are independent of the nodes in B_j, clearly $\delta_{B_i}(A^{B_{i,j}})$ is an attractor in B_i. Therefore, $\delta_{B_i}(A^{B_{i,j}}) \in \mathcal{A}^{B_i}$.

Let us consider the fulfilment of block B_j with respect to $\delta_{B_i}(A^{B_{i,j}})$. We proceed to show that $\delta_{B_j}(A^{B_{i,j}})$ is an attractor of this fulfilment. Let us assume that there exists $\boldsymbol{x} \in \delta_{B_j}(A^{B_{i,j}})$ such that it can reach a state $\boldsymbol{y} \notin \delta_{B_j}(A^{B_{i,j}})$ by one transition in the fulfilment. Let $\tilde{\boldsymbol{x}} \in A^{B_{i,j}}$ be the corresponding state of $\boldsymbol{x}$ in $A^{B_{i,j}}$, i.e. $\delta_{B_j}(\tilde{\boldsymbol{x}}) = \boldsymbol{x}$. It follows that there exists a state $\tilde{\boldsymbol{y}}$ of $B_{i,j}$ reachable from $\tilde{\boldsymbol{x}}$ by one transition such that $\delta_{B_j}(\tilde{\boldsymbol{y}}) = \boldsymbol{y}$. In consequence, $\tilde{\boldsymbol{y}} \notin A^{B_{i,j}}$ and $A^{B_{i,j}}$ cannot be an attractor. This contradicts the original assumption.

Now we show that there is a path between any two states $\boldsymbol{x}$ and $\boldsymbol{y}$ of $\delta_{B_j}(A^{B_{i,j}})$ in the fulfilment only via states in $\delta_{B_j}(A^{B_{i,j}})$. The existence of such path follows in a straightforward way from the fact that there exist two corresponding states $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{y}}$ in $A^{B_{i,j}}$ such that $\delta_{B_j}(\tilde{\boldsymbol{x}}) = \boldsymbol{x}$ and $\delta_{B_j}(\tilde{\boldsymbol{y}}) = \boldsymbol{y}$. In consequence, there is a path from one to the other as both are in the attractor $A^{B_{i,j}}$. Projection of this path on B_j forms a hyper-path in the fulfilment. By Lemma 3.3.1, $\boldsymbol{y}$ is reachable from $\boldsymbol{x}$ in the fulfilment and only via states in $\delta_{B_j}(A^{B_{i,j}})$ as shown above. Hence, $\delta_{B_j}(A^{B_{i,j}})$ is an attractor of the considered fulfilment, i.e. $\delta_{B_j}(A^{B_{i,j}}) \in \mathcal{A}^{B_j}$.

Finally, it is straightforward to verify that $\delta_{B_i}(A^{B_{i,j}})\ \mathcal{C}\ \delta_{B_j}(A^{B_{i,j}})$. Therefore, $A \in \Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$, which concludes the proof of Statement 2 and the theorem. □

Finally, from Theorem 3.3.2 we obtain the following corollary which states that for specific configurations of blocks, certain orderings according to which the blocks are merged are equivalent in terms of the resulting attractor set for the merged block.

Corollary 3.3.1. *Given a BN G with B_i, B_j, and B_k being its three blocks, let $\mathcal{A}^{B_i}$, $\mathcal{A}^{B_j}$, and $\mathcal{A}^{B_k}$ be the sets of attractors for blocks B_i, B_j, and B_k, respectively. If the three blocks are all elementary blocks or B_i is an elementary block and it is the only parent block of B_j and B_k, it holds that $\Pi(\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j}), \mathcal{A}^{B_k}) = \Pi(\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_k}), \mathcal{A}^{B_j})$.*

Proof. According to Theorem 3.3.2, $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$ is the set of attractors of $B_{i,j}$ and $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_k})$ is the set of attractors of $B_{i,k}$. Merging B_i with B_j results in an elementary block $B_{i,j}$, and merging B_i with B_k results in an elementary block $B_{i,k}$. Applying Theorem 3.3.2 again, we get $\Pi(\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j}), \mathcal{A}^{B_k})$ is the set of attractors of $B_{i,j,k}$ and $\Pi(\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_k}), \mathcal{A}^{B_j})$ is the set of attractors of $B_{i,k,j}$. Since $B_{i,j,k}$ and $B_{i,k,j}$ are actually the same block, $\Pi(\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j}), \mathcal{A}^{B_k}) = \Pi(\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_k}), \mathcal{A}^{B_j})$. □

The above developed theoretical background with Theorem 3.3.2 being its core result, allows us to design a new decomposition-based approach towards detection of attractors in large asynchronous BNs. The idea is as follows. We divide a BN into blocks according to the detected SCCs. We sort the blocks in ascending order based on their credits and detect attractors of the ordered blocks one by one in an iterative way. According to Theorem 3.3.2, we can perform a cross operation for any two elementary blocks (credits 0) or an elementary block (credit 0) with one of its child blocks (credit 1) which has no other parent blocks to recover the attractors of the two merged blocks. The resulting merged block will form a new elementary block, i.e., one with credit 0. By iteratively performing the cross operation until a single elementary block containing all the nodes of the BN is obtained, we can recover the attractors of the original BN. The details of this new approach are discussed in the next section.

3.4 Implementation

In this section, we explain how we implement the above mentioned decomposition method using BDDs. We first introduce the concept of BDDs and how to encode BNs in BDDs in Section 3.4.1. We then introduce a BDD-based algorithm to detect attractors for relatively small BNs in Section 3.4.2. and describe how our SCC-based decomposition method can be implemented using the BDD-based algorithm in Section 3.4.3.

3.4.1 Encoding BNs in BDDs

Binary decision diagrams (BDDs) were introduced to represent Boolean functions [Lee59, Ake78]. A BDD consists of three types of nodes: a root node, (intermediate) decision nodes, and two terminal nodes, i.e., 0-terminal and 1-terminal. It uses a decision node to represent a variable of a Boolean function. Each decision node contains two outgoing edges, representing the two possible values, i.e., 0 and 1, of the variable. A path from the root node to the 1-terminal represents an assignment of values to the variables that results in the true value of the Boolean function; while a path from the root node to the 0-terminal represents an assignment of values to the variables that results in the false value of the Boolean function. BDDs have the advantage of memory efficiency and have been applied in model checking algorithms to alleviate the state space explosion problem. A BN $G(V, \boldsymbol{f})$ can be easily encoded in a BDD by modelling a BN as an STS. Each variable in V can be represented by a binary BDD variable. By slight

abuse of notation, we use V to denote the set of BDD variables. In order to encode the transition relation, another set V' of BDD variables, which is a copy of V, is introduced: V encodes the possible current states, i.e., $\boldsymbol{x}(t)$, and V' encodes the possible next states, i.e., $\boldsymbol{x}(t+1)$. Hence, the transition relation T can be viewed as a Boolean function $T^f : 2^{|V|+|V'|} \rightarrow \{0,1\}$, where values 1 and 0 indicate a valid and an invalid transition, respectively. Our attractor detection algorithms also use two basis functions: $Image(X,T) = \{s' \in S \mid \exists s \in X \text{ such that } (s,s') \in T\}$, which returns the set of target states that can be reached from any state in $X \subseteq S$ with a single transition in T; $Preimage(X,T) = \{s' \in S \mid \exists s \in X \text{ such that } (s',s) \in T\}$, which returns the set of predecessor states that can reach a state in X with a single transition. To simplify the presentation, we also define $Preimage^i(X,T) = \underbrace{Preimage(...(Preimage(X,T)))}_{i}$ with $Preimage^0(X,T) = X$. Thus, the set of all states that can reach a state in X via transitions in T is defined as a fix point $Predecessors(X,T) = \bigcup_{i=0}^{n} Preimage^n(X,T)$ such that $Preimage^n(X,T) = Preimage^{n+1}(X,T)$. Given a set of states $X \subseteq S$, the projection $T|_X$ of T on X is defined as $T|_X = \{(s,s') \in T \mid s \in X \wedge s' \in X\}$.

The BDD b representing a state $s = (x_1, x_2, \ldots, x_n)$ can be seen as a Boolean formula $g(b) = (v_1 = x_1) \wedge (v_2 = x_2) \wedge \ldots \wedge (v_n = x_n)$. Let $g(b)^{-i} = (v_1 = x_1) \wedge \ldots \wedge (v_{i-1} = x_{i-1}) \wedge (v_{i+1} = x_{i+1}) \ldots \wedge (v_n = x_n)$ $(1 \leq i \leq n)$. The *existential abstraction* of v_i from b produces a new BDD $b|_{\{v_i\}}$ equivalent to the Boolean formula $g(b)^{-i} \wedge (v_i = x_i \vee v_i = \neg x_i)$. For our convenience, we say that node v_i is set to value "-" by existential abstraction and the new BDD can be written as $g(b)^{-i} \wedge (v_i =$ "-" $)$. The existential abstraction can be applied to a set $V' \subseteq V$ of nodes on a set of states $S' \subseteq S$, written as $S'|_{V'}$. The intersection of two BDDs b_1 and b_2, written as $b_1 \cap b_2$, is equivalent to the Boolean formula $g(b_1) \wedge g(b_2)$.

3.4.2 A BDD-based Attractor Detection Algorithm

Attractors of an asynchronous BN are in fact bottom strongly connected components (BSCCs) in the state transition system of the BN. Thus, detecting attractors is the same as detecting the BSCCs. Formally, the definition of BSCCs is given as follows.

Definition 3.4.1. *A bottom strongly connected component (BSCC) is an SCC Σ such that no state outside Σ is reachable from Σ.*

We encode a BN with BDDs, and adapt the hybrid Tarjan algorithm described in Algorithm 7 of [KPQ11] to detect BSCCs in the corresponding transition system of the BN. Given a state transition system $\mathcal{T} = \langle S, S_0, T \rangle$, our attractor detection algorithm DETECT($\mathcal{T}$) in Algorithm 1 computes the set of BSCCs in $\mathcal{T}$. If $\mathcal{T}$ is converted from a BN G, then DETECT($\mathcal{T}$) computes all the attractors of G. The correctness of Algorithm 1 is guaranteed by the following two propositions.

Proposition 3.4.1. *The first SCC returned by the Tarjan's algorithm is a BSCC.*

Proposition 3.4.2. *If a state that reaches a BSCC is located outside the BSCC, then this state is not contained by any BSCC.*

The first proposition can be deduced from the fact that the Tarjan's algorithm is a depth-first search. The second one comes from the definition of BSCCs, as no states inside

Algorithm 1 Attractor detection using the hybrid Tarjan's algorithm

```
1: procedure DETECT(𝒯)
2:     𝒜 := ∅; X := S;                                  //S is from 𝒯
3:     while X ≠ ∅ do
4:         Randomly pick a state s ∈ X;
5:         Σ := HybridTarjan(s, T);                     //a variant of Tarjan's algorithm
6:         𝒜 := 𝒜 ∪ Σ;
7:         X := X \ Predecessors(Σ, T);
8:     end while
9:     return 𝒜.
10: end procedure
```

a BSCC can lead to a state in any other BSCC. In Algorithm 1, the hybrid Tarjan algorithm $HybridTarjan(s, T)$ takes as input a starting state s and the transition relation T. When it finds the first SCC Σ (also a BSCC), which is reached from s, it terminates immediately and returns Σ.

With the use of BDD representation, DETECT($\mathcal{T}$) can deal with relatively small BNs (e.g., a BN with tens of nodes) with small memory usage. Moreover, the computation of SCCs can also benefit from the efficient BDD operations. However, real life biological BNs usually contain hundreds of nodes and the state space is exponential in the number of nodes. Therefore, DETECT($\mathcal{T}$) would still suffer from the state space explosion problem when dealing with large BNs. Thus for large BNs, we propose to use the SCC-based decomposition method as described in Section 3.3. We now give the algorithm for implementing this method in the following section.

3.4.3 An SCC-based Decomposition Algorithm

We describe the detection process in Algorithm 2. This algorithm takes a BN G and its corresponding transition system $\mathcal{T}$ as inputs and outputs the set of attractors of G. Lines 23-26 of this algorithm describe the process for detecting attractors of a non-elementary block. The algorithm detects the attractors of all the fulfilments of the non-elementary block and performs the union operation on the sets of detected attractors. For this, if the non-elementary block has only one parent block, its attractors are already computed as the blocks are considered in ascending order with respect to their credits by the main **for** loop in Line 4. Otherwise, all the ancestor blocks are considered in the **for** loop in Lines 14-21. By iteratively applying the cross operation in Line 17 to the attractor sets of the ancestor blocks in ascending order, the attractors of a new block formed by merging all the ancestor blocks are computed as assured by Theorem 3.3.2. The new block is in fact an elementary block which is a single parent of the considered non-elementary block. By considering blocks in ascending order, the order in which blocks with the same credit are considered does not influence the final result due to Corollary 3.3.1. The correctness of the algorithm is stated as Theorem 3.4.1.

Theorem 3.4.1. *Algorithm 2 correctly identifies the set of attractors of a given BN G.*

Proof. Algorithm 2 divides a BN into SCC blocks and detects attractors of each block. Line 5 to 27 describe the process for detecting attractors of a block. The algorithm

Algorithm 2 SCC-based decomposition algorithm

procedure SCC_DETECT(G, $\mathcal{T}$)
  $B :=$ FORM_BLOCK(G); $\mathcal{A} := \emptyset$; $B_a := \emptyset$; $k :=$ size of B;
  initialise $\mathcal{A}^\ell$; *//$\mathcal{A}^\ell$ is a dictionary storing the set of attractors for each block*
  for $i := 1; i <= k; i++$ **do**
    if B_i is an elementary block **then**
      $\mathcal{T}^{B_i} :=$ transition system converted from B_i;
      $\mathcal{A}_i :=$ DETECT($\mathcal{T}^{B_i}$);
    else $\mathcal{A}_i := \emptyset$;
      if B_i^p is the only parent block of B_i **then**
        $\mathcal{A}_i^p := \mathcal{A}^\ell.getAtt(B_i^p)$; *//obtain attractors of B_i^p*
      else $B^p := \{B_1^p, B_2^p, \ldots, B_m^p\}$ be the ancestor blocks
        of B_i (ascending ordered);
        $B_c := B_1^p$; *//B^p is ordered based on credit*
        for $j := 2; j <= m; j++$ **do**
          $B_{c,j} :=$ a new block containing nodes in B_c and B_j^p;
          if $(\mathcal{A}_i^p := \mathcal{A}^\ell.getAtt(B_{c,j})) == \emptyset$ **then**
            $\mathcal{A}_i^p := \Pi(\mathcal{A}^\ell.getAtt(B_c), \mathcal{A}^\ell.getAtt(B_j))$;
            $\mathcal{A}^\ell.add(B_{c,j}, \mathcal{A}_i^p)$;
          end if
          $B_c := B_{c,j}$;
        end for
      end if
      for $A \in \mathcal{A}_i^p$ **do**
        $\mathcal{T}^{B_i}(A) := \langle S^{B_i}(A), T^{B_i}(A) \rangle$; *//obtain the fulfilment of B_i with A*
        $\mathcal{A}_i := \mathcal{A}_i \cup$ DETECT($\mathcal{T}^{B_i}(A)$);
      end for
    end if
    $\mathcal{A}^\ell.add(B_i, \mathcal{A}_i)$; *//the add operation will not add duplicated elements*
    if $B_a! = \emptyset$ **then** $\mathcal{A} = \Pi(\mathcal{A}_i, \mathcal{A})$; $B_a := B_{a,i}$; $\mathcal{A}^\ell.add(B_a, \mathcal{A})$;
    else $B_a := B_i$
    end if
  end for
  return $\mathcal{A}$.
end procedure

procedure FORM_BLOCK(G)
  decompose G into SCCs and form blocks with SCCs and their control nodes;
  sort the blocks in ascending order according to their credits;
  $B := (B_1, \ldots, B_k)$;
  return B. *//B is the list of blocks after ordering*
end procedure

Figure 3.4: Transition graphs of the two fulfilments for block B_2.

distinguishes between two different types of blocks. The first type is an elementary block. Since it is in fact a BN, the attractors of this type of block are directly detected via Algorithm 1. The second type is a non-elementary block. The algorithm constructs the fulfilments of this type of block, detects attractors of each fulfilment and merges them as the attractors of the block. The algorithm takes special care of those blocks with more than one parent blocks. It merges all the ancestor blocks of such a block as its parent block. Since the ancestor blocks are in ascending operations based on their credits, the cross operation in Line 18 will iteratively recover the attractors of the parent block according to Theorem 3.3.2. Whenever the attractors of a block B_i are detected, it performs a cross operation between block B_i and the elementary block B_c formed by nodes in all previous blocks (Line 31). According to Theorem 3.3.2, the cross operation will result in the attractors of the block formed by nodes in the two blocks. Since Algorithm 2 iteratively performs this operation to all the blocks, it will recover the attractors of the BN in the last iteration. Note that how to order two blocks with the same credit does not affect the result of this algorithm, as proved in Corollary 3.3.1. □

The algorithm stores all computed attractors for the original SCC blocks and all auxiliary merged blocks in the dictionary structure $\mathcal{A}^\ell$. We use BDDs to encode transitions and the fulfilments are performed via BDD operations directly. Given a BN $G(V, \boldsymbol{f})$ with n nodes, our implementation, which is based on the CUDD library [Som15], encodes the whole network with $2n$ BDD variables. Each state in G is encoded by n BDD variables, and a projection of a state on a subset of nodes $V' \subseteq V$ is performed by setting all BDD variables for nodes in $V \backslash V'$ to "-", which represents that its value can be either 0 or 1, and therefore, can be ignored. As a state for a block B is encoded by $|V^B|$ BDD variables, the variables in $V \backslash V^B$ are set to "-" in the BDD representation. This way, after we verify that S^{B_i} and S^{B_j} are crossable, i.e., $S^{B_i}\ \mathcal{C}\ S^{B_j}$, the cross operation $\Pi(S^{B_i}, S^{B_j})$ is equivalent to the AND operation on two BDDs, i.e., $bdd_{S^{B_i}}$ and $bdd_{S^{B_j}}$ encoding S^{B_i} and S^{B_j}, respectively. Formally, we have that $\Pi(S^{B_i}, S^{B_j}) = bdd_{S^{B_i}} \cap bdd_{S^{B_j}}$. Let $\mathcal{T}^B = \langle S^B, T^B \rangle$ be the transition system converted from block B, and let V^C be the set of control nodes in B. The set of states $S^B(A)$ of the fulfilment of block B with respect to attractor A is $\mathcal{M}_B(\delta_C(A))$ and the transition relation $T^B(A)$ of the fulfilment is $T^B|_{S^B(A)}$. We continue to illustrate in the following example how Algorithm 2 detects attractors.

Example 3.4.1. *Consider the BN shown in Example 3.3.2 and its four blocks. Block B_1 is an elementary block and it has two attractors, i.e., $\mathcal{A}_1 = \{\{(0*)\}, \{(11)\}\}$. To detect the attractors of block B_2, we first form fulfilments of B_2 with respect to the attractors of its parent block B_1. B_1 has two attractors so there are two fulfilments for B_2. The transition graphs of the two fulfilments are shown in Figure 3.4. We get three attractors for block B_2, i.e., $\mathcal{A}_2 = \{\{(010)\}, \{(101)\}, \{(110)\}\}$. Performing a cross operation, we get the attractors of the merged block $B_{1,2}$, i.e., $\mathcal{A}_{1,2} = \Pi(\mathcal{A}_1, \mathcal{A}_2) = \{\{(0 * 10)\}, \{(1101)\}, \{(1110)\}\}$. In Example 3.3.2, we have shown*

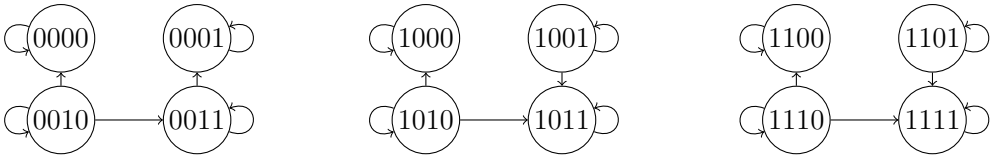


Figure 3.5: Transition graphs of the three fulfilments for block B_4.

the two fulfilments of B_3 with respect to the two attractors of B_1. Clearly, B_3 has three attractors, i.e., $\mathcal{A}_3 = \{\{(*00)\}, \{(100)\}, \{(111)\}\}$. Merging $B_{1,2}$ and B_3, we get the attractors of the merged block $B_{1,2,3}$, i.e., $\mathcal{A}_{1,2,3} = \Pi(\mathcal{A}_{1,2}, \mathcal{A}_3) = \{\{(0 * 10\ 00)\}, \{(110100)\}, \{(110111)\}, \{(111000)\}, \{(111011)\}\}$. B_4 has two parent blocks. Therefore, we need to merge B_4's ancestors (B_1 and B_3) as its new parent block. After merging, we get the attractors of the merged block as $\mathcal{A}_{1,3} = \Pi(\mathcal{A}_1, \mathcal{A}_3) = \{\{(0 * 00)\}, \{(1100)\}, \{(1111)\}\}$. There are three attractors so there will be three fulfilments for block B_4. The transition graphs of the three fulfilments are shown in Figure 3.5. From the transition graphs, we easily get the attractors of B_4, i.e., $\mathcal{A}_4 = \{\{(0000)\}, \{(0001)\}, \{(1000)\}, \{(1011)\}, \{(1100)\}, \{(1111)\}\}$. Now the attractors for all the blocks have been detected. We can then obtain the attractors of the BN by applying one more cross operation, i.e., $\mathcal{A} = \mathcal{A}_{1,2,3,4} = \Pi(\mathcal{A}_{1,2,3}, \mathcal{A}_4) = \{\{(0 * 1\ 00000)\}, \{(0*100001)\}, \{(11010000)\}, \{(11010011)\}, \{(11011100)\}, \{(11011111)\}, \{(11100000)\}, \{(11100011)\}, \{(11101100)\}, \{(11101111)\}\}$.

3.5 Evaluation

We have implemented the decomposition algorithm presented in Section 3.4 in the model checker MCMAS [LQR15]. In this section, we demonstrate the efficiency of our method using two real-life biological systems. One is a logical MAPK network model of [GCBP+13] containing 53 nodes and the other is a Boolean network model of apoptosis, originally presented in [SSV+09], containing 97 nodes. All the experiments are conducted on a computer with an Intel Xeon W3520@2.67GHz CPU and 12GB memory. Notice that we tried to apply genYsis [GXMD07] to these two systems, but it failed in both cases to detect attractors within 5 hours.

MAPK network. Mitogen-activated protein kinases (MAPKs) are a family of serine/ threonine kinases that transduce biochemical signals from the cell membrane to the nucleus in response to a wide range of stimuli, such as growth factors, hormones, inflammatory cytokines and environmental stresses. Cascades of these kinases participate in multiple intracellular signalling pathways that control a wide range of cellular processes, e.g. cell cycle machinery, differentiation, survival and apoptosis. MAPK pathways are highly evolutionary conserved among all eukaryotic cells and allow the cells to respond coordinately to multiple and diverse inputs. To date, three main pathways have been extensively studied: Extracellular Regulated Kinases (ERK), Jun NH_2 Terminal Kinases (JNK), and p38 Kinases (p38), named after their specific MAPK kinases involved. These pathways are characterised by enormous crass-talk with each other, which gives rise to a complex network of molecular interactions [KN08]. Malfunctioning of MAPK signalling mechanisms is often observed in cancer [DHRK07]. Therefore, a deeper comprehension of the MAPK pathways and their interactions is of

utter importance to elucidate the roles of MAPKs in the development and progression of cancer. This in turn is crucial for the development of new, effective therapeutic strategies. In [GCBP+13] a predictive dynamical Boolean model of the MAPK network is presented. It recapitulates observed responses of the MAPK network to characteristic stimuli in selected urinary bladder cancers together with its specific contribution to cell fate decision on proliferation, apoptosis, and growth arrest. For the wiring of the logical model, we refer to [GCBP+13]. In our study we consider two mutants of the model: one with EGFR over-expression and the other with FGFR3 activating mutation which correspond to the r3 and r4 variants of [GCBP+13], respectively, and therefore we refer to them as as MAPK_r3 and MAPK_r4. However, in contrast to the original variants r3 and r4, we do not set the values for the four stimuli nodes to 0 but perform the computations for all 2^4 possible fixed sets of values for these nodes. For the remaining nodes, all possible initial states are considered as in [GCBP+13]. In consequence, our results for MAPK_r3 and MAPK_r4 include the attractors for variants r7, r13 and r8, r14 of [GCBP+13], respectively. The structure of the network is shown in Figure 3.6. The corresponding SCC structure of the mutant MAPK_r3 is shown in Figure 3.7 and Table 3.1. We compute the attractors of the MAPK_r3 and MAPK_r4 BNs using both the BDD-based algorithm, i.e., Algorithm 1 and our decomposition algorithm, i.e., Algorithm 2. The two algorithms compute the same attractors for the same network. We show in the left part of Table 4.1 the number of attractors and the computational time costs (in seconds) for both mutants. Besides, we show the speedups of Algorithm 2 with respect to Algorithm 1.

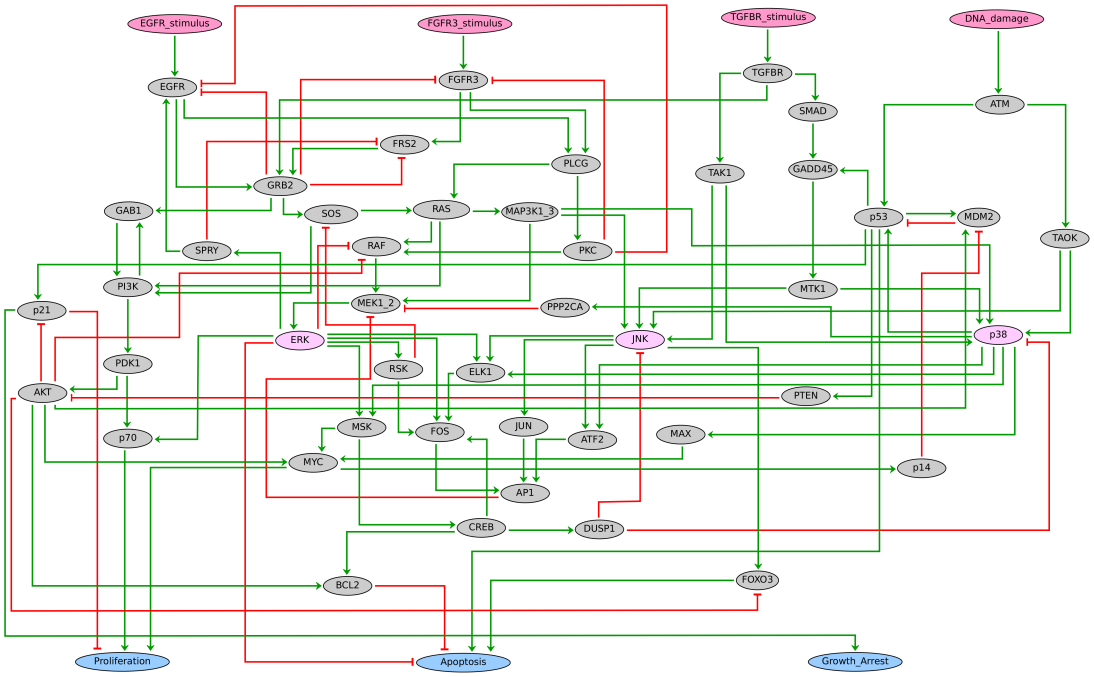


Figure 3.6: Wiring of the MAPK logical model of [GCBP+13]. The diagram contains three types of nodes: stimuli nodes (pink), signalling component nodes (gray) with highlighted MAPK protein nodes (light pink), and cell fate nodes (blue). Green arrows and red blunt arrows represent positive and negative regulations, respectively. For detailed information on the Boolean model of the MAPK network containing all modelling assumptions and specification of the logical rules refer to [GCBP+13] and the supplementary material thereof.

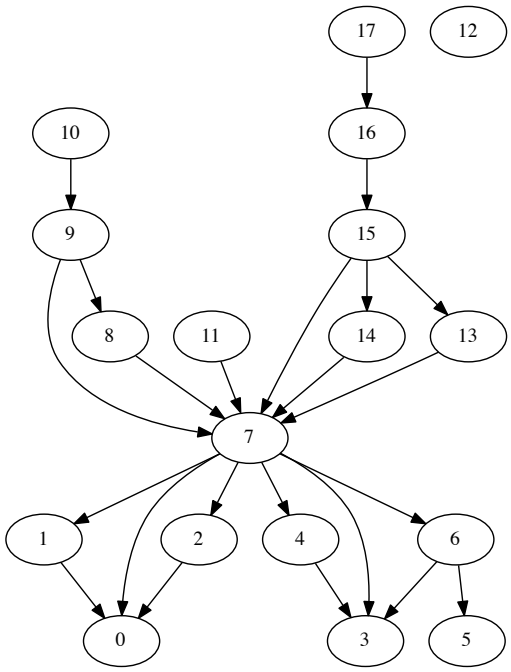


Figure 3.7: The SCC structure of the MAPK network (mutant MAPK_r3). Each node represents an SCC. Model components contained in each SCC are listed in Table 3.1. For each pair of a parent SCC and one of its child SCCs, a directed edge is drawn pointing from the parent SCC to the child SCC. Node 12 is not connected to any other node as EGFR is set to be always true and hence the influence from EGFR_stimulus (node 12) is cut. The SCC structure of mutant MAPK_r4 is virtually the same; the only difference is that model components contained in certain SCCs are slightly different: EGFR is switched with FGFR3 and EGFR_stimulus is switched with FGFR3_stimulus.

scc #	nodes	scc #	nodes	scc #	nodes	scc #	nodes
0	Apoptosis	4	p70	9	ATM	13	FGFR3_stimulus
1	BCL2	5	Growth_Arrest	10	DNA_damage	14	SMAD
2	FOXO3	6	p21	11	EGFR	15	TAK1
3	Proliferation	8	TAOK	12	EGFR_stimulus	16	TGFBR
17	TGFR_stimulus						
7	AKT AP1 ATF2 CREB DUSP1 FGFR3 ELK1 ERK FOS FRS2 GAB1 GADD45 GRB2 JNK JUN MAP3K1_3 MAX MDM2 MEK1_2 MSK MTK1 MYC PDK1 PI3K PKC PLCG PPP2CA PTEN RAF RAS RSK SOS SPRY p14 p38 p53						

Table 3.1: Nodes of the MAPK pathway (mutant r3) in SCCs as shown in Figure 3.7.

In our study we consider two mutants of the model: one with EGFR over-expression and the other with FGFR3 activating mutation which correspond to the r3 and r4 variants of [GCBP$^+$13], respectively, and therefore we refer to them as as MAPK_r3 and MAPK_r4. However, in contrast to the original variants r3 and r4, we do not set the values for the four stimuli nodes to 0 but perform the computations for all 2^4 possible fixed

Networks	# attractors	Time(s) Algorithm 1	Time(s) Algorithm 2	Speedup
MAPK_r3	20	6.070	2.614	**2.32**
MAPK_r4	24	11.674	1.949	**5.99**
apoptosis	1024	1633.970	103.856	**15.73**
apoptosis*	2048	8564.680	218.230	**39.25**

Table 3.2: Evaluation results on two real-life biological systems.

sets of values for these nodes. For the remaining nodes, all possible initial states are considered as in [GCBP+13]. In consequence, our results for MAPK_r3 and MAPK_r4 include the attractors for variants r7, r13 and r8, r14 of [GCBP+13], respectively. We compute the attractors of the MAPK_r3 and MAPK_r4 BNs using both the BDD-based algorithm, i.e., Algorithm 1 and our decomposition algorithm, i.e., Algorithm 2. The SCC structure of mutant MAPK_r3 is shown in Figure 3.7 and the nodes in all the SCCs are shown in Table 3.1. We show in the rows of networks MAPK_r3 and MAPK_r4 in Table 3.2 the number of attractors and the computational time costs for both mutants. Besides, we show the speedups of Algorithm 2 with respect to Algorithm 1. Notice that our computations are performed for the full model presented in Figure 3.6 contrary to [GCBP+13], where various reduced models were used for the computations of attractors due to their computation limit.

scc #	nodes	scc #	nodes	scc #	nodes	scc #	nodes
0	apoptosis	16	C8a_DISCa_2	32	IRS_P2	47	UV
1	gelsolin	17	C8a_DISCa	33	IRS	48	UV_2
2	C3a_c_IAP	18	proC8	34	IKKdeact	49	FASL
3	I_kBb	19	p38	35	FLIP	50	PKA
4	CAD	20	ERK1o2	36	DISCa_2	51	cAMP
5	PARP	21	Ras	37	DISCa	52	AdCy
6	ICAD	22	Grb2_SOS	38	FADD	53	GR
7	JNK	23	Shc	39	Bid	54	Glucagon
8	C8a_FLIP	24	Raf	40	housekeeping	55	Insulin
10	XIAP	25	MEK	41	FAS_2	56	smac_mimetics
11	TRADD	26	Pak1	42	FAS	57	P
12	RIP	27	Rac	43	FASL_2	58	T2R
13	Bad_14_3_3	28	GSK_3	44	IL_1	59	T2RL
14	P14_3_3	29	Bad	45	TNFR_1		
15	C8a_2	31	IR	46	TNF		
9	Apaf_1 apopto A20 Bax Bcl_xl BIR1_2 c_IAP c_IAP_2 complex1 comp1_IKKa cyt_c C3ap20 C3ap20_2 C3a_XIAP C8a_comp2 C9a FLIP_2 NIK RIP_deubi smac smac_XIAP tBid TRAF2 XIAP_2 IKKa I_kBa I_kBe complex2 NF_kB C8a C3ap17 C3ap17_2						
30	IRS_P PDK1 PKB PKC PIP3 PI3K C6						

Table 3.3: Nodes of the apoptosis network in SCCs as shown in Figure 3.9.

Apoptosis network. Apoptosis is a process of programmed cell death and has been linked to many diseases. It is often regulated by several signaling pathways extensively linked by crosstalks. We take the apoptosis signalling network presented in [SSV+09]

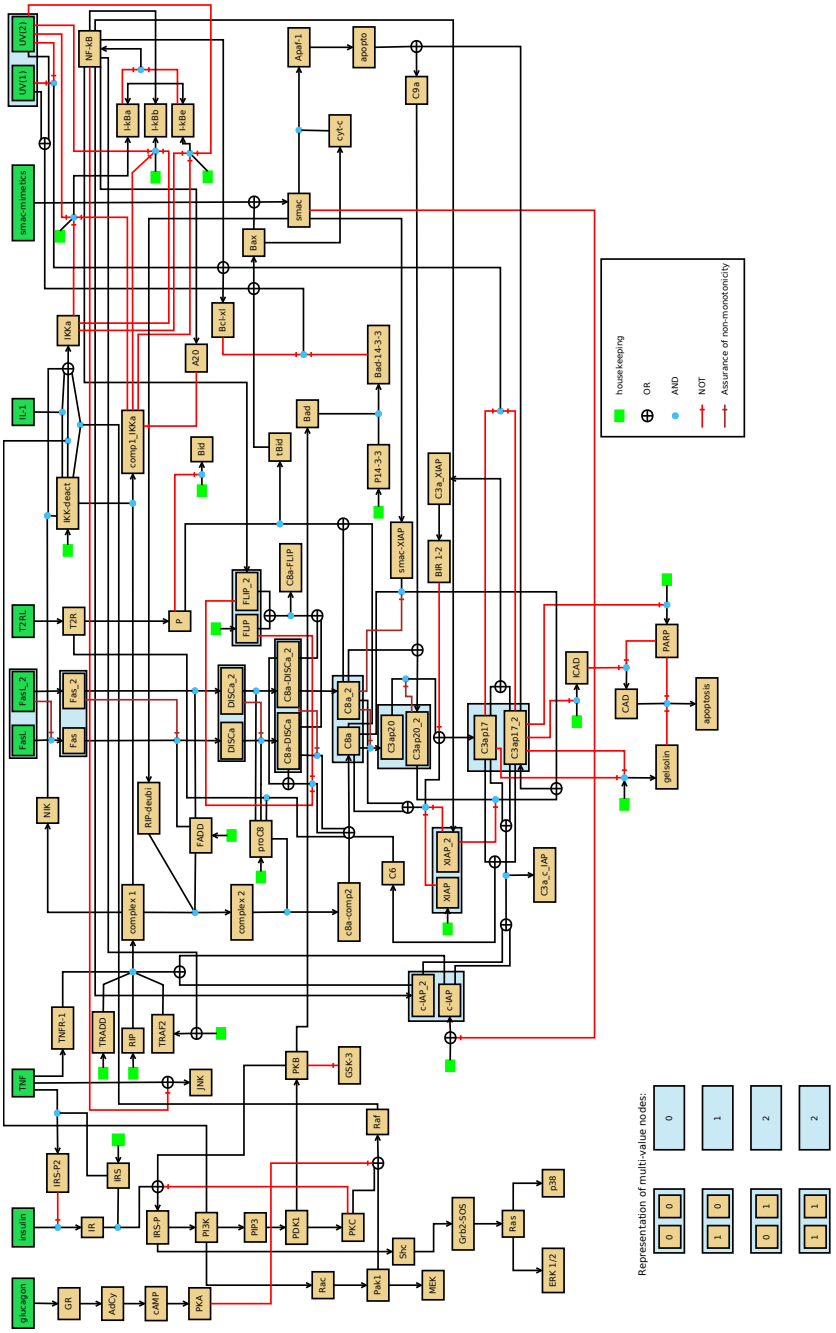


Figure 3.8: The wiring of the multi-value logic model of apoptosis by Schlatter et al. [SSV+09] recast into a binary Boolean network. For clarity of the diagram the nodes I-kBa, I-kBb, and I-kBe have two positive inputs. The inputs are interpreted as connected via $\oplus$ (logical OR).

and recast it into the Boolean network framework: a BN model which compromise 97 nodes. In this network, there are 10 input nodes. One of them is a housekeeping node which value is fixed to 1 and which is used to model constitutive activation of certain nodes in the network. For the wiring of the BN model, see Figure 3.8. The SCC structure of this network is shown in Figure 3.9 and the nodes in all SCCs are shown in Table 3.3. Similar to the MAPK network, we compute the attractors of the apoptosis network with both Algorithm 1 and Algorithm 2. The results are shown in the right part of Table 3.2. Moreover, we also consider the network where the value of housekeeping is not fixed and show the result in the row apoptosis*. When the housekeeping node is not fixed, the state-space of the network is doubled. The results clearly indicate that our proposed decomposition method provides better speedups with respect to Algorithm 1 for larger models.

3.6 Discussions and Future Work

We have presented an SCC-based decomposition method for detecting attractors in large asynchronous BNs, which often arise and are important in the holistic study of biological systems. This problem is very challenging as the state space of such networks is exponential in the number of nodes in the networks and therefore huge. Meanwhile, asynchrony greatly increases the difficulty of attractor detection as the density of the transition graph is inflated dramatically and the structure of attractors may be complex. Our method performs SCC-based decomposition of the network structure of a give BN to manage the cyclic dependencies among network nodes, computes the attractors of each SCC, and finally recovers the attractors of the original BN by merging the detected (partial) attractors. To the best of our knowledge, our method is the first scalable one able to deal with large biological systems modelled as asynchronous BNs, thanks to its divide and conquer strategy. We have prototyped our method and performed experiments with two real biological networks. The obtained results are very promising.

We have observed that the network structure of BNs can vary quite a lot, which potentially has impact on the performance of our proposed method. In principle, our method works well on large networks which contain several relatively small SCCs. Each of the two mutants of the MAPK network, however, contains one large SCC with 36 nodes and 17 SCCs each with one node only. Moreover, the large SCC is in the middle of the SCC network structure (see Figure 3.7). This network structure in fact does not fit well with our method. This explains why the speedups achieved for this network are less than 10. Both the MAPK network and the apoptosis network contain many small SCCs with only one node (see Figure 3.7 and Figure 3.9). One way to improve our method is to merge these small SCCs into larger blocks so that there will be fewer iterations in the main loop of Algorithm 1. Moreover, the single-node SCCs which do not have child SCCs are in fact leaves and they can be removed to reduce the network size. When the attractors in the reduced network are detected, we can then recover the attractors in the whole network.[2] Such optimisations will be part of our future work. We will also apply our method to other realistic large biological networks and we will develop optimisations fitted towards different SCC network structures.

[2]This is in general related to network reduction techniques (e.g., see [SAA10]) which aim to simplify the networks prior to dynamic analysis.

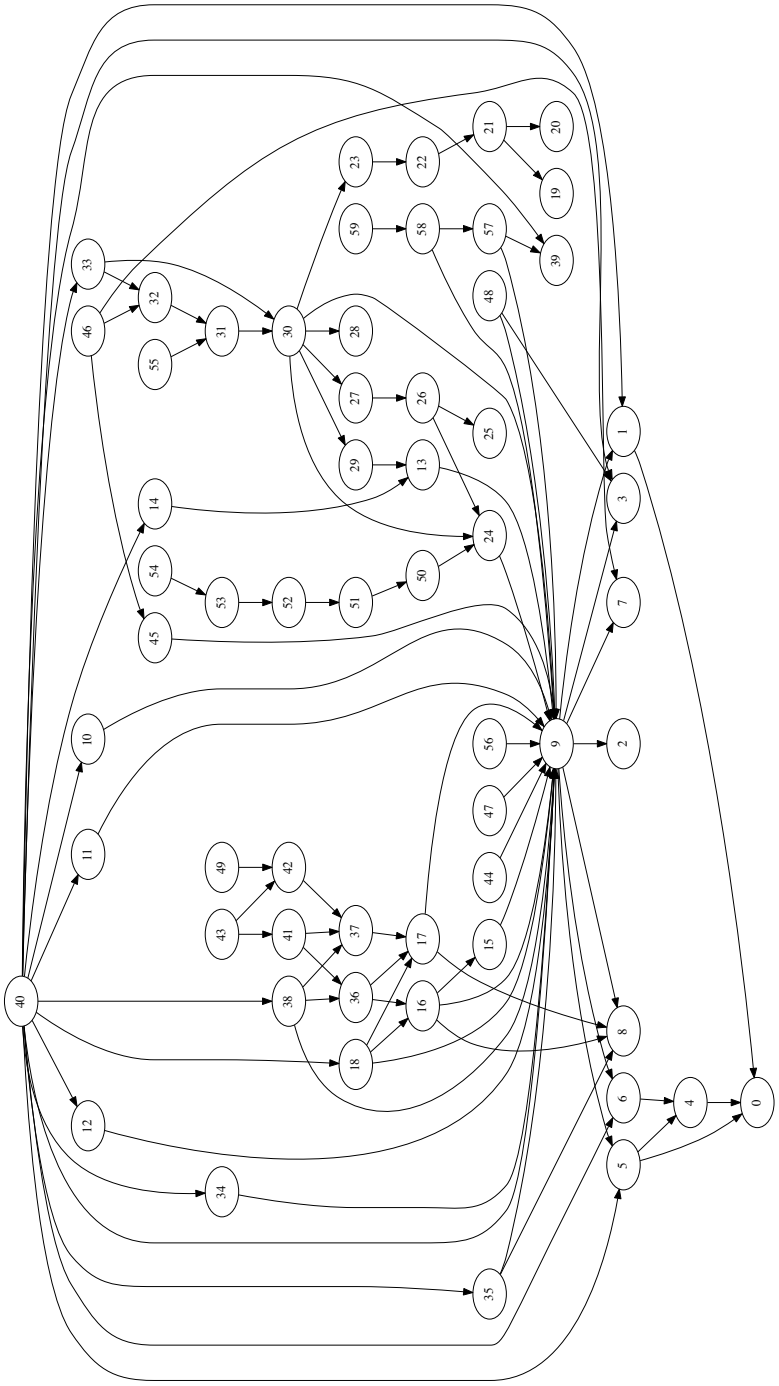


Figure 3.9: The SCC structure of the apoptosis model. Each node represents an SCC in the apoptosis model. The nodes contained in each SCC are listed in Table 3.3. For each pair of a parent SCC and one of its child SCCs, a directed edge is added pointing from the parent SCC to the child SCC.

4

Attractor Detection in Synchronous Networks

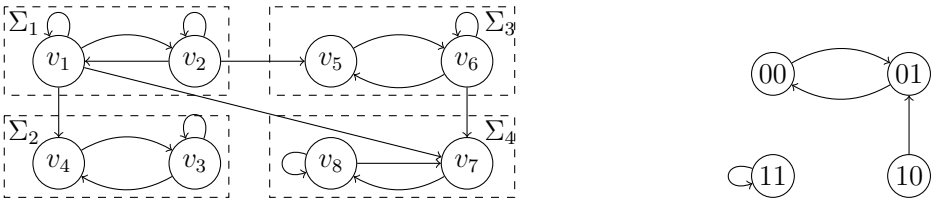
4.1 Introduction

In this chapter, we consider attractor detection in synchronous networks. The networks refer to synchronous BNs and synchronous context-sensitive PBNs and we will only discuss synchronous BNs in the remaining part of this chapter for the reason of simplification.

In Section 3.2, we have reviewed the current status for attractor detection. Identification attractors in large BNs still remains a problem. In this chapter, we propose a new decomposition method for attractor detection in BNs, in particular, in large synchronous BNs, where all the nodes are updated synchronously at each time point. Considering the fact that a few decomposition methods have already been introduced, we explain our new method by showing its main differences from the existing ones. First, our method carefully considers the semantics of synchronous BNs and thus it does not encounter a problem that the method proposed in [GYW+14] does. We explain this in more details in Section 3.3. Second, our new method considers the dependency relation among different sub-networks when detecting attractors of them while the previous method [YQPM16] does not require this. We show with experimental results that this consideration can significantly improve the performance of attractor detection in large BNs. Further, the decomposition method in the previous chapter is designed for asynchronous networks while here we extend it for synchronous networks. As a consequence, the key operation *fulfilment* for the synchronous BNs is completely re-designed with respect to the one for asynchronous BNs in the previous chapter. Last but not least, we provide a proof of the correctness of our new method.

4.2 An SCC-based Decomposition Method

In this section, we describe in details our new SCC-based decomposition method for detecting attractors of large synchronous BNs and we prove its correctness. The method consists of three steps. First, we divide a BN into sub-networks called *blocks* and this step is performed on the *network structure*, instead of the state transition system of the network. Secondly, we detect attractors in each block. Lastly, we recover attractors of the original BN based on attractors of the blocks. The three steps share some similarities with those in Chapter 3; therefore, we will describe this method by comparing it with the one in Chapter 3.



(a) SCC decomposition. (b) Transition graph of block B_1.

Figure 4.1: SCC decomposition and the transition graph of block B_1.

4.2.1 Decomposition of a BN

We use the same definition of blocks as in Definition 3.3.1. But we consider synchronous networks in this chapter and therefore a block is also under the synchronous updating scheme, i.e., all the nodes in the block will be updated synchronously at each given time point no matter this node is undetermined or not.

We now introduce a method to construct blocks, using SCC-based decomposition. The definition of an SCC has been given in Definition 3.3.2. Moreover, we use the same concept of *control node*, *parent block* , *parent SCC*, *ancestor SCC*, and the same way for forming blocks as in Chapter 3. An example for decomposing a BN has been given in Chapter 3. We now give another example, which we will use later in this chapter for explaining our method. Figure 4.1a shows the decomposition of a BN into four SCCs: Σ_1, Σ_2, Σ_3, and Σ_4. In this example, node v_1 is a control node of Σ_2 and Σ_4; node v_2 is a control node of Σ_3; and node v_6 is a control node of Σ_4. The SCC Σ_1 does not have any control node. Σ_2 and its control node v_1 form one block B_2. Σ_1 itself is a block, denoted as B_1, since the SCC it contains does not have any control node.

The state of a block in a synchronous network is slightly different from that of a block in an asynchronous network. In a synchronous network, the state of a block includes not only the values of nodes in this block, but also the values of nodes that in its ancestor blocks. Formally, a state of a block of a synchronous BN is a binary vector $(x_1, \cdots, x_i, x_j, \cdots, x_k)$ where $(x_j, \cdots, x_k)$ are the values of the nodes in the block and $(x_1, \cdots, x_i)$ are the values of the nodes in its ancestor blocks. This difference will lead to the difference of the definition for *fulfilment* in Definition 4.2.1, which is one of the key differences between our decomposition methods for synchronous BNs and asynchronous BNs. As in the previous chapter, we use a number of operations on the states of a BN and its blocks and their definitions are the same. See Definition 3.3.4 and 3.3.5 for details.

4.2.2 Detection of Attractors in a Block

We now consider how to detect attractors in a block. We also consider elementary blocks and non-elementary blocks separately. An elementary block does not depend on any other block while a non-elementary block does. An elementary block is in fact a BN; therefore, the definition of attractors in a BN can be directly taken to the concept of an elementary block. We take the same definition for *preservation of attractors* as in Definition 3.3.6.

(a) Transition graph of Block B_1 in G_1. (b) Transition graph of the "fulfilment".

Figure 4.2: Two transition graphs used in Example 4.2.1 and Example 4.2.2.

Example 4.2.1. *Consider the BN G_1 in Example 2.2.1. Its set of attractors is $\mathcal{A} = \{\{(000), (1*1)\}\}$. Nodes v_1 and v_2 form an elementary block B_1. Since B_1 is an elementary block, it can be viewed as a BN. The transition graph of B_1 is shown in Figure 4.2a. Its set of attractors is $\mathcal{A}^{B_1} = \{\{(00), (1*)\}\}$ (nodes are arranged as v_1, v_2). We have $\pi_{B_1}(\{(000), (1*1)\}) = \{(00), (1*)\} \in \mathcal{A}^{B_1}$, i.e., block B_1 preserves the attractors of G_1.*

With Definition 3.3.6, we have the following lemma and theorem.

Lemma 4.2.1. *Given a BN G and an elementary block B in G, let Φ be the set of attractor states of G and Φ^B be the set of attractor states of B. If B preserves the attractors of G, then $\Phi \subseteq \mathcal{M}_G(\Phi^B)$.*

Proof. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be the set of attractors of G and $\mathcal{A}^B = \{A_1^B, A_2^B, \ldots, A_{m'}^B\}$ be the set of attractors of B. Since B preserves the attractors of G, for any $k \in [1, m]$, there exists a $k' \in [1, m']$ such that $\pi_B(A_k) \subseteq A_{k'}^B$. Therefore, $\pi_B(\Phi) = \cup_{i=1}^{m} \pi_B(A_i) \subseteq \cup_{i=1}^{m'} A_i^B = \Phi^B$. By Definition 3.3.4, we have that $\Phi \subseteq \mathcal{M}_G(\pi_B(\Phi))$. Hence, $\Phi \subseteq \mathcal{M}_G(\Phi^B)$. □

Theorem 4.2.1. *Given a BN G, let B be an elementary block in G. B preserves the attractors of G.*

Proof. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be the set of attractors of G. For any $i \in [1, m]$, let $L = \boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ be a path containing all the states in A_i and let $\boldsymbol{x}_1 = \boldsymbol{x}_k$. In fact, L is an attractor system of A_i. Therefore, $\pi_B(\boldsymbol{x}_1) \to \pi_B(\boldsymbol{x}_2) \to \cdots \to \pi_B(\boldsymbol{x}_k)$ is a path in B. We denote this path as L^B. Given that the choice of the attractor A_i is arbitrary, the claim holds if we can prove that states in the path L^B form an attractor of B. Since $\boldsymbol{x}_1 = \boldsymbol{x}_k$, we have $\pi_B(\boldsymbol{x}_1) = \pi_B(\boldsymbol{x}_k)$. The path L^B is in fact a loop. As B is a synchronous BN, the transitions in B are determined and thus starting from any state in B, no state not in B is reachable. Therefore, the states in the path L^B form an attractor. □

For an elementary block B, the mirror states of its attractor states cover all G's attractor states according to Lemma 4.2.1 and Theorem 4.2.1. Therefore, by searching from the mirror states only instead of the whole state space, we can detect all the attractor states.

We now consider the case of non-elementary blocks. For an SCC Σ_j, if it has no parent SCC, then this SCC can form an elementary block; if it has at least one parent, then it must have an ancestor that has no parent, and all its ancestors $\Omega(\Sigma_j)$ together can form an elementary block, which is also a BN. The SCC-based decomposition will usually result in one or more non-elementary blocks.

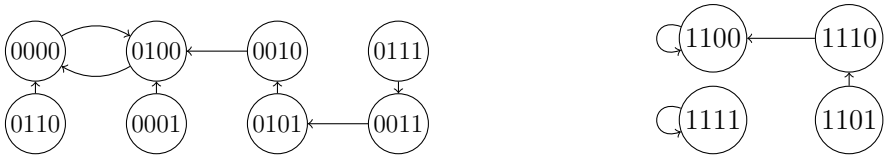
After decomposing a BN into SCCs, there is at least one SCC with no control nodes. Hence, there is at least one elementary block in every BN. Moreover, for each non-elementary block we can construct, by merging all its predecessor blocks, a single parent elementary block. We detect the attractors of the elementary blocks and use the detected attractors to guide the values of the control nodes of their child blocks. The guidance is achieved by considering *fulfilment of the dynamics of a non-elementary block with respect to the attractors of its parent elementary block*, shortly referred to as *fulfilment of a non-elementary block*. In some cases, a fulfilment of a non-elementary block can be easily obtained by assigning new Boolean functions to the control nodes of the block. However, in many cases, such simple assignments are not enough; instead, obtaining a fulfilment of a non-elementary block requires explicitly constructing a transition system of this block corresponding to the considered attractor of the elementary parent block. Since the parent block of a non-elementary block may have more than one attractor, a non-elementary block may have more than one fulfilment.

Definition 4.2.1 (Fulfilment of a non-elementary block)**.** *Let B_i be a non-elementary block formed by merging an SCC with its control nodes. Let nodes $u_1, u_2, \ldots, u_r$ be all the control nodes of B_i which are also contained by its elementary parent block B_j (we can merge B_i's ancestor blocks to form B_j if B_i has more than one parent block or has a non-elementary parent block). Let $A_1^{B_j}, A_2^{B_j}, \ldots, A_t^{B_j}$ be the AS' of B_j. For any $k \in [1, t]$, a* fulfilment *of block B_i with respect to $A_k^{B_j}$ is a state transition system such that*

1. *the state space is the maximal set of states of the merged block $B_{i,j}$ that is crossable with $A_k^{B_j}$;*
2. *the transitions are as follows: for any transition $\boldsymbol{x}^{B_j} \rightarrow \tilde{\boldsymbol{x}}^{B_j}$ in the attractor system of $A_k^{B_j}$, there is a transition $\boldsymbol{x}^{B_i,j} \rightarrow \tilde{\boldsymbol{x}}^{B_{i,j}}$ in the fulfilment such that $\boldsymbol{x}^{B_{i,j}}$ $\mathcal{C}$ $\boldsymbol{x}^{B_j}$ and $\tilde{\boldsymbol{x}}^{B_{i,j}}$ $\mathcal{C}$ $\tilde{\boldsymbol{x}}^{B_j}$; each transition in the fulfilment is caused by the update of all nodes synchronously: the update of non-control nodes of B_i is regulated by the Boolean functions of the nodes and the update of nodes in its parent block B_j is regulated by the transitions of the attractor system of $A_k^{B_j}$;*

In the fulfilment of a non-elementary block, it is not only the control nodes, but all the nodes of its single elementary parent block that are considered. This allows to distinguish the potentially different states in which the values of control nodes are the same. Without this, a state in the state transition graph of the fulfilment may have more than one out-going transition, which is contrary to the fact that the out-going transition for a state in a synchronous network is always determined. Although the definition of attractors can still be applied to such a transition graph, the attractor detection algorithms for synchronous networks, e.g., SAT-based algorithms, may not work any more. Moreover, the meaning of attractors in such a graph are not consistent with the synchronous semantics and therefore the detected "attractors" may not be attractors of the synchronous BN. Note that the decomposition method mentioned in [GYW$^+$14] did not take care of this and therefore produces incorrect results in certain cases. We now give an example to illustrate one of such cases.

Example 4.2.2. *Consider the BN in Example 2.2.1, which can be divided into two blocks: block B_1 with nodes v_1, v_2 and block B_2 with nodes v_2, v_3. The transition graph of B_1 is shown in Figure 4.2a and its attractor is $(00) \rightarrow (10) \rightarrow (11)$. If we do not include the node v_1 when forming the fulfilment of B_2, we will get a transition graph as*



(a) Fulfilment 1 of Example 4.2.3. (b) Fulfilment 2 of Example 4.2.3.

Figure 4.3: Transition graphs of two fulfilments in Example 4.2.3.

shown in Figure 4.2b, which contains two states with two out-going transitions. This is contrary to the synchronous semantics. Moreover, recovering attractors with the attractors in this graph will lead to a non-attractor state of the original BN, i.e., (001).

For asynchronous networks, however, such a distinction is not necessary since the situation of multiple out-going transitions is in consistent with the asynchronous updating semantics. Definition 4.2.1 forms the basis for a key difference between this decomposition method for synchronous BNs and the one for asynchronous BNs proposed in the previous chapter.

Constructing fulfilments for a non-elementary block is a key process for obtaining its attractors. For each fulfilment, the construction process requires the knowledge of all the transitions in the corresponding attractor of its elementary parent block. In Section 4.3, we explain in details how to implement it with BDDs. We now give an example for constructing fulfilments.

Example 4.2.3. *Consider the BN in Figure 4.1a. Its Boolean functions are given as follows:*

$$\begin{cases} f_1 = x_1 \wedge x_2, & f_2 = x_1 \vee \neg x_2, \\ f_3 = \neg x_4 \wedge x_3, & f_4 = x_1 \vee x_3, \\ f_5 = x_2 \wedge x_6, & f_6 = x_5 \wedge x_6, \\ f_7 = (x_1 \vee x_6) \wedge x_8, & f_8 = x_7 \vee x_8. \end{cases} \tag{4.1}$$

The network contains four SCCs $\Sigma_1, \Sigma_2, \Sigma_3$ *and* Σ_4*. For any* Σ_i *(*$i \in [1, 4]$*), we form a block* B_i *by merging* Σ_i *with its control nodes. Block* B_1 *is an elementary block and its transition graph is shown in Figure 4.1b. Block* B_1 *has two attractors, i.e.,* $\{(0*)\}$ *and* $\{(11)\}$*. Regarding the first attractor, block* B_3 *has a fulfilment by setting the nodes* v_1 *and* v_2 *(nodes from its parent block* B_1*) to contain transitions* $\{(00) \to (01), (01) \to (00)\}$*. The transition graph of this fulfilment is shown in Figure 4.3a. Regarding the second attractor, block* B_3 *has a fulfilment by setting nodes* v_1 *and* v_2 *to contain only the transition* $\{(11) \to (11)\}$*. Its transition graph is shown in Figure 4.3b.*

Similar to the case of asynchronous BNs, a fulfilment of a non-elementary block in synchronous BNs also provides a transition system of the block by taking care of the dynamics of the undetermined nodes. Therefore, the attractor definition of fulfilments and non-elementary blocks for asynchronous BNs in Definition 3.3.9 can be directly the taken for synchronous BNs.

With the attractor definition in non-elementary blocks, we can extend Definition 4.2.1 by allowing B_j to be a non-elementary block as well. When forming the fulfilments of a non-elementary block, we only need the attractors of its parent block that contains all its control nodes, no matter whether this parent block is elementary or not. Observe that

using a non-elementary block as a parent block does not change the fact that the attractor states of the parent block contain the values of all the nodes in the current block and all its ancestor blocks.

Computing attractors for non-elementary blocks requires the knowledge of the attractors of its parent blocks. Therefore, we need to order the blocks so that for any block B_i, the attractors of its parent blocks are always detected before it. To do this, we use the concept of a credit as defined in Definition 3.3.10.

4.2.3 Recovery of Attractors for the Original BN

After computing attractors for all the blocks, we need to recover attractors for the original BN, with the help of the following theorem.

Theorem 4.2.2. *Let G be a BN and let B_i be one of its blocks. Denote $\Omega(B_i)$ as the block formed by all B_i's ancestor blocks and denote $\mathcal{X}(B_i)$ as the block formed by merging B_i with $\Omega(B_i)$. $\mathcal{X}(B_i)$ is in fact an elementary block, which is also a BN. The attractors of block B_i are in fact the attractors of $\mathcal{X}(B_i)$.*

Proof. If B_i is an elementary block, B_i is the same as $\mathcal{X}(B_i)$ and the claim holds. We now prove the case where B_i is a non-elementary block. This is equivalent to proving the following two statements: 1) any attractor of B_i is an attractor in $\mathcal{X}(B_i)$; 2) any attractor in $\mathcal{X}(B_i)$ is an attractor of B_i.

Statement 1: Let A^{B_i} be an attractor of B_i and let $\boldsymbol{x}_1^{B_i} \to \boldsymbol{x}_2^{B_i} \to \cdots \to \boldsymbol{x}_k^{B_i}$ be a path L^{B_i} containing all the states in this attractor with $\boldsymbol{x}_1^{B_i} = \boldsymbol{x}_k^{B_i}$. For any state $\boldsymbol{x}_\ell^{B_i}$ in this path, $\boldsymbol{x}_\ell^{B_i}$ is also a state in the block $\mathcal{X}(B_i)$ as $\boldsymbol{x}_\ell^{B_i}$ is a vector formed by the values of nodes in B_i and all its ancestors. In the transition $\boldsymbol{x}_\ell^{B_i} \to \boldsymbol{x}_{\ell+1}^{B_i}$, the nodes in block B_i are updated by their Boolean functions and the nodes that are not in B_i are updated in accordance with the attractor that forms the corresponding fulfilment. Therefore, all the nodes are in fact updated in accordance with their Boolean functions. Hence, such a transition $\boldsymbol{x}_\ell^{B_i} \to \boldsymbol{x}_{\ell+1}^{B_i}$ also exists in the block $\mathcal{X}(B_i)$. Path L^{B_i} is therefore a path in $\mathcal{X}(B_i)$. Since $\boldsymbol{x}_1^{B_i} = \boldsymbol{x}_k^{B_i}$, states in the path L^{B_i}, i.e., states in the attractor A^{B_i} form an attractor in $\mathcal{X}(B_i)$.

Statement 2: This can be proved iteratively. We first consider the case that the parent block of B_i is an elementary block. Let $A^{\mathcal{X}(B_i)}$ be an attractor of $\mathcal{X}(B_i)$ and $\boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ be a path $L^{\mathcal{X}(B_i)}$ containing all the states in this attractor and $\boldsymbol{x}_1 = \boldsymbol{x}_k$. Denote B_i's parent block as B_j. Since B_j is an elementary block, $\pi_{B_j}(\boldsymbol{x}_1) \to \pi_{B_j}(\boldsymbol{x}_2) \to \cdots \to \pi_{B_j}(\boldsymbol{x}_k)$ is a path in B_j and $\pi_{B_j}(\boldsymbol{x}_1) = \pi_{B_j}(\boldsymbol{x}_k)$. Therefore, states in this path form an attractor, denoted as A^{B_j}. We consider the fulfilment of block B_i with respect to A^{B_j}. For any $\ell \in [1, k]$, state $\boldsymbol{x}_\ell$ in the path $L^{\mathcal{X}(B_i)}$ is crossable with $\pi_{B_j}(\boldsymbol{x}_\ell)$. Therefore, $\boldsymbol{x}_\ell$ is also a state in the fulfilment. Hence, $A^{\mathcal{X}(B_i)}$ is also an attractor in the fulfilment and the claim holds for this case. We now consider the case where B_j (the parent block of B_i) is not an elementary block and the parent block of B_j is an elementary block. We have that any attractor in $\mathcal{X}(B_j)$ is an attractor of B_j. Let $A^{\mathcal{X}(B_i)}$ be an attractor of $\mathcal{X}(B_i)$ and $\boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \cdots \to \boldsymbol{x}_k$ be a path $L^{\mathcal{X}(B_i)}$ containning all the states in this attractor and $\boldsymbol{x}_1 = \boldsymbol{x}_k$. Since $\mathcal{X}(B_j)$ is an elementary block, we have that $\pi_{\mathcal{X}(B_j)}(\boldsymbol{x}_1) \to \pi_{\mathcal{X}(B_j)}(\boldsymbol{x}_2) \to \cdots \to \pi_{\mathcal{X}(B_j)}(\boldsymbol{x}_k)$ is a path in $\mathcal{X}(B_j)$ and $\pi_{\mathcal{X}(B_j)}(\boldsymbol{x}_1) = \pi_{\mathcal{X}(B_j)}(\boldsymbol{x}_k)$. Therefore, states in this path form an attractor of $\mathcal{X}(B_j)$, which is also an attractor of

B_j, denoted as A^{B_j}. Regarding A^{B_j}, block B_i has a fulfilment. For any $\ell \in [1, k]$, state $\boldsymbol{x}_\ell$ in the path $L^{\mathcal{X}(B_i)}$ is crossable with $\pi_{B_j}(\boldsymbol{x}_\ell)$ (which is also $\pi_{\mathcal{X}(B_j)}(\boldsymbol{x}_\ell)$). Therefore, $\boldsymbol{x}_\ell$ is also a state in the fulfilment. Hence, $A^{\mathcal{X}(B_i)}$ is also an attractor in the fulfilment. Hence, the claim holds. □

Theorem 4.2.3. *Given a BN G, where B_i and B_j are its two blocks, let $\mathcal{A}^{B_i}$ and $\mathcal{A}^{B_j}$ be the set of attractors for B_i and B_j, respectively. Let $B_{i,j}$ be the block got by merging the nodes in B_i and B_j. Denote the set of all attractor states of $B_{i,j}$ as $S^{B_{i,j}}$. If B_i and B_j are both elementary blocks, $\mathcal{A}^{B_i}$ $\mathcal{C}$ $\mathcal{A}^{B_j}$ and $\cup_{A\in\Pi(\mathcal{A}^{B_i},\mathcal{A}^{B_j})}A = S^{B_{i,j}}$.*

Proof. Let B_i and B_j be two elementary blocks of G. If B_i and B_j do not have common nodes, then it holds by definition that $\mathcal{A}^{B_i}$ $\mathcal{C}$ $\mathcal{A}^{B_j}$. If they have common nodes, their common nodes must form an elementary block. Denote this block as B_c. For any attractor $A^{B_i} \in \mathcal{A}^{B_i}$, $\pi_{B_c}(A^{B_i})$ is an attractor in B_c and A^{B_i} $\mathcal{C}$ $\pi_{B_c}(A^{B_i})$. Denote the nodes in B_i but not in B_c as N. The nodes in N and their control nodes in B_c (if they have) form a block B^N. We have the following two claims. Claim I: For any attractor A^{B_c} of B_c, there exists an attractor A^{B_N} in B^N such that A^{B_c} $\mathcal{C}$ A^{B_N}. Claim II: For any attractor A^{B_c} of B_c, there exists an attractor $A^{B_i} \in \mathcal{A}^{B_i}$ such that A^{B_i} $\mathcal{C}$ A^{B_c}. We first prove Claim I. If block B^N does not share nodes with B_c, Claim I holds according to the definition of crossability. If block B^N shares nodes with B_c, the fulfilments of B^N is constructed based on the attractors of B_c. According to Definition 4.2.1, for any attractor A^{B_c} of B_c, a fulfilment will be constructed and the attractor of this fulfilment is crossable with A^{B_c}, thus Claim I holds in this case as well. We continue to prove Claim II. Denote the length of attractor A^{B_c} as ℓ^{B_c} and the length of attractor A^{B_N} as ℓ^{B_N}. Let $\boldsymbol{x}^{B_c}$ be a state in A^{B_c} and $\boldsymbol{x}^{B_N}$ be a state in A^{B_N}. Let $\boldsymbol{x}_1 = \Pi(\boldsymbol{x}^{B_c}, \boldsymbol{x}^{B_N})$. Let L be a path starting from state $\boldsymbol{x}_1$ and of length $k = lcm(\ell^{B_c}, \ell^{B_N}) + 1$, where lcm means the lowest common multiple. Since $\boldsymbol{x}^{B_c}$ is an attractor state, $\pi_{B_c}(\boldsymbol{x}_k) = \boldsymbol{x}^{B_c}$. Similarly, $\pi_{B_N}(\boldsymbol{x}_k) = \boldsymbol{x}^{B_N}$. Hence, $\boldsymbol{x}_k = \Pi(\pi_{B_c}(\boldsymbol{x}_k), \pi_{B_N}(\boldsymbol{x}_k)) = \boldsymbol{x}_1$. Therefore, states in L form an attractor of B_i. Hence the claim holds. Similarly, it holds that for any attractor A^{B_c} of B_c, there exists an attractor $A^{B_j} \in \mathcal{A}^{B_j}$ such that A^{B_j} $\mathcal{C}$ A^{B_c}. Now, let A^{B_i} be an attractor in $\mathcal{A}^{B_i}$. Then, $\pi_{B_c}(A^{B_i})$ is an attractor in B_c and by the above, there exists an attractor $A^{B_j} \in \mathcal{A}^{B_j}$ such that $\pi_{B_c}(A^{B_i})$ $\mathcal{C}$ A^{B_j}. Thus, A^{B_i} $\mathcal{C}$ A^{B_j}. By similar argument, for any $A^{B_j} \in \mathcal{A}^{B_j}$ there exists $A^{B_i} \in \mathcal{A}^{B_i}$ such that A^{B_j} $\mathcal{C}$ A^{B_i}. In consequence, $\mathcal{A}^{B_i}$ $\mathcal{C}$ $\mathcal{A}^{B_j}$.

We now prove that $\cup_{A\in\Pi(\mathcal{A}^{B_i},\mathcal{A}^{B_j})}A = S^{B_{i,j}}$. Denote $S = \cup_{A\in\Pi(\mathcal{A}^{B_i},\mathcal{A}^{B_j})}A$. This is equivalent to showing the following two statements: 1) for any state $\boldsymbol{s} \in S$, $\boldsymbol{s}$ is in $S^{B_{i,j}}$; 2) any state in $S^{B_{i,j}}$ is contained in S. We prove them one by one.

Statement 1: Let A be any set of states in $\Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})$. Then there exists $A^{B_i} \in \mathcal{A}^{B_i}$ and $A^{B_j} \in \mathcal{A}^{B_j}$ such that $A = \Pi(A^{B_i}, A^{B_j})$ and A^{B_i} $\mathcal{C}$ A^{B_j}. Given the choice of A is arbitrary, it is enough to show that any $\boldsymbol{s} \in A$ is an attractor state of $B_{i,j}$. It holds that $\boldsymbol{s} = \Pi(\pi_{B_i}(\boldsymbol{s}), \pi_{B_j}(\boldsymbol{s}))$, where $\pi_{B_i}(\boldsymbol{s}) \in A^{B_i}$ and $\pi_{B_j}(\boldsymbol{s}) \in A^{B_j}$. Let l^{B_i} be the attractor length of A^{B_i} and l^{B_j} be the attractor length of A^{B_j}. Further, let $L = \boldsymbol{s}_1 \to \boldsymbol{s}_2 \to \cdots \to \boldsymbol{s}_k$ be a path starting from state $\boldsymbol{s}$, i.e., $\boldsymbol{s}_1 = \boldsymbol{s}$, with $k = lcm(l^{B_i}, l^{B_j}) + 1$. Since both B_i and B_j are elementary blocks, it holds that $\pi_{B_i}(\boldsymbol{s}_1) \to \pi_{B_i}(\boldsymbol{s}_2) \to \cdots \to \pi_{B_i}(\boldsymbol{s}_k)$ is a path in B_i and $\pi_{B_j}(\boldsymbol{s}_1) \to \pi_{B_j}(\boldsymbol{s}_2) \to \cdots \to \pi_{B_j}(\boldsymbol{s}_k)$ is a path in B_j. Since $\pi_{B_i}(\boldsymbol{s}_1) = \pi_{B_i}(\boldsymbol{s})$, we have $\pi_{B_i}(\boldsymbol{s}_k) = \pi_{B_i}(\boldsymbol{s})$. Similarly, we have $\pi_{B_j}(\boldsymbol{s}_k) = \pi_{B_j}(\boldsymbol{s})$. Then, $\boldsymbol{s}_k = \Pi(\pi_{B_i}(\boldsymbol{s}_k), \pi_{B_j}(\boldsymbol{s}_k)) = \Pi(\pi_{B_i}(\boldsymbol{s}), \pi_{B_j}(\boldsymbol{s})) = \boldsymbol{s}$. In consequence, $\boldsymbol{s}_1 = \boldsymbol{s} = \boldsymbol{s}_k$ and the states in L form an attractor of $B_{i,j}$ with $\boldsymbol{s}$ being one of its states.

Statement 2: Let $\boldsymbol{s}$ be a state in $S^{B_{i,j}}$ and let A be the attractor of $B_{i,j}$ containing state $\boldsymbol{s}$. Let $L = \boldsymbol{s} \to \boldsymbol{s}_1 \to \boldsymbol{s}_2 \to \cdots \to \boldsymbol{s}_k \to \boldsymbol{s}$ be a path starting end ending with $\boldsymbol{s}$. It holds that $\pi_{B_i}(\boldsymbol{s}) \to \pi_{B_i}(\boldsymbol{s}_1) \to \pi_{B_i}(\boldsymbol{s}_2) \to \cdots \to \pi_{B_i}(\boldsymbol{s}_k) \to \pi_{B_i}(\boldsymbol{s})$ is an attractor system in the elementary block B_i. Let us denote the attractor's set of states as A^{B_i}. We have that $\pi_{B_i}(\boldsymbol{s}) \in A^{B_i}$. Similarly, $\pi_{B_j}(\boldsymbol{s})$ belongs to an attractor of B_j, denoted as A^{B_j}. Therefore, $\boldsymbol{s} = \Pi(\pi_{B_i}(\boldsymbol{s}), \pi_{B_j}(\boldsymbol{s})) \in \Pi(A^{B_i}, A^{B_j}) \subseteq S$. Given the arbitrary choice of $\boldsymbol{s}$, the claim holds. □

The above developed theoretical background with Theorem 4.2.2 and Theorem 4.2.3 being its core result, allows us to design a new decomposition-based approach towards detection of attractors in large synchronous BNs. The idea is as follows. We divide a BN into blocks according to the detected SCCs. We sort the blocks in ascending order based on their credits and detect attractors of the ordered blocks one by one in an iterative way. We start from detecting attractors of elementary blocks (credit 0), and continue to detect blocks with higher credits after constructing their fulfilments. According to Theorem 4.2.2, by detecting the attractors of a block, we in fact obtain the attractors of the block formed by the current block and all its ancestor blocks. Hence, after the attractors of all the blocks have been detected, either we have obtained the attractors of the original BN or we have obtained the attractors of several elementary blocks of this BN. According to Theorem 4.2.3, we can perform a cross operation for any two elementary blocks (credits 0) to recover the attractor states of the two merged blocks. The resulting merged block will form a new elementary block, i.e., one with credit 0. The attractors can be easily identified from the set of attractor states. By iteratively performing the cross operation until a single elementary block containing all the nodes of the BN is obtained, we can recover the attractors of the original BN. The details of this new algorithm are discussed in the next section. In addition, we have the following corollary which will be used in the next section.

Corollary 4.2.1. *Given a BN G, where B_i and B_j are its two blocks, let $\mathcal{A}^{B_i}$ and $\mathcal{A}^{B_j}$ be the set of attractors for B_i and B_j, respectively. Let $B_{i,j}$ be the block got by merging the nodes in B_i and B_j. Denote the set of attractor states of $B_{i,j}$ as $S^{B_{i,j}}$. It holds that $\mathcal{A}^{B_i}$ $\mathcal{C}$ $\mathcal{A}^{B_j}$ and $\cup_{S \in \Pi(\mathcal{A}^{B_i}, \mathcal{A}^{B_j})} S = S^{B_{i,j}}$.*

Proof. If B_i and B_j are both elementary blocks, the claim holds according to Theorem 4.2.3. We now prove the general cases. Denote $\Omega(B_i)$ the block formed by all B_i's ancestor blocks and denote $\mathcal{X}(B_i)$ the block formed with B_i and $\Omega(B_i)$. Denote $\Omega(B_j)$ the block formed by all B_j's ancestor blocks and denote $\mathcal{X}(B_j)$ the block formed with B_j and $\Omega(B_j)$. According to Theorem 4.2.2, the attractors of block B_i are in fact the attractors of the elementary block $\mathcal{X}(B_i)$ and the attractors of block B_j are in fact the attractors of the elementary block $\mathcal{X}(B_j)$. Since both $\mathcal{X}(B_i)$ and $\mathcal{X}(B_j)$ are elementary blocks, the claim holds by Theorem 4.2.3. □

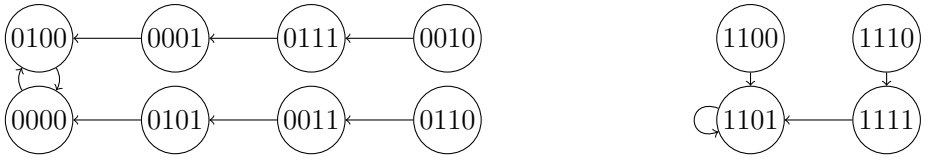
4.3 A BDD-based Implementation

We describe the SCC-based attractor detection method in Algorithm 3. This algorithm takes a BN G and its corresponding transition system $\mathcal{T}$ as inputs, and outputs the set of attractors of G. In this algorithm, we denote by DETECT($\mathcal{T}$) a basic function for detecting attractors of a given transition system $\mathcal{T}$. Lines 24-27 of this algorithm describe

Algorithm 3 SCC-based decomposition algorithm

procedure SCC_DETECT(G, $\mathcal{T}$)
  $B :=$ FORM_BLOCK(G); $\mathcal{A} := \emptyset$; $B_a := \emptyset$; $k :=$ size of B;
  initialise dictionary $\mathcal{A}^\ell$; *//$\mathcal{A}^\ell$ is stores the set of attractors for each block*
  for $i := 1; i <= k; i++$ **do**
    if B_i is an elementary block **then**
      $\mathcal{T}^{B_i} :=$ transition system converted from B_i; *//see Section 3.4.1*
      $\mathcal{A}_i :=$ DETECT($\mathcal{T}^{B_i}$); $\mathcal{A}^\ell.add((B_i, \mathcal{A}_i))$; *//for more details*
    else $\mathcal{A}_i := \emptyset$;
      if B_i^p is the only parent block of B_i **then**
        $\mathcal{A}_i^p := \mathcal{A}^\ell.getAtt(B_i^p)$; *//obtain attractors of B_i^p*
      else $B^p := \{B_1^p, B_2^p, \ldots, B_m^p\}$ be the parent blocks of
        B_i (ascending ordered);
        $B_c := B_1^p$; *//B^p is ordered based on credits*
        for $j := 2; j <= m; j++$ **do**
          $B_{c,j} :=$ a new block containing nodes in B_c and B_j^p;
          if $(\mathcal{A}_i^p := \mathcal{A}^\ell.getAtt(B_{c,j})) == \emptyset$ **then**
            $A := \Pi(\mathcal{A}^\ell.getAtt(B_c), \mathcal{A}^\ell.getAtt(B_j))$; $\mathcal{A}_i^p := D(A)$;
            //$D(A)$ returns all the attractors from attractor states sets A
            $\mathcal{A}^\ell.add(B_{c,j}, \mathcal{A}_i^p)$;
          end if
          $B_c := B_{c,j}$;
        end for
      end if
      for $A \in \mathcal{A}_i^p$ **do**
        $\mathcal{T}^{B_i}(A) := \langle S^{B_i}(A), T^{B_i}(A) \rangle$; *//obtain the fulfilment of B_i with A*
        $\mathcal{A}_i := \mathcal{A}_i \cup$ DETECT($\mathcal{T}^{B_i}(A)$);
      end for
      $\mathcal{A}^\ell.add((B_i, \mathcal{A}_i))$; *//the add operation will not add duplicated elements*
      $\mathcal{A}^\ell.add((B_{i,ancestors}, \mathcal{A}_i))$; *//$B_{i,ancestors}$ is B_i and all its ancestor blocks*
      for any $B^p \in \{B_1^p, B_2^p, \ldots, B_m^p\}$ **do** *//$B_1^p, B_2^p, \ldots, B_m^p$ are parent blocks*
        $\mathcal{A}^\ell.add((B_{i,p}, \mathcal{A}_i))$; *//of B_i*
      end for
    end if
  end for
  for $B_i \in B$ and B_i has no child block **do**
    $\mathcal{A} = D(\Pi(\mathcal{A}^\ell.get(B_i), \mathcal{A}))$;
  end for
  return $\mathcal{A}$.
end procedure

procedure FORM_BLOCK(G)
  decompose G into SCCs and form blocks with SCCs and their control nodes;
  sort the blocks in ascending order according to their credits;
  $B := (B_1, \ldots, B_k)$;
  return B. *//B is the list of blocks after ordering*
end procedure



(a) Fulfilment 1 of B_2. (b) Fulfilment 2 of B_2.

Figure 4.4: Two fulfilments used in Example 4.3.1.

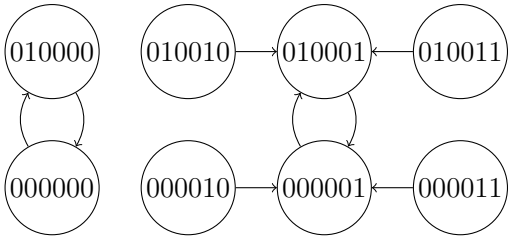
the process for detecting attractors of a non-elementary block. The algorithm detects the attractors of all the fulfilments of the non-elementary block and performs the union operation of the detected attractors. For this, if the non-elementary block has only one parent block, its attractors are already computed as the blocks are considered in ascending order with respect to their credits by the main **for** loop in line 4. Otherwise, all the parent blocks are considered in the **for** loop in lines 14-22. By iteratively applying the cross operation in line 17 to the attractor sets of the ancestor blocks in ascending order, the attractor states of a new block formed by merging all the parent blocks are computed as assured by Corollary 4.2.1. The attractors are then identified from the attractor states with one more operation. The correctness of the algorithm is stated as Theorem 4.3.1.

Theorem 4.3.1. *Algorithm 3 correctly identifies the set of attractors of a given BN G.*

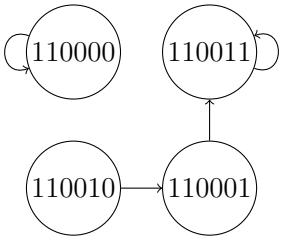
Proof. Algorithm 3 divides a BN into SCC blocks and detects attractors of each block. Lines 5 to 33 describe the process for detecting attractors of a block. The algorithm distinguishes between two different types of blocks. The first type is an elementary block. Since it is in fact a BN, the attractors of this type of block are directly detected via the basic attractor detection function $\textsc{Detect}(\mathcal{T})$. The second type is a non-elementary block. The algorithm constructs the fulfilments of this type of block, detects attractors of each fulfilment and merges them as the attractors of the block. The algorithm takes special care of blocks with more than one parent block. It merges all the parent blocks of such a block to form a single parent block. Since the parent blocks are considered in ascending order operations with respect to their credits, the two operations in Line 17 will iteratively recover the attractors of the parent block according to Corollary 4.2.1. After all attractors of the blocks have been detected, the algorithm performs several cross and detect operations to recover the attractors of the original BN in Lines 35 to 37. Since the attractors of all blocks are considered, it will finally recover the attractors of the BN. □

We continue to illustrate in Example 4.3.1 how Algorithm 3 detects attractors.

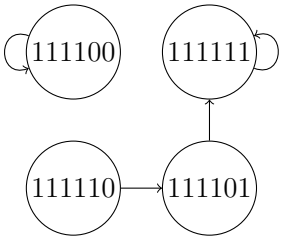
Example 4.3.1. *Consider the BN shown in Example 4.2.3 and its four blocks. Block B_1 is an elementary block and it has two attractors, i.e., $\mathcal{A}_1 = \{\{(0*)\}, \{(11)\}\}$. To detect the attractors of block B_2, we first form fulfilments of B_2 with the attractors of its parent block B_1. B_1 has two attractors so there are two fulfilments for B_2. The transition graphs of the two fulfilments are shown in Figures 4.4a and 4.4b. We get two attractors for block B_2, i.e., $\mathcal{A}_2 = \{\{(0*00)\}, \{(1101)\}\}$. Those two attractors are also attractors for the merged block $B_{1,2}$, i.e., $\mathcal{A}_{1,2} = \mathcal{A}_2$. In Example 4.2.3, we have shown the two fulfilments of B_3 regarding the two attractors of B_1. Clearly, B_3 has two attractors, i.e., $\mathcal{A}_3 = \{\{(0*00)\}, \{(1100)\}, \{(1111)\}\}$. B_4 has two parent blocks. Therefore, we need to merge the two parent blocks to form a single parent block. Since the attractors of*



(a) The first fulfilment of B_4



(b) The second fulfilment of B_4



(c) The third fulfilment of B_4

Figure 4.5: Transition graphs of the three fulfilments for block B_4.

the merged block $B_{1,3}$ are the same as B_3, we directly obtain the attractors of $B_{1,3}$, i.e., $\mathcal{A}_{1,3} = \mathcal{A}_3 = \{\{(0*00)\}, \{(1100)\}, \{(1111)\}\}$. There are three attractors so there will be three fulfilments for block B_4. The transition graph of the three fulfilments are shown in Figure 4.5. From the transition graphs, we easily get the attractors of B_4, i.e., $\mathcal{A}_4 = \{\{(0*0000)\}, \{(0*0001)\}, \{(110000)\}, \{(110011)\}, \{(111100)\}, \{(111111)\}\}$. Now the attractors for all the blocks have been detected. We can now obtain all the attractors of the BN by several cross operations. We start from the block with the largest credit, i.e., block B_4. The attractors of B_4 in fact cover blocks B_1, B_3 and B_4. The remaining block is B_2. We perform a cross operation on $\mathcal{A}_2$ and $\mathcal{A}_4$ and based on the obtained result we detect the attractors of the BN, i.e., $\mathcal{A} = D(\Pi(\mathcal{A}_2, \mathcal{A}_4) = \{\{(0*000000)\}, \{(0*000001)\}, \{(11010011)\}, \{(11010000)\}, \{(11011111)\}, \{(11011100)\}\}$.

4.3.1 An Optimisation

It often happens that a BN contains many *leaf* nodes that do not have any child node. Each of the leaf nodes will be treated as an SCC in our algorithm and it is not worth the effort to process an SCC with only one leaf node. Therefore, we treat leaf nodes in a special way. Formally, leaf nodes are recursively defined as follows.

Definition 4.3.1. *A node in a BN is a* leaf node *(or* leaf *for short) if and only if it is not the only node in the BN and either (1) it has no child nodes except for itself or (2) it has no other children after iteratively removing all its child nodes which are leaf nodes.*

Algorithm 4 outlines the leaf-based decomposition approach for attractor detection. We now show that Algorithm 4 can identify all attractor states of a given BN.

Theorem 4.3.2. *Algorithm 4 correctly identifies all the attractor states of a given BN G.*

Proof. Block B formed in Line 2 is an elementary block. Algorithm 4 finds the attractor states of B, denoted Φ^B in Line 3. Since B is an elementary block, it preserves the attractors of G by Theorem 4.2.1 and thus, by Lemma 4.2.1, it holds that $\mathcal{M}_G(\Phi^B)$

Algorithm 4 Leaf-based optimisation

1: **procedure** LEAF_DETECT(G)
2: form an elementary block B by removing all the leaves of G;
3: $\mathcal{A}^B :=$ SCC_DETECT (B); $\Phi^B := \cup_{A^B \in \mathcal{A}^B} A^B$; *//detect attractors of* B
4: $\mathcal{T} :=$ transition system of G with state space restricted to $\mathcal{M}_G(\Phi^B)$;
5: $\mathcal{A} :=$ DETECT $(\mathcal{T})$;
6: **return** $\mathcal{A}$.
7: **end procedure**

contains all the attractor states of G. Therefore, the basic attractor detection function DETECT applied in Line 5 to the transition system of G restricted to the states $\mathcal{M}_G(\Phi^B)$ identifies all the attractor states of G. □

4.4 Experimental Results

We have implemented the decomposition algorithm presented in Section 4.3 in the model checker MCMAS [LQR15]. In this section, we demonstrate the efficiency of our method by comparing our method with the state-of-the-art decomposition method mentioned in [YQPM16] which is also based on BDD implementation. We generate 33 random BN models with different number of nodes using the tool ASSA-PBN [MPY15, MPY16b] and compare the performance of the two methods on these 33 models. All the experiments are conducted on a computer with an Intel Xeon W3520@2.67GHz CPU and 12GB memory.

We name our proposed decomposition method as M_1 and the one in [YQPM16] as M_2. There are two possible implementations of the DETECT function used in Algorithm 3 as mentioned in [YQPM16]: monolithic and enumerative. We use the monolithic one which is shown to be more suitable for small networks as the decomposed sub-networks are relatively small. Since the method in [YQPM16] uses similar leaf reduction technique, we make comparisons on both the original models and the models whose leaves are removed in order to eliminate the influence of leaf nodes. We set the expiration time to 3 hours. Before removing leaf nodes, there are 11 cases that both methods fail to process. Among the other 22 cases, our method is faster than M_2 in 16 cases. After removing leaf nodes, there are 5 cases that both methods fail to process. Among the other 28 cases, our method is faster than M_2 in 25 cases. We demonstrate the results for 7 models in Table 4.1 and the remaining result can be found in [MPQY]. Since our method considers the dependency relation between different blocks, the attractors of all the blocks need to be computed; while method M_2 can ignore the blocks with only leaf nodes. Therefore, the performance of our method is more affected by the leaf nodes. This is why our method is not significantly faster than M_2 when leaf nodes are not removed. Notably, after eliminating the influence of leaf nodes, our method is significantly faster than M_2. The "–" in Table 4.1 means the method fails to process the model within 3 hours. The speedup is therefore not applicable (N/A) for this result. The speedup is computed as t_{M_2}/t_{M_1}, where t_{M_1} is the time cost for M_1 and t_{M_2} is the time cost for M_2. All the time shown in Table 4.1 is in seconds. In general, we obtain a larger speedup when the number of attractors is relatively small. This is due to that our method takes the attractors of the parent block into account when forming a fulfilment of a non-

model ID	# nodes	# non-leaves	# attractors	original models $t_{M_2}[s]$	original models $t_{M_1}[s]$	original models Speedup	models with leaves removed $t_{M_2}[s]$	models with leaves removed $t_{M_1}[s]$	models with leaves removed Speedup
1	100	7	32	4.56	0.86	5.3	0.58	0.02	29.0
2	120	9	1	18.13	0.95	19.1	1.10	0.04	27.5
3	150	19	2	201.22	1.66	121.2	0.74	0.02	37.0
4	200	6	16	268.69	7.04	38.2	0.97	0.02	48.5
5	250	25	12	533.57	11.16	47.8	0.90	0.04	22.5
6	300	88	1	–	–	N/A	238.96	65.33	3.7
7	450	43	8	–	60.82	N/A	3704.33	0.17	21790.2

Table 4.1: Selected results for the performance comparison of methods M_1 and M_2.

elementary block and the number of fulfilments increases with the number of attractors. Summarising, our new method shows a significant improvement on the state-of-the-art decomposition method.

4.5 Conclusion and Future Work

We have introduced a new SCC-based decomposition method for attractor detection of large synchronous BNs. Although our decomposition method shares similar ideas on how to decompose a large network with existing decomposition methods, our method differs from them in the key process and has significant advantages.

First, our method is designed for synchronous BNs, as a consequence the key process for constructing fulfilments in our method is totally different from the one for asynchronous BNs as described in the previous chapter, which is designed for asynchronous networks. Secondly, our method considers the dependency relation among the sub-networks. The method in [YQPM16] does not rely on this relation and only takes the detected attractors in sub-networks to restrict the initial states when recovering the attractors for the original network. In this way, the decomposition method in [YQPM16] potentially cannot scale up very well for large networks, as it still requires a BDD encoding of the transition relation of the whole network. Experimental results show that our method is significantly faster than the one proposed in [YQPM16]. Next, we have also shown that the method proposed in [GYW$^+$14] cannot compute correct results in certain cases. Finally, we provide a proof of the correctness of our method in this work.

Our current implementation is based on BDDs. One future work is to use SAT-solvers to implement the DETECT function as SAT-based methods are normally more efficient in terms of attractor detection for synchronous BNs [DT11].

Part II

Steady-state Computation

5

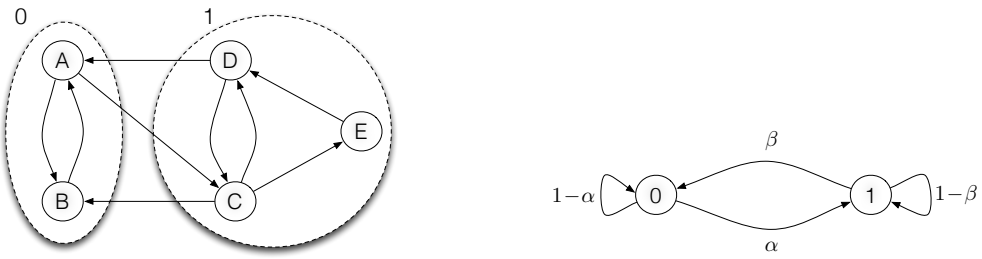
Efficient Steady-state Computation

Starting from this chapter, we focus on the second research problem, i.e., how to compute steady-state probabilities of a PBN efficiently, especially for the large ones. For small networks with a few tens of nodes, numerical methods like the GaussSeidel method, can quickly solve the problem of steady-state computation. However, these numerical methods are prohibited when it comes to large networks with hundreds of nodes due to the huge state-space. Monte Carlo simulation methods are in fact the only feasible ones. Shmulevich et al. [SGH$^+$03] proposed to use the two-state Markov chain approach for analysing the steady-state dynamics of PBNs in 2003. However, since then it has not been widely applied. In this chapter, we revive the two-state Markov chain approach by demonstrating its usefulness for approximating steady-state probabilities of large PBNs. The rest of this chapter is arranged as follows. In Section 5.1, we introduce the theory of the two-state Markov chain approach. We identify an initialisation problem which may lead to a biased result of the two-state Markov chain approach and propose several heuristics for avoiding it in Section 5.2. Then we evaluate the performance of the two-state Markov chain approach by comparing it with another statistical method called Skart in Section 7.2 and provide a case study of this method on a real-life biological network in Section 5.4. Lastly, we conclude this chapter and provide the deviations of formulas used in this chapter.

5.1 The Two-state Markov Chain Approach

The two-state Markov chain approach [RL92] is a method for estimating the steady-state probability of a subset of states of a DTMC. In this approach the state space of an arbitrary DTMC is split into two disjoint sets, referred to as meta states. One of the meta states, numbered 1, is the subset of interest and the other, numbered 0, is its complement. The steady-state probability of meta state 1, denoted π_1, can be estimated by performing simulations of the original Markov chain. For this purpose a two-state Markov chain abstraction of the original DTMC is considered. Let $\{Z_t\}_{t\geqslant 0}$ be a family of binary random variables, where Z_t is the number of the meta state the original Markov chain is in at time t. $\{Z_t\}_{t\geqslant 0}$ is a binary (0-1) stochastic process, but in general it is not a Markov chain. However, as argued in [RL92], a reasonable assumption is that the dependency in $\{Z_t\}_{t\geqslant 0}$ falls off rapidly with lag. Therefore, a new process $\{Z_t^{(k)}\}_{t\geqslant 0}$, where $Z_t^{(k)} = Z_{1+(t-1)k}$, will be approximately a first-order Markov chain for k large enough. A procedure for determining appropriate k is given in [RL92]. The first-order chain consists of the two meta states with transition probabilities α and β between them. See Figure 5.1 for an illustration of the construction of this abstraction.

The steady-state probability estimate $\hat{\pi}_1$ is computed from a simulated trajectory of the



(a) Original DTMC (b) Two-state DTMC

Figure 5.1: Conceptual illustration of the idea of the two-state Markov chain construction. (a) The state space of the original discrete-time Markov chain is split into two meta states: states A and B form meta state 0, while states D, C, and E form meta state 1. The split of the state space into meta states is marked with dashed ellipses. (b) Projecting the behaviour of the original chain on the two meta states results in a binary (0-1) stochastic process. After potential subsampling, it can be approximated as a first-order, two-state Markov chain with the transition probabilities α and β set appropriately.

original DTMC. The key point is to determine the optimal length of the trajectory. Two requirements are imposed. First, the abstraction of the DTMC, i.e., the two-state Markov chain, should converge close to its steady-state distribution $\pi = [\pi_0\ \pi_1]$. Formally, t satisfying $|\mathbb{P}[Z_t^{(k)} = i \,|\, Z_0^{(k)} = j] - \pi_i| < \epsilon$ for a given $\epsilon > 0$ and all $i, j \in \{0, 1\}$ needs to be determined. t is the so-called 'burn-in' period and determines the part of the trajectory of the two-state Markov chain that needs to be discarded. Second, the estimate $\hat{\pi_1}$ is required to satisfy $\mathbb{P}[\pi_1 - r \leqslant \hat{\pi_1} \leqslant \pi_1 + r] \geqslant s$, where r is the required precision and s is a specified confidence level. This condition is used to determine the length of the second part of the trajectory used to compute $\hat{\pi_1}$, i.e., the sample size. Now, the total required trajectory length of the original DTMC is then given by $M + N$, where $M = 1 + (t-1)k$ and $N = 1 + (\lceil n(\alpha, \beta) \rceil - 1)k$, where $t = \lceil m(\alpha, \beta) \rceil$. The functions m and n depend on the transitions probabilities α and β and are given by

$$m(\alpha, \beta) = \frac{\log\left(\frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)}\right)}{\log\left(|1 - \alpha - \beta|\right)} \tag{5.1}$$

and

$$n(\alpha, \beta) = \frac{\alpha\beta(2 - \alpha - \beta)}{(\alpha + \beta)^3} \frac{\left(\Phi^{-1}(\frac{1}{2}(1 + s))\right)^2}{r^2}, \tag{5.2}$$

where Φ^{-1} is the inverse of the standard normal cumulative distribution function. For the completeness of the presentation, the detailed derivations of the expressions for m and n are given in the Sections 5.6.1 and 5.6.2.

Since α and β are unknown, they need to be estimated. This is achieved iteratively in the two-state Markov chain approach of [RL92]. It starts with sampling an arbitrary initial length trajectory, which is then used for estimating the values of α and β. M and N are calculated based on these estimates. Next, the trajectory is extended to reach the required length, and α and β values are re-estimated. The new estimates are used to re-calculate M and N. This process is iterated until $M + N$ is smaller than the current trajectory length. Finally, the resulting trajectory is used to estimate the steady-state probability of meta state 1. For more details, see [RL92]. Notice however the small

oversights in the formulas for m (absolute value missing in the denominator) and n (the inverse of Φ should be used) therein.

5.2 Two-state Markov Chain Approach: The Initialisation Problem

In this section, we first identify an initialisation problem of the original approach due to the size of the initial sample, this particular problem can lead to biased results. We then propose three heuristics to extend the approach for avoiding unfortunate initialisations.

Given good estimates of α and β, the theory of the two-state Markov chain approach presented above guarantees that the obtained value satisfies the imposed precision requirements. However, the method starts with generating a trajectory of the original DTMC of an arbitrarily chosen initial length, i.e., $M_0 + N_0 = 1 + (m_0 - 1)k + 1 + (n_0 - 1)k$, where m_0 is the 'burn-in' period and n_0 is the sample size of the two-state Markov chain abstraction. An unfortunate choice may lead to initial estimates of α and β that are biased and result in the new values of M and N such that $M + N$ is either smaller or not much larger than the initial $M_0 + N_0$. In the former case the algorithm stops immediately with the biased values for α, β and, more importantly, with an estimate for the steady-state probability that does not satisfy the precision requirements. The second case may lead to the same problem. As an illustration we considered a two-state Markov chain with $\alpha = \frac{24}{11873}$ (0.0020214) and $\beta = \frac{24}{25}$ (0.96). The steady-state probability distribution was $[0.997899\ 0.002101]$. With $k = 1$, $\epsilon = 10^{-6}$, $r = 10^{-3}$, $s = 0.95$, $m_0 = 5$, and $n_0 = 1,920$ the first estimated values for α and β were $\frac{1}{1918}$ (0.0005214) and 1, respectively. This subsequently led to $M = 2$ and $N = 1,999$, resulting in a request for the extension of the trajectory by 76. After the extension, the new estimates for α and β were $\frac{1}{1997}$ and 1, respectively. These estimates gave $M = 2$, $N = 1,920$, and the algorithm stopped. The estimated steady-state probability distribution was $[0.99950\ 0.00050]$, which was outside the pre-specified precision interval given by r. Independent 10^4 runs resulted in estimates of the steady-state probabilities that were outside the pre-specified precision interval 10% of times. Given the rather large number of repetitions, it can be concluded that the specified 95% confidence interval was not reached.

The reason for the biased result is the unfortunate initial value for n_0 and the fact that the real value of α is small. In the initialisation phase the value of α is underestimated and $\lceil n(\alpha, \beta) \rceil$ calculated based on the estimated values of α and β is almost the same as n_0. Hence, subsequent extension of the trajectory does not provide any improvement to the underestimated value of α since the elongation is too short.

To identify and avoid some of such pitfalls, we consider a number of cases and formulate some of the conditions in which the algorithm may fail to achieve the specified precision. To start, let n_0 be the initial size of the sample used for initial estimation of α and β. Neither α nor β is zero. It might be the case that the initial sample size is not big enough to provide non-zero estimates for both α and β. If this is the case, n_0 is doubled and the trajectory is elongated to collect a sample of required size. This is repeated iteratively until non-zero estimates for α and β are obtained. In the continuation we assume that n_0 provides non-zero estimates for both α and β. Then, the smallest possible estimates for both α and β are greater than $\frac{1}{n_0}$.

For a moment, let us set an upper bound value for n_0 to be 10^4. For most cases this

r	0.01			0.001			0.0001		
s	0.9	0.95	0.975	0.9	0.95	0.975	0.9	0.95	0.975
$n_0 \in$	$\emptyset$	$[2, 136]$	$\emptyset$	$[2, 1161]$	$[2, 1383]$	$[2, 1582]$	$[2, 11628]$	$[2, 13857]$	$[2, 15847]$

Table 5.1: Ranges of integer values for n_0 that do not satisfy the 'critical' condition $n(\alpha, \beta) < 2\, n_0$ for the given values of r and s.

boundary value is reasonable. Notice however that this is the case only if the real values of α and β are larger than 10^{-4}. In general, the selection of a proper value for n_0 heavily depends on the real values of α and β, which are unknown *a priori*. From what was stated above, it follows that both first estimates for α and β are greater than 10^{-4}. The following cases are possible.

(1) If both α and β are small, e.g., less than 0.1, then we have that $10^{-4} < \alpha, \beta < 0.1$ and $n(\alpha, \beta) > 72,765$ as can be seen by investigating the $n(\cdot, \cdot)$ function. In this case the sample size is increased more than 7-fold which is reasonable since the two-state Markov chain seems to be bad-mixing by the first estimates of the values for α and β and the algorithm asks for a significant increase of the sample size. We therefore conclude that the bad-mixing case is properly handled by the algorithm.

(2) Both first estimates of α and β are close to 1. If $\alpha, \beta \in [0.7, 0.98]$, the value of $n(\alpha, \beta)$ is larger than $19,000$. If both $\alpha, \beta > 0.98$, then the size of the sample drops, but in this case the Markov chain is highly well-mixing and short trajectories are expected to provide good estimates.

(3) The situation is somewhat different if one of the parameters is estimated to be small and the other is close to 1 as in the example described above. The extension to the trajectory is too small to significantly change the estimated value of the small parameter and the algorithm halts.

Considering the above cases leads us to the observation that the following situation needs to be treated with care: *The estimated value for one of the parameters is close to* $\frac{1}{n_0}$*, the value of the second parameter is close to* 1*, and* $n(\alpha, \beta)$ *is either smaller or not significantly larger than* n_0.

First approach: pitfall avoidance. To avoid this situation, we determine n_0 which in principle could lead to inaccurate initial estimates of α or β and such that the next sample size given by $\lceil n(\alpha, \beta) \rceil$ would practically not allow for an improvement of the estimates. As stated above, the 'critical' situation may take place when one of the parameters is estimated to be very small, i.e., close to $\frac{1}{n_0}$, and the increase in the sample size is not significant enough to improve the estimate. If the initial estimate is very small, the real value is most probably also small, but the estimate is not accurate. If the value is underestimated to the lowest possible value, i.e., $\frac{1}{n_0}$, on average the improvement can take place only if the sample size is increased at least by n_0. Therefore, with the trade-off between the accuracy and efficiency of the method in mind, we propose the sample size to be increased at least by n_0. Then the 'critical' situation condition is $n(\alpha, \beta) < 2\, n_0$. By analysing the function $n(\cdot, \cdot)$ as described in details in Section 5.6.4, we can determine the values of n_0 that are 'safe', i.e., which do not satisfy the 'critical' condition. We present them in Table 5.1 for a number of values for r and s.

Second approach: controlled initial estimation of α and β. The formula for n is asymptotically valid provided that the values for α and β are known. However, these

values are not known *a priori* and they need to be estimated. Unfortunately, the original approach does not provide any control over the quality of the initial estimate of the values of these parameters. In certain situation, e.g., as in the case discussed above, the lack of such control mechanism may lead to results with worse statistical confidence level than the specified one given by s. In the discussed example $s = 95\%$, but this value was not reached in the performed experiment. In order to address this problem, we propose to extend the initial phase of the two-state approach algorithm in the following way. The algorithm samples a trajectory of the original DTMC and estimates the values of α and β. We denote the estimates as $\hat{\alpha}$ and $\hat{\beta}$, respectively. Next, the algorithm computes the sample size required to reach the s confidence level that the true value of $\min(\alpha, \beta)$ is within a certain interval. For definiteness, we assume from now on that $\hat{\alpha} < \hat{\beta}$, which suggests that $\min(\alpha, \beta) = \alpha$. During the execution of the procedure outlined in the following the inequality may be inverted. If this is the case, the algorithm makes corresponding change in the consideration of α and β.

The aim is to have a good estimate for α. Notice that the smallest possible initial value of $\hat{\alpha}$ isgreater than $\frac{1}{n_0}$. We refer to $\frac{1}{n_0}$ as the *resolution of estimation*. Given the resolution, one cannot distinguish between values of α in the interval $(\hat{\alpha} - \frac{1}{n_0}, \hat{\alpha} + \frac{1}{n_0})$. In consequence, if $\alpha \in (\hat{\alpha} - \frac{1}{n_0}, \hat{\alpha} + \frac{1}{n_0})$, then the estimated value $\hat{\alpha}$ should be considered as optimal. Hence, one could use this interval as the one which should contain the real value with specified confidence level. Nevertheless, although the choice of this interval usually leads to very good results, as experimentally verified, the results are obtained at the cost of large samples which make the algorithm stop immediately after the initialisation phase. Consequently, the computational burden is larger than would be required by the original algorithm to reach the desired precision specified by r and s parameters in most cases. In order to reduce this unnecessary overhead, we consider the interval $(\hat{\alpha} - \frac{\hat{\alpha}}{2}, \hat{\alpha} + \frac{\hat{\alpha}}{2})$, which is wider than the previous one whenever $\hat{\alpha} > \frac{1}{n_0}$ and leads to smaller sample sizes.

The two-state Markov chain consists of two states 0 and 1, i.e., the two meta states of the original DTMC. We set α as the probability of making the transition from state 0 to state 1 (denoted as $0 \to 1$). The estimate $\hat{\alpha}$ is computed as the ratio of the number of transitions from state 0 to state 1 to the number of transition from state 0. Let $n_{0,\alpha}$ be the number of transitions in the sample starting from state 0. Let X_i, $i = 1, 2, \ldots, n_{0,\alpha}$, be a random variable defined as follows: X_i is 1 if ith transition from meta-state 0 is $0 \to 1$ and 0 otherwise.

Notice that state 0 is an accessible atom in the terminology of the theory of Markov chains, i.e., the Markov chain regenerates after entering state 0, and hence the random variables X_i, $i = 1, 2, \ldots, n_{0,\alpha}$, are independent. They are Bernoulli distributed with parameter α. The unbiased estimate of the population variance from the sample, denoted $\hat{\sigma}^2$, is given by $\hat{\sigma}^2 = \hat{\alpha} \cdot (1 - \hat{\alpha}) \cdot \frac{n_{0,\alpha}}{n_{0,\alpha}-1}$. Due to independence, $\hat{\sigma}^2$ is also the asymptotic variance and, in consequence, the sample size that provides the specified confidence level for the estimate of the value of α is given by $n_{\alpha,s}(\hat{\alpha}, n_{0,\alpha}) = \hat{\alpha} \cdot (1 - \hat{\alpha}) \cdot \frac{n_{0,\alpha}}{n_{0,\alpha}-1} \cdot \left(\frac{\Phi^{-1}(\frac{1}{2}(1+s))}{\hat{\alpha}/2} \right)^2$. The Markov chain is in state 0 with steady-state probability $\frac{\beta}{\alpha+\beta}$. Then, given that the chain reached the steady-state distribution, the expected number of regenerations in a sample of size n is given by $\frac{n \cdot \beta}{\alpha+\beta}$. Therefore, the sample size used to estimate the value of α with the specified confidence level s is given by $n_\alpha = \frac{\alpha+\beta}{\beta} \cdot n_{\alpha,s}(\hat{\alpha}, n_{0,\alpha})$. As the real values of α and β are unknown, the estimated

Model	Computed confidence level			Average sample size		
	Original	2nd	3rd	Original	2nd	3rd
PBN 1	88.3	96.7	95.5	9265	9040	9418
PBN 2	87.8	99.3	96.5	7731	13635	8201

Table 5.2: Performance of the second and third approaches.

values $\hat{\alpha}$ and $\hat{\beta}$ can be used in the above formula. If the computed n_α is bigger than the current number of transitions $n_{0,\alpha}$, we extend the trajectory to reach n_α transitions from 0 to 1 and re-estimate the values for α and β using the extended trajectory. We repeat this process until the computed n_α value is smaller than the number of transitions used to estimate α. In this way, good initial estimates for α and β are obtained and the original two-state Markov chain approach using the formula for $n(\alpha, \beta)$ is run.

Third approach: simple heuristics. When performing the initial estimation of α and β, we require both the count of transitions from state 0 to state 1 and the count of transitions from meta-state 1 to state 0 be at least 3. If this condition is not satisfied, we proceed by doubling the length of the trajectory. In this way the problem of reaching the resolution boundary is avoided. Our experiments showed that this simple approach in many cases led to good initial estimates of the α and β probabilities.

Discussions. The first approach provides us with safe initial starting points. As can be seen in Table 5.1, there might however be no safe starting point in certain conditions. Nevertheless, the first approach can be used in the initialisation phase of the other two approaches. The second approach introduces a new iteration process to provide a good estimate of α or β. The third one modifies the two-state Markov chain approach by adding only one extra restriction and therefore is the most simple one. We have verified with experiments that the last two approaches have the potential to make the two-state Markov chain approach meet the predefined precision requirement even in the case of an unlucky initial sample size. As a small example, we show in Table 5.2 the results for verifying two PBNs each of eight nodes. For each of the PBNs, we compute the steady-state probability for one subset of states using three different approaches: 1) the original two-state Markov chain approach (columns 'Original'), the proposed second approach (columns '2nd') and the proposed third approach (columns '3rd'). The precision and confidence level are set to 0.001 and 0.95 respectively. We repeat the computation for 1000 times and count the percentage of times that the calculated result is within the precision requirement (shown in columns labelled 'Computed confidence level'). As can be seen in Table 5.2, the original two-state Markov chain approach fails to meet the confidence level requirement while the both proposed approaches can meet the requirement. Due to its simplicity, we use the third approach in the remaining of the paper.

5.3 Evaluation

In this section, we focus on verifying the performance of the two-state Markov chain approach with another related method called the Skart method [TWLS08]. We use the tool ASSA-PBN [MPY15, MPY16a] as the platform for this verification. ASSA-PBN is a tool specially designed for steady-state analysis of large PBNs; it includes the two-state Markov chain approach with the simple heuristics presented in Section 5.1 and

the Skart method. For the steady-state analysis of large PBNs, applications of these two methods necessitate generation of trajectories of significant length. To make this efficient, we applied the alias method [Wal77] to sample the consecutive trajectory state. This enables ASSA-PBN, e.g, to simulate $12,580$ steps within 1s for a $2,000$ nodes PBN, which is hundreds of times faster than the related tool optPBN [TMP+14].

In Section 5.3.1, we briefly describe the Skart method. We present an empirical comparison of the performance of these two methods in Section 5.3.2.

5.3.1 The Skart Method

We choose the Skart method [TWLS08] as a reference for the evaluation of the performance of the two-state Markov chain approach. The Skart method is a successor of ASAP3, WASSP, and SBatch methods, which are all based on the idea of batch means [TWLS08]. It is a procedure for on-the-fly statistical analysis of the simulation output, asymptotically generated in accordance with a steady-state distribution. Usually it requires an initial sample of size smaller than other established simulation analysis procedures [TWLS08]. Briefly, the algorithm partitions a long simulation trajectory into batches, for each batch computes a mean and constructs an interval estimate using the batch means. Further, the interval estimate is used by Skart to decide whether a steady state distribution is reached or more samples are required. For a more detailed description of this method, see [TWLS08].

The Skart method differs in three key points with the two-state Markov chain approach. First, it specifies the initial trajectory length to be at least $1,280$, while for the two-state Markov chain approach this information is not provided. This difference, however, does not change the fact that the two methods can provide the same accuracy guarantee providing that the unlucky choice of the initial trajectory length of the two-state Markov chain approach is fixed as mentioned in the previous section. Second, the Skart method applies the student distribution for skewness adjustment while the two-state approach makes use of the normal distribution for confidence interval calculations. Thirdly, the two-state Markov chain approach does not require to keep track of the simulated trajectories; instead, the statistics (e.g., the α and β as in Figure 5.1) of the trajectories are enough. the Skart method, however, requires to keep track of the simulated trajectories, which consumes a large memory in the cases of large size trajectories.

5.3.2 Performance Evaluation

To compare the performance of the two methods, we randomly generated 882 different PBNs using ASSA-PBN. ASSA-PBN can randomly generate a PBN which satisfies structure requirements given in the form of five input parameters: the node number, the minimum and the maximum number of predictor functions per node, finally the minimum and maximum number of parent nodes for a predictor function. We generated PBNs with node numbers from $\{15, 30, 80, 100, 150, 200, 300, 400, 500, 1000, 2000\}$. We assigned the obtained PBNs into three different classes with respect to the density measure $\mathcal{D}$: *dense models* with density 150–300, *sparse models* with density around 10, and *in-between models* with density 50–100. The two-state Markov chain approach and the Skart method were tested on these PBNs with precision r set to the values in $\{10^{-2}, 5\times10^{-3}, 10^{-3}, 5\times10^{-4}, 10^{-4}, 5\times10^{-5}, 1\times10^{-5}, 5\times10^{-6}, 1\times10^{-6}\}$. We set

k	0	5	10	15	20	25	30
$t_{TS} \leq t_{Skart}$	69.03%	54.04%	40.06%	30.19%	25.24%	22.22%	20.18%
$t_{Skart} \leq t_{TS}$	30.97%	19.32%	11.98%	8.27%	6.42%	5.39%	4.83%

Table 5.3: Performance comparison of the Skart and the two-state MC methods.

	$t_{TS} \leq t_{Skart}$						$t_{Skart} \leq t_{TS}$					
k	0	5	10	15	20	25	0	5	10	15	20	25
node number	-0.17	-0.15	-0.09	-0.01	0.04	0.09	0.17	0.20	0.21	0.27	0.25	0.28
precision	0.34	0.49	0.68	0.84	0.93	0.92	-0.34	-0.09	0.16	0.38	0.45	0.48
density	-0.15	-0.19	-0.29	-0.41	-0.52	-0.53	0.15	0.11	-0.04	-0.15	-0.27	-0.37

Table 5.4: Logistic regression coefficient estimates for performance prediction.

precision	10^{-2}	5×10^{-3}	10^{-3}	5×10^{-4}	10^{-4}	5×10^{-5}	10^{-5}	5×10^{-6}	10^{-6}
$t_{TS} \leq t_{Skart}$	84%	76%	67%	64%	65%	59%	73%	75%	85%

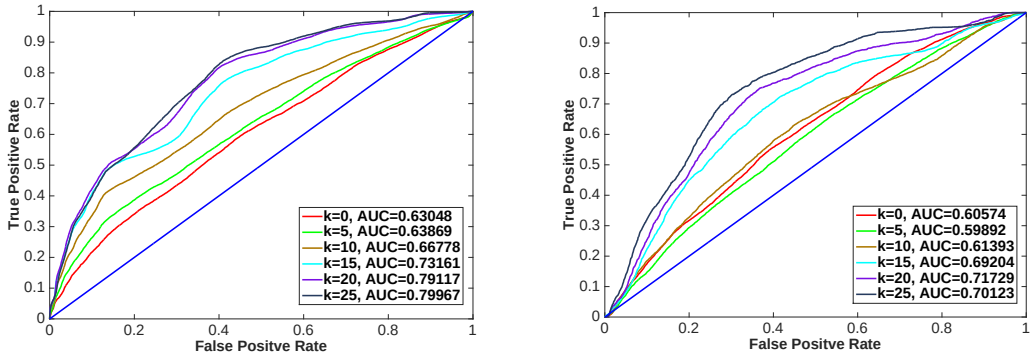
Table 5.5: Performance of the two methods with respect to different precisions.

ϵ to 10^{-10} for the two-state Markov chain approach and s to 0.95 for both methods.

The experiments were performed on a HPC cluster, with CPU speed ranging between 2.2GHz and 3.07GHz. ASSA-PBN is implemented in Java and the initial and maximum Java virtual machine heap size were set to 503MB and 7.86GB, respectively. We collected 5596 valid (precision being smaller than probability) results with the information on the PBN node number, its density class, the precision value, the estimated steady-state probabilities computed by the two methods, and their CPU time costs. The steady-state probabilities computed by the two methods are comparable in all the cases (data not shown in the paper). For each experimental result i, we compare the time costs of the two methods. Let $t_{TS}(i)$ and $t_{Skart}(i)$ be the time cost for the two-state Markov chain approach and the Skart method, respectively. We say that the two-state Markov chain approach is by k per cent faster than the Skart method if $\frac{(t_{Skart}(i)-t_{TS}(i))}{t_{Skart}(i)} \geqslant \frac{k}{100}$. The definition for the Skart method to be faster than the two-state Markov chain approach is symmetric. In Table 5.3 we show the percentage of cases in which the two-state approach was by k per cent faster than Skart and vice versa for different k. In general, *in nearly 70% of the results, the two-state Markov chain approach was faster than the Skart method*. The number of cases the two-state Markov chain approach was faster than the Skart method is also larger than in the opposite case.

Next, we analyse the results with a machine learning technique, i.e., logistic regression, in MATLAB. We use the node number, the precision, and the density class as features. We label each result as 1 if the two-state Markov chain approach is by k per cent faster than the Skart method and as 0 otherwise. We plot the *receiver operating characteristic* (ROC) curve, which is commonly used to illustrate the performance of a binary classifier against varying discrimination threshold and we give the computed area under the curve (AUC) for different k in Figure 5.2a. When $k \geqslant 15$, the AUC value is over 0.7, which means that the prediction is very good. In another word, for a given PBN and precision requirement, we are able to predict whether the two-state Markov chain approach will be by 15 per cent faster than the Skart method in a very high accuracy rate.

We show in Table 5.4 (left part) the regression coefficient estimates of the three features.



(a) ROC ($t_{TS} \leq t_{Skart}$). (b) ROC ($t_{Skart} \leq t_{TS}$).

Figure 5.2: Prediction on the performance of the the Skart and the two-state MC methods.

Clearly, the precision plays an important role in the prediction since the absolute value is always the largest. We further analyse how the performance of the two methods change with precision and show in Table 5.5 the percentage of cases that the two-state Markov chain approach is faster than the Skart method with respect to different precisions. *The two-state Markov chain approach has a larger chance to be faster than the Skart method in all the studied precisions*, especially when the precision r is low (e.g., 10^{-2}) or very high (e.g., equal to or less than 10^{-5}). Notably, the chance that the two-state Markov chain approach is faster than the Skart method becomes very large when the precision is very high. This is due to the fact that the Skart method requires a large memory to keep track of the large size trajectory when the precision is high. The CPU performance drops when operating on a large memory; on the other hand, the Skart method may run out of memory.

Moreover, we analyse the situation when the Skart method is by k per cent faster than the two-state Markov chain approach. This time it becomes difficult to make an accurate prediction as the largest AUC is only about 0.72 for $k = 20$ (see Figure 5.2b). Besides, the coefficient estimates in the right part of Table 5.4 also vary a lot with k, and precision is not always the dominating factor. The detailed experiment data can be obtained at `http://satoss.uni.lu/software/ASSA-PBN/benchmark/benchmark.xlsx`.

From the above analysis, we conclude that the two-state Markov chain approach outperforms the Skart method (state-of-the-art) in analysing large PBNs, especially for computing steady-state probabilities with very high precision.

5.4 A Biological Case study

In this case study, we perform a list of steady-state analysis which require the knowledge of a few definitions. We give them in the following section.

5.4.1 Preliminaries of Steady-state Analysis

Within the framework of PBNs the concept of influences is defined; it formalizes the impact of parents nodes on a target node and enables its quantification ([SDKZ02]). The concept is based on the notion of a partial derivative of a Boolean function f with respect to variable x_j ($1 \leq j \leq n$):

$$\frac{\partial f(x)}{\partial x_j} = f(x^{(j,0)}) \oplus f(x^{(j,1)}),$$

where $\oplus$ is addition modulo 2 (exclusive OR) and for $l \in \{0, 1\}$

$$x^{(j,l)} = (x_1, x_2, \ldots, x_{j-1}, l, x_{j+1}, \ldots, x_n).$$

The *influence of node x_j on function f* is the expected value of the partial derivative with respect to the probability distribution $D(x)$:

$$I_j(f) = \mathbb{E}_D\Big[\frac{\partial f(x)}{\partial x_j}\Big] = \mathbb{P}\Big\{\frac{\partial f(x)}{\partial x_j} = 1\Big\} = \mathbb{P}\{f(x^{(j,0)}) \neq f(x^{(j,1)})\}.$$

Let now F_i be the set of predictors for x_i with corresponding probabilities $c_j^{(i)}$ for $j = 1, \ldots, l(i)$ and let $I_k(f_j^{(i)})$ be the influence of node x_k on the predictor function $f_j^{(i)}$. Then, the *influence of node x_k on node x_i* is defined as:

$$I_k(x_i) = \sum_{j=1}^{l(i)} I_k(f_j^{(i)}) \cdot c_j^{(i)}.$$

The *long-term influences* are the influences computed when the distribution $D(x)$ is the stead-state distribution of the PBN.

We define and consider in this study two types of long-run sensitivities.

Definition 5.4.1. *The* long-run sensitivity with respect to selection probability perturbation *is defined as*

$$s_c[c_j^{(i)} = p] = \|\tilde{\pi}[c_j^{(i)} = p] - \pi\|_l,$$

where $\|\cdot\|_l$ denotes the l-norm, π is the steady-state distribution of the original PBN, $p \in [0, 1]$ is the new value for $c_j^{(i)}$, and $\tilde{\pi}[c_j^{(i)} = p]$ is the steady-state probability distribution of the PBN perturbed as follows. The jth selection probability for node x_i is replaced with $\tilde{c}_j^{(i)} = p$ and all $c_k^{(i)}$ selection probabilities for $k \in I_{-j} = \{1, 2, \ldots, j-1, j+1, \ldots, l(i)\}$ are replaced with

$$\tilde{c}_k^{(i)} = c_k^{(i)} + (c_j^{(i)} - p) \cdot \frac{c_k^{(i)}}{\sum_{l \in I_{-j}} c_l^{(i)}},$$

The remaining selection probabilities of the original PBN are unchanged.

Definition 5.4.2. *The* long-run sensitivity with respect to permanent on/off perturbations of a node x_i *as*

$$s_g[x_i] = \max\{\|\tilde{\pi}[x_i \equiv 0] - \pi\|_l, \|\tilde{\pi}[x_i \equiv 1] - \pi\|_l\},$$

where π, $\tilde{\pi}[x_i \equiv 0]$, and $\tilde{\pi}[x_i \equiv 1]$ are the steady-state probability distributions of the original PBN, of the original PBN with all $f^{(i)} \in F_i$ replaced by $\tilde{f}^{(i)} \equiv 0$, and all $f^{(i)} \in F_i$ replaced by $\tilde{f}^{(i)} \equiv 1$, respectively.

Notice that the definition of long-run sensitivity with respect to permanent on/off perturbations is similar but not equivalent to the definition of long-run sensitivity with respect to 1-gene function perturbation of [SDKZ02].

5.4.2 An Apoptosis Network

In [SSV+09], a large-scale Boolean network of apoptosis (see Figure 3.8) in hepatocytes was introduced, where the assigned Boolean interactions for each molecule were derived from literature study. In [TMP+14], the original multi-value Boolean model was cast into the PBN framework: a binary PBN model, so-called 'extended apoptosis model' which comprised 91 nodes (state-space of size 2^{91}) and 102 interactions was constructed. In this extended version the possibility of activation of NF-κB through Caspase 8 (C8*), as described in [TMP+14], was included. The model was fitted to steady-state experimental data obtained in response to six different stimulations of the input nodes, see [TMP+14] for details.

As can be seen from the wiring of the network, the activation of complex2 (co2) by RIP-deubi can take place in two ways: 1) by a positive feedback loop from activated C8* and P $\rightarrow$ tBid $\rightarrow$ Bax $\rightarrow$ smac $\rightarrow$ RIP-deubi $\rightarrow$ co2 $\rightarrow$ C8*-co2 $\rightarrow$ C8*, and 2) by the positive signal from UV-B irradiation (input nodes UV(1) or UV(2)) $\rightarrow$ Bax $\rightarrow$ smac $\rightarrow$ RIP-deubi $\rightarrow$ co2. The former to be active requires the stimulation of the type 2 receptor (T2R). The latter way requires complex1 (co1) to be active, which cannot happen without the stimulation of the TNF receptor-1. Therefore, RIP-deubi can activate co2 only in the condition of co-stimulation by TNF and either UV(1) or UV(2). In consequence, it was suggested in [TMP+14] that the interaction of activation of co2 via RIP-deubi is not relevant and could be removed from the model in the context of modelling primary hepatocyte. However, due to the problem with efficient generation of very long trajectories in optPBN toolbox, quantitative analysis was hindered and this hypothesis could not be verified ([TMP+14]).

In this work, we take up this challenge and we quantitatively investigate the relevancy of the interaction of activation of co2 via RIP-deubi. We perform an extensive analysis in the context of co-stimulation by TNF and either UV(1) or UV(2): we compute long-term influences of parent nodes on the co2 node and the long-run sensitivities with respect to various perturbations related to specific predictor functions and their selection probabilities. For this purpose we apply the two-state Markov chain approach as implemented in our ASSA-PBN tool [MPY15] to compute the relevant steady-state probabilities for the best-fit models described in [TMP+14]. Due to the efficient implementation, the ASSA-PBN tool can easily deal with trajectories of length exceeding 2×10^9 for this case study.

We consider 20 distinct parameter sets of [TMP+14] that resulted in the best fit of the 'extended apoptosis model' to the steady-state experimental data in six different stimulation conditions. In [TMP+14], parameter estimation was performed with steady-state measurements for the nodes apoptosis, C3ap17 or C3ap17_2 depending on the stimulation condition considered, and NF-κB. The optimisation procedure used was Particle Swarm and fit score function considered was the sum of squared errors of prediction (SSE) and the sum was taken over the three nodes in the six stimulation conditions. We took all the optimisation results from the three independent parameter estimation runs of [TMP+14], each containing 7500 parameter sets. We sorted them increasingly with

	TNF and UV(1)			TNF and UV(2)		
	$I_{\text{RIP-deubi}}$	I_{co1}	I_{FADD}	$I_{\text{RIP-deubi}}$	I_{co1}	I_{FADD}
Best fit	0.2614	0.9981	0.9935	0.2615	0.9980	0.9936
Min	0.0000	0.9979	0.9935	0.0000	0.9979	0.9936
Max	0.3145	0.9988	0.9944	0.3146	0.9990	0.9947
Mean	0.2087	0.9982	0.9937	0.2088	0.9982	0.9938
Std	0.0735	0.0002	0.0002	0.0735	0.0002	0.0003

Table 5.6: Long-term influences of RIP-duebi, co1, and FADD on co2 in the 'extended apoptosis model' in [TMP+14] under the co-stimulation of both TNF and UV(1) or UV(2).

respect to the cost function value obtained during optimisation, removed duplicates, and finally took the first 20 best-fit parameter sets.

As mentioned above, we fix the experimental context to co-stimulation of TNF and either UV(1) or UV(2). We note that originally in [SSV+09] UV-B irradiation conditions were imposed via a multi-value input node UV which could take on three values, i.e., 0 (no irradiation), 1 ($300\,J/m^2$ UV-B irradiation), and 2 ($600\,J/m^2$ UV-B irradiation). In the model of [TMP+14], UV input node was refined as UV(1) and UV(2) in order to cast the original model into the binary PBN framework. Therefore, we consider in our study two cases: 1) co-stimulation of TNF and UV(1) and 2) co-stimulation of TNF and UV(2). Node co2 has two independent predictor functions: co2 = co1 $\wedge$ FADD or co2 = co1 $\wedge$ FADD $\wedge$ RIP-deubi. The selection probabilities are denoted as $c_1^{(co2)}$ and $c_2^{(co2)}$, respectively. Their values have been optimised in [TMP+14].

We start with computing the influences with respect to the steady-state distribution, i.e., the long-term influences on co2 of each of its parent nodes: RIP-deubi, co1, and FADD, in accordance with the definition in Section 5.4.1. Notice that the computation of the three influences requires several joint steady-state probabilities to be estimated with the two-state Markov chain approach, e.g., (co1=1,FADD=1,RIP-deubi=0) or (co1=1,FADD=0). Each probability determines a specific split of the original Markov chain. For example, in the case of the estimation of the joint steady-state probability for (co1=1,FADD=0), the states of the underlying Markov chain of the apoptosis PBN model in which co1=1 and FADD=0 constitute meta state 1 and all the remaining states form meta state 0. Therefore, the estimation of influences is computationally demanding. The summarised results for the 20 parameter sets are presented for the co-stimulation of TNF and UV(1) or TNF and UV(2) in Table 5.6. They are consistent across the different parameter sets and clearly indicate that the influence of RIP-deubi on co2 is small compared to the influence of co1 or FADD on co2. However, the influence of RIP-deubi is not negligible.

We take the analysis of the importance of the interaction between RIP-deubi and co2 further and we compute various long-run sensitivities with respect to selection probability perturbation. In particular, we perturb the selection probability $c_2^{(co2)}$ by $\pm 5\%$, i.e., we set the new value by multiplying the original value by (1 ± 0.05), and compute in line with Definition 5.4.1 how the joint steady-state distribution for (apoptosis,C3ap17,NF-κB) differs from the non-perturbed one with respect to the l_1 norm, i.e., $||\cdot||_1$. We notice that the computation of the full steady-state distribution for the considered PBN model of apoptosis is practically intractable, i.e., it would require the estimation of 2^{91} val-

$c_2^{(co2)}$	TNF and UV(1)			TNF and UV(2)		
	$+5\%$	-5%	$=0$	$+5\%$	-5%	$=0$
Best fit	0.0003	0.0002	0.0011	0.0002	0.0004	0.0011
Min	0.0002	0.0002	0.0003	0.0002	0.0002	0.0002
Max	0.0008	0.0008	0.0014	0.0012	0.0007	0.0013
Mean	0.0005	0.0005	0.0009	0.0004	0.0004	0.0009
Std	0.0001	0.0001	0.0003	0.0002	0.0001	0.0003

Table 5.7: Long-run sensitivities w.r.t selection probability perturbations.

RIP-deubi f. pert.	Best fit	Min	Max	Mean	Std
TNF & UV(1)	0.3075	0.0130	0.3595	0.2089	0.0823
TNF & UV(2)	0.3097	0.0105	0.3612	0.2105	0.0827

Table 5.8: Long-run sensitivities w.r.t permanent on/off perturbations of RIP-deubi.

ues. Therefore, we restrict the computations to the estimation of eight joint steady-state probabilities for all possible combinations of values for (apoptosis,C3ap17,NFκB), i.e., the experimentally measured nodes. Each estimation is obtained by a separate run of the two-state Markov chain approach with the split into meta states determined by the considered probability as explained above in the case of the computation of long-term influences. To compare the estimated distributions we choose the l_1 norm after [QD09], where it is used in the computations of similar types of sensitivities for PBNs to these defined in Section 5.4.1. Notice that the l_1 norm of the difference of two probability distributions on a finite sample space is twice the *total variation distance*. The latter is a well-established metric for measuring the distance between probability distributions defined as the maximum difference between the probabilities assigned to a single event by the two distributions (see, e.g., [LPW09]). Additionally, we check the difference when $c_2^{(co2)}$ is set to 0 (and, in consequence, $c_1^{(co2)}$ is set to 1). The obtained results for the 20 parameter sets in the conditions of co-stimulation of TNF and UV(1) and co-stimulation of TNF and UV(2) are summarised in Table 5.7. In all these cases, the sensitivities are very small. Therefore, the system turns to be insensitive to small perturbations of the value of $c_2^{(co2)}$. Also the complete removal of the second predictor function for co2 does not cause any drastic changes in the joint steady-state distribution for (apoptosis,C3ap17,NF-κB).

Finally, we compute the long-run sensitivity with respect to permanent on/off perturbations of the node RIP-deubi in accordance with Definition 5.4.2. As before, we consider the joint steady-state distributions for (apoptosis,C3ap17,NF-κB) and we choose the l_1-norm. The results, given in Table 5.8, show that in both variants of UV-B irradiation the sensitivities are not negligible and the permanent on/off perturbations of RIP-deubi have impact on the steady-state distribution.

To conclude, all the obtained results indicate that in the context of co-stimulation of TNF and either UV(1) or UV(2) the interaction between RIP-deubi and co2 plays a certain role. Although the elimination of the interaction does not invoke significant changes to the considered joint steady-state distribution, the long-term influence of RIP-deubi on co2 is not negligible and may be important for other nodes in the network.

5.5 Discussions and Conclusion

Most current tools for statistical model checking, a simulation-based approach using hypothesis testing to infer whether a stochastic system satisfies a property, are restricted for bounded properties which can be checked on finite executions of the system. Recently, both the Skart method [Roh13] and the perfect simulation algorithm [EP09] have been explored for statistical model checking of steady state and unbounded until properties. The perfect simulation algorithm for sampling the steady-state of an ergodic DTMC is based on the indigenous idea of the *backward coupling scheme* [PW96]. It allows to draw independent samples which are distributed exactly in accordance with the steady-state distribution of a DTMC. However, due to the nature of this method, each state in the state space needs to be considered at each step of the coupling scheme. If a DTMC is monotone, then it is possible to sample from the steady-state distribution by considering the maximal and minimal states only [PW96]. This was exploited in [EP09] for model checking large queuing networks. Unfortunately, it is not applicable to PBNs with perturbations. In consequence, the perfect simulation algorithm is only suited for at most medium-size PBNs and large-size PBNs are out of its scope. Thus, we have only compared the performance of the two-state Markov chain approach with the Skart method.

Moreover, in this study we have identified a problem of generating biased results by the original two-state Markov chain approach and have proposed three heuristics to avoid wrong initialisation. Finally, we demonstrated the potential of the two-state Markov chain approach on a study of a large, 91-node PBN model of apoptosis in hepatocytes. The two-state Markov chain approach facilitated the quantitative analysis of the large network and the investigation of a previously formulated hypothesis regarding the relevance of the interaction of activation of co2 via RIP-deubi. In the future, we aim to investigate the usage of the discussed statistical methods for approximate steady-state analysis in a research project on systems biology, where we will apply them to develop new techniques for minimal structural interventions to alter steady-state probabilities for large regulatory networks.

5.6 Derivation of Formulas

5.6.1 Derivation of the Number of "Burn-in" Iterations

Let $\{Z_t\}_{t\geq 0}$ be a discrete-time two-state Markov chain as given in Figure 5.1b. Z_t has the value 0 or 1 if the system is in state 0 or state 1 at time n, respectively. The transition probabilities satisfy $0 < \alpha, \beta < 1$ and the transition matrix for this chain has the following form

$$P = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}.$$

Matrix P has two distinct eigenvalues: 1 and $\lambda = (1 - \alpha - \beta)$. Notice that $|\lambda| < 1$.

The chain is ergodic and the unique steady-state distribution is $\pi = [\pi_0\ \pi_1] = [\frac{\beta}{\alpha+\beta}\ \frac{\alpha}{\alpha+\beta}]$. Let $\mathbb{E}_\pi(Z_t)$ denote the expected value of Z_t for any fixed $t \geq 0$, with respect to the steady-state distribution π. We have that $\mathbb{E}_\pi(Z_t) = \frac{\alpha}{\alpha+\beta}$.

The m-step transition matrix can be written, as can be checked by induction, in the form

$$P^m = \begin{bmatrix} \pi_0 & \pi_1 \\ \pi_0 & \pi_1 \end{bmatrix} + \frac{\lambda^m}{\alpha+\beta} \cdot \begin{bmatrix} \alpha & -\alpha \\ -\beta & \beta \end{bmatrix},$$

where λ is the second eigenvalue of P.

Suppose we require m to be such that the following condition is satisfied

$$\begin{bmatrix} |\,\mathbb{P}[Z_m = 0 \,|\, Z_0 = j] - \pi_0\,| \\ |\,\mathbb{P}[Z_m = 1 \,|\, Z_0 = j] - \pi_1\,| \end{bmatrix} < \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \tag{5.3}$$

for some $\epsilon > 0$. If $e_0 = [1\ 0]$ and $e_1 = [0\ 1]$, then for $j \in \{0, 1\}$ we have that

$$\begin{bmatrix} \mathbb{P}[Z_m = 0 \,|\, Z_0 = j] \\ \mathbb{P}[Z_m = 1 \,|\, Z_0 = j] \end{bmatrix} = (e_j P^m)^{\mathrm{T}} = (P^m)^{\mathrm{T}} (e_j)^{\mathrm{T}},$$

where T is the transposition operator. For any vector $v = [v_1\ v_2\ \ldots\ v_n]^{\mathrm{T}} \in \mathbb{R}^n$ we use $|v|$ to denote $[|v_1|\ |v_2|\ \ldots\ |v_n|]^{\mathrm{T}}$. Therefore, condition (5.3) can be rewritten as

$$\left| (P^m)^{\mathrm{T}} (e_j)^{\mathrm{T}} - \begin{bmatrix} \pi_0 \\ \pi_1 \end{bmatrix} \right| < \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix}.$$

For $j = 0$ and $j = 1$ the above simplifies to

$$\left| \frac{\lambda^m}{\alpha+\beta} \cdot \begin{bmatrix} \alpha \\ -\alpha \end{bmatrix} \right| < \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \text{ and } \left| \frac{\lambda^m}{\alpha+\beta} \cdot \begin{bmatrix} -\beta \\ \beta \end{bmatrix} \right| < \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix},$$

respectively. Therefore, it is enough to consider the following two inequalities

$$\left| \frac{\lambda^m \alpha}{\alpha+\beta} \right| < \epsilon \qquad \text{and} \qquad \left| \frac{\lambda^m \beta}{\alpha+\beta} \right| < \epsilon,$$

which, since $\alpha, \beta > 0$, can be rewritten as

$$|\lambda^m| < \frac{\epsilon(\alpha+\beta)}{\alpha} \qquad \text{and} \qquad |\lambda^m| < \frac{\epsilon(\alpha+\beta)}{\beta}.$$

Equivalently, m has to satisfy

$$|\lambda^m| < \frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)}.$$

By the fact that $|\lambda^m| = |\lambda|^m$ this can be expressed as

$$|\lambda|^m < \frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)}.$$

Then, by taking the logarithm to base 10 on both sides[1], we have that

$$m \cdot \log(|\lambda|) < \log\left(\frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)} \right)$$

and in consequence, since $|\lambda| < 1$ and $\log|\lambda| < 0$,

$$m > \frac{\log\left(\frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)} \right)}{\log(|\lambda|)}.$$

[1] In fact, by the formula for change of base for logarithms, the natural logarithm ($\ln$), the logarithm to base 2 ($\log_2$), or a logarithm to any other base could be used to calculate m instead of $\log$. Notice that m does **not** depend on the choice of the base of the logarithm!

5.6.2 Derivation of the Sample Size

By the Law of Large Numbers for irreducible positive recurrent Markov chains $\overline{Z}_n \to \pi_1$ $a.\ s.$ with $n \to \infty$, where $\overline{Z}_n = \frac{1}{n}\sum_{t=1}^{n} Z_t$. Now, by a variant of the Central Limit Theorem for non-independent random variables[2], for n large, $\overline{Z}_n$ is approximately normally distributed with mean $\pi_1 = \frac{\alpha}{\alpha+\beta}$ and asymptotic variance $\sigma^2_{\text{as}} = \frac{1}{n}\frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3}$, see Section 5.6.3 for the derivation of the asymptotic variance. Let X be the standardised $\overline{Z}_n$, i.e.,

$$X = \frac{\overline{Z}_n - \pi_1}{\sigma_{\text{as}}/\sqrt{n}}.$$

If follows that X is normally distributed with mean 0 and variance 1, i.e., $X \sim N(0,1)$. Now, we require n to be such that the condition $\mathbb{P}[\pi_1 - r \leq \overline{Z}_n \leq \pi_1 + r] = s$ is satisfied for some specified r and s. This condition can be rewritten as

$$\mathbb{P}[-r \leq \overline{Z}_n - \pi_1 \leq r] = s,$$

and further as

$$\mathbb{P}[-r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}} \leq \frac{\overline{Z}_n - \pi_1}{\sigma_{\text{as}}/\sqrt{n}} \leq r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}}] = s,$$

which is

$$\mathbb{P}[-r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}} \leq X \leq r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}}] = s.$$

Since $X \sim N(0,1)$ and $N(0,1)$ is symmetric around 0, it follows that

$$\mathbb{P}[0 \leq X \leq r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}}] = \frac{s}{2}$$

and

$$\mathbb{P}[X \leq r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}}] = \frac{1}{2} + \frac{s}{2} = \frac{1}{2}(1+s).$$

Let $\Phi(\cdot)$ be the standard normal cumulative distribution function. Then the above can be rewritten as

$$\Phi(r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}}) = \frac{1}{2}(1+s).$$

Therefore, if we denote the inverse of the standard normal cumulative distribution function with $\Phi^{-1}(\cdot)$, we have that

$$r \cdot \frac{\sqrt{n}}{\sigma_{\text{as}}} = \Phi^{-1}(\frac{1}{2}(1+s)).$$

In consequence,

$$n = \frac{\sigma^2_{\text{as}}}{\left\{\frac{r}{\Phi^{-1}(\frac{1}{2}(1+s))}\right\}^2} = \frac{\frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3}}{\left\{\frac{r}{\Phi^{-1}(\frac{1}{2}(1+s))}\right\}^2}.$$

[2]Notice that the random variables Z_t, Z_{t+1} which values are consecutive states of a trajectory are correlated and are not independent.

5.6.3 Derivation of the Asymptotic Variance

By the Central Limit Theorem for stationary stochastic processes[3] $\sqrt{n}(\overline{Z}_n - \pi_1) \xrightarrow{d} N(0, \sigma^2_{\text{as}})$ as $n \to \infty$, where σ^2_{as} is the so-called asymptotic variance given by

$$\sigma^2_{\text{as}} = \text{Var}_\pi(Z_j) + 2\sum_{k=1}^{\infty} \text{Cov}_\pi(Z_j, Z_{j+k}) \tag{5.4}$$

and $\text{Var}_\pi(\cdot)$ and $\text{Cov}_\pi(\cdot)$ denote the variance and covariance with respect to the steady-state distribution π, respectively. We proceed to calculate σ^2_{as}. First, observe that $\mathbb{E}_\pi(Z_n Z_{n+1}) = \frac{\alpha}{\alpha+\beta}(1-\beta)$: $Z_n Z_{n+1} \neq 0$ if and only if the chain is state 1 at time n and remains in 1 at time $n+1$, i.e., $Z_n = Z_{n+1} = 1$. The probability of this event at steady state is $\frac{\alpha}{\alpha+\beta}(1-\beta)$. Then, by the definition of covariance, we have that the steady-state covariance between consecutive random variables of the two-state Markov chain, i.e., $\text{Cov}_\pi(Z_n, Z_{n+1})$ is

$$\begin{aligned}
&\text{Cov}_\pi(Z_n, Z_{n+1}) = \mathbb{E}_\pi\left[(Z_n - \mathbb{E}_\pi(Z_n))(Z_{n+1} - \mathbb{E}_\pi(Z_{n+1}))\right] \\
&= \mathbb{E}_\pi\left[(Z_n - \frac{\alpha}{\alpha+\beta})(Z_{n+1} - \frac{\alpha}{\alpha+\beta})\right] \\
&= \mathbb{E}_\pi\left[Z_n Z_{n+1} - \frac{\alpha}{\alpha+\beta}(Z_n + Z_{n+1}) + \frac{\alpha^2}{(\alpha+\beta)^2}\right] \\
&= \mathbb{E}_\pi(Z_n Z_{n+1}) - \frac{\alpha}{\alpha+\beta}(\mathbb{E}_\pi(Z_n) + \mathbb{E}_\pi(Z_{n+1})) + \frac{\alpha^2}{(\alpha+\beta)^2} \\
&= \frac{\alpha(1-\beta)}{\alpha+\beta} - 2\frac{\alpha^2}{(\alpha+\beta)^2} + \frac{\alpha^2}{(\alpha+\beta)^2} \\
&= \frac{\alpha\beta(1-\alpha-\beta)}{(\alpha+\beta)^2}.
\end{aligned}$$

Further, we have that $\text{Var}_\pi(Z_n) = \pi_0 \cdot \pi_1 = \frac{\alpha\beta}{(\alpha+\beta)^2}$ (variance of the Bernoulli distribution) and it can be shown that $\text{Cov}_\pi(Z_n, Z_{n+k}) = (1-\alpha-\beta)^k \cdot \text{Var}_\pi(Z_n)$ for any $k \geq 1$. Now, according to Equation (5.4), we have

$$\begin{aligned}
\sigma^2_{\text{as}} &= \text{Var}_\pi(X_j) + 2\sum_{k=1}^{\infty} \text{Cov}_\pi(X_j, X_{j+k}) \\
&= \frac{\alpha\beta}{(\alpha+\beta)^2} + 2\sum_{k=1}^{\infty}(1-\alpha-\beta)^k \cdot \frac{\alpha\beta}{(\alpha+\beta)^2} \\
&= \frac{\alpha\beta}{(\alpha+\beta)^2} + \frac{2\alpha\beta}{(\alpha+\beta)^2} \cdot \sum_{k=1}^{\infty}(1-\alpha-\beta)^k \\
&= \frac{\alpha\beta}{(\alpha+\beta)^2} + \frac{2\alpha\beta}{(\alpha+\beta)^2} \cdot \frac{1-\alpha-\beta}{\alpha+\beta} \\
&= \frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3}.
\end{aligned}$$

In consequence, $\overline{Z}_n$ is approximately normally distributed with mean $\frac{\alpha}{\alpha+\beta}$ and variance $\frac{1}{n}\frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3}$.

[3] After discarding the 'burn-in' part of the trajectory, we can assume that the Markov chain in a stationary stochastic process.

5.6.4 'Pitfall Avoidance' Heuristic Method: Formula Derivations

We start with analysing the minimum values $n(\cdot,\cdot)$ can attain. The function is considered on the domain $D = (0,1] \times (0,1]$ and, as mentioned before, the estimated values of α and β are within the range $[\frac{1}{n_0}, 1]$. Computing the partial derivatives, equating them to zero, and solving for α and β yields $\alpha = -\beta$, which has no solution in the considered domain. Hence, the function has neither local minimum nor maximum on D. Let us fix β for a moment and consider $n(\alpha,\beta)$ as a function of α. We denote it as $n_\beta(\alpha)$. By differentiating with respect to α, we obtain

$$\frac{\partial}{\partial\alpha} n_\beta(\alpha) = \frac{1}{c_{r,s}} \frac{\beta\,(\alpha^2 - \beta^2 - 4\,\alpha + 2\,\beta)}{(\alpha+\beta)^4},$$

where

$$c_{r,s} = \frac{r^2}{\left(\Phi^{-1}\left(\frac{1}{2}(1+s)\right)\right)^2}.$$

By equating to zero and solving for α we get two solutions: $\alpha_1 = 2 - \sqrt{\beta^2 - 2\beta + 4}$ and $\alpha_2 = 2 + \sqrt{\beta^2 - 2\beta + 4}$. Since the second solution is always greater than 1 on the $(0,1]$ interval, only the first solution is valid. The sign of the second derivative of $n_\beta(\alpha)$ with respect to α at α_1 is negative. This shows that for any fixed β, $n_\beta(\alpha)$ grows on the interval $[\frac{1}{n_0}, \alpha_1]$, attains its maximum at α_1 and decreases on the interval $[\alpha_1, 1]$. Notice that n is symmetric, i.e., $n(\alpha,\beta) = n(\beta,\alpha)$. Thus the minimum value n could attain for α and β estimated from a sample of size n_0 is given by $\min\left(n\left(\frac{1}{n_0},\frac{1}{n_0}\right), n\left(\frac{1}{n_0},1\right)\right)$. After evaluating n we get

$$n\left(\frac{1}{n_0},\frac{1}{n_0}\right) = \frac{n_0 - 1}{4\,c_{r,s}}$$

and

$$n\left(\frac{1}{n_0},1\right) = \frac{(n_0-1)\cdot n_0}{c_{r,s}\cdot(1+n_0)^3}.$$

Now, to avoid the situation where the initial estimates of α and β lead to $n(\alpha,\beta) < 2\,n_0$, it is enough to make sure that given r and s the following condition is satisfied: $\min(n(\frac{1}{n_0},\frac{1}{n_0}), n(\frac{1}{n_0},1)) \geqslant 2\,n_0$. This can be rewritten as

$$\begin{cases} (8\,c_{r,s} - 1)\,n_0 + 1 \leq 0 \\ \\ 2\,c_{r,s}\,n_0^3 + 6\,c_{r,s}\,n_0^2 + (6\,c_{r,s} - 1)\,n_0 + 2\,c_{r,s} + 1 \leq 0 \end{cases}$$

Both inequalities can be solved analytically. Given that $n_0 > 0$, the solution of the first inequality is

$$\begin{cases} n_0 \in [-\frac{1}{8\cdot c_{r,s}-1}, \infty) & c_{r,s} < \frac{1}{8} \\ n_0 \in \emptyset & c_{r,s} \geqslant \frac{1}{8}. \end{cases} \tag{5.5}$$

The solution of the second inequality is more complicated, but can be easily obtained with computer algebra system software (e.g., Maple™). In Table 5.1 we present some solutions for a number of values for r and s.

6

Multiple-core Based Parallel Steady-state Computation

As discussed in the previous chapter, statistical methods like the two-state Markov chain approach requires simulating the PBN under study for a certain length and the simulation speed is an important factor in the performance of these approaches. For large PBNs and long trajectories, a slow simulation speed could render these methods infeasible as well. A natural way to address this problem is to parallelise the simulation process. Recent improvements in the computing power and the general purpose graphics processing units (GPUs) enable the possibilities to massively parallelise this process. In this chapter, we propose a *trajectory-level parallelisation framework* to accelerate the computation of steady-state probabilities in large PBNs. This framework allows us to parallelise the simulation process with either multiple central processing unit (CPU) cores or multiple GPU cores. Parallelising with GPU cores requires special design of algorithms and/or data structure in order to maximize the computation power of GPU cores. However, these extra requirements usually lead to larger speedups since the number of available GPU cores is much more than that of the CPU cores. Hence, we focus on GPUs and explain how we apply the trajectory-level parallelisation framework with multiple GPU cores in this chapter.

The architecture of a GPU is very different from that of a CPU and the performance of a GPU-based program is highly related to how the synchronisation between cores is processed and how memory access is managed. Our framework reduces the time-consuming synchronisation cost by allowing each core to simulate one trajectory. Regarding the memory management, we contribute in four aspects. We first develop a dynamical data arrangement mechanism for handling different size PBNs with a GPU to maximise the computation efficiency on a GPU for relatively small-size PBNs. We then propose a specific way of storing predictor functions of a PBN and the state of the PBN in the GPU memory to reduce the memory consumption and to improve the access speed. Thirdly, we take special care of large and dense networks using our reorder-and-split method so that our parallelisation framework can not only handle large and dense network but also do it in an efficient way. Lastly, we develop a network reduction technique which can significantly reduce the unnecessary memory usage as well as the amount of required computations. We show with experiments that our GPU-accelerated parallelisation gains a speedup of more than two orders of magnitudes.

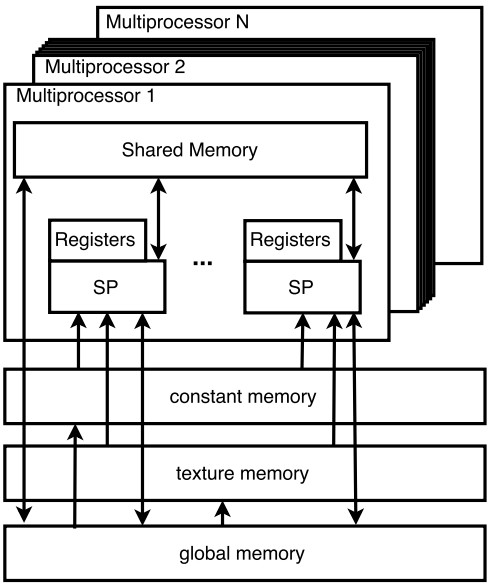


Figure 6.1: Architecture of a GPU.

6.1 GPU Architecture

We review the basics of GPU architecture and its programming approach, i.e., common unified device architecture (CUDA) released by NVIDIA.

At the physical hardware level, an NVIDIA GPU usually contains tens of streaming multiprocessors (SMs, also abbreviated as MPs), each containing a fixed number of streaming processors (SPs), fixed number of *registers*, and fast *shared memory* as illustrated in Figure 6.1, with N being the number of MPs.

Accessing registers and shared memory is fast, but the size of these two types of memory is very limited. In addition, a large size *global memory*, a small size *texture memory*, and *constant memory* are available outside the MPs. Global memory has a high bandwidth (128 bytes in our GPU), but also a high latency. Accessing global memory is usually orders of magnitude slower than accessing registers or shared memory. Constant memory and texture memory are memories of special type which can only store read-only data. Accessing constant memory is most efficient if all threads are accessing exactly the same data, while texture memory is better for dealing with random access. We refer to registers and shared memory as *fast memory*; global memory as *slow memory*; and constant memory and texture memory as *special memory*.

At the programming level, the programming interface CUDA is in fact an extension of C/C++. A segment of code to be run in a GPU is put into a function called a *kernel*. The kernels are then executed as a grid of blocks of threads. A thread is the finest granularity in a GPU and each thread can be viewed as a copy of the kernel. A block is a group of threads executed together in a batch. Each thread is executed in an SP and threads in a block can only be executed in one MP. One MP, however, can launch several blocks in parallel. Communications between threads in the same block are possible via shared memory. NVIDIA GPUs use a processor architecture called single instruction multiple thread (SIMT), i.e., a single instruction stream is executed via a group of 32 threads, called a *warp*. Threads within a warp are bounded together, i.e., they always execute the

same instruction. Therefore, branch divergence can occur within a warp: if one thread within a warp moves to the 'if' branch of an 'if-then-else' sentence and the others choose the 'else' branch, then actually all the 32 threads will "execute" both branches, i.e., the thread moving to the 'if' branch will wait for other threads when they execute the 'else' branch and vice versa. If both branches are long, then the performance penalty is huge. Therefore, branches should be avoided as much as possible in terms of performance. Moreover, the data accessing pattern of the threads in a warp should be taken care of as well. We consider the access pattern of shared memory and global memory in this work. Accessing shared memory is most efficient if all threads in a warp are fetching data in the same position or each thread is fetching data in a distinct position. Otherwise, the speed of accessing shared memory is reduced by the so-called *bank conflict*. Accessing global memory is most efficient if all threads in a warp are fetching data in a *coalesced* pattern, i.e., all threads in a warp are reading data in adjacent locations in global memory. In principle, the number of threads in a block should always be an integral multiple of the warp size due to the SIMT architecture; and the number of blocks should be an integral multiple of the number of MPs since each block can only be executed in one MP.

An important task for GPU programmer is to hide latency. This can be done via the following four ways:

1. increase the number of active warps;
2. reduce the access to global memory by caching the frequently accessed data in fast memory, or in constant memory or texture memory, if the access pattern is suitable;
3. reduce bank conflict of shared memory access;
4. coalesce accesses to the global memory to use the bandwidth more efficiently.

However, the above four methods often compete with one another due to the restrictions of the hardware resources. For example, using more shared memory would restrict the number of active blocks and hence the number of active warps is limited. Therefore, a trade-off between the use of fast memory and the number of threads has to be considered carefully. We discuss this problem and provide our solution to it in Section 6.2.2.

6.2 PBN Simulation in a GPU

In this section, we present how simulation of a PBN is performed in a GPU, while addressing the problems identified at the end of Section 6.1. More specifically, we discuss in Subsections 6.2.1–6.2.3 how in general the simulation of a PBN can be performed efficiently in a GPU; in Subsection 6.2.4, we take special care of large and dense PBNs, and demonstrate our reorder-and-split method for handling the large memory required in the dense network.

6.2.1 Trajectory-level Parallelisation

In general, there are two ways of parallelising the PBN simulation process. One way is to update all nodes synchronously, i.e., each GPU thread only updates one node of a PBN; the other way is to simulate multiple trajectories simultaneously. The first way requires synchronisation among the threads, which is time-consuming in the current GPU architecture. Besides, this way does not work for the asynchronous update mode

Algorithm 5 The Gelman & Rubin method

procedure GENERATECONVERGEDCHAINS(ω, ψ_0)
  $\psi := \psi_0$;
  Generate in parallel ω trajectories of length 2ψ;
  repeat
    chains($1 \dots \omega$,$1 \dots 2\psi$) := Extend all the ω trajectories to length 2ψ;
    for $i = 1..\omega$ **do**
      $\mu_i :=$ mean of the last ψ values of chain i;
      $s_i :=$ standard deviation of the last ψ values of chain i;
    end for
    $\mu := \frac{1}{\omega} \sum_{i=1}^{\omega} \mu_i$;
    $B := \frac{\psi}{\omega - 1} \sum_{i=1}^{\omega} (\mu_i - \mu)^2$; $W := \frac{1}{\omega} \sum_{i=1}^{\omega} s_i^2$; *//Between and within variance*
    $\hat{\sigma}^2 := (1 - \frac{1}{\psi})W + \frac{1}{\psi}B$; *//The variance of the stationary distribution*
    $\hat{R} := \sqrt{\hat{\sigma}^2 / W}$; *//Compute the potential scale reduction factor*
    $\psi := 2 \cdot \psi$;
  until $\hat{R}$ is close to 1
  return (chains,$\psi/2$);
end procedure

since only one node is updated at each time point. Therefore, in our implementation, we take the second way and simulate multiple trajectories concurrently. In order to use samples from multiple trajectories to compute the steady-state probabilities of a PBN, we propose to combine the Gelman & Rubin method [GR92] with the two-state Markov chain approach [RL92, MPY17].

The Gelman & Rubin method [GR92] is an approach for monitoring the convergence of multiple chains. It starts from simulating 2ψ steps of $\omega \geq 2$ independent Markov chains in parallel. The first ψ steps of each chain, known as the 'burn-in' period, are discarded from it. The last ψ elements of each chain are used to compute the within-chain (W) and between-chain (B) variance, which are used to estimate the variance of the steady state distribution ($\hat{\sigma}^2$). Next, the potential scale reduction factor $\hat{R}$ is computed with $\hat{\sigma}^2$. $\hat{R}$ indicates the convergence to the steady state distribution. The chains are considered as converged and the algorithm stops if $\hat{R}$ is close to 1; otherwise, ψ is doubled, the trajectories are extended, and $\hat{R}$ is recomputed. We list the steps of this approach in Algorithm 5. For further details of this method and the discussion on the choice of the initial states for the ω chains we refer to [GR92].

Once convergence is reached, the second halves of the chains are merged into one sample, and the two-state Markov chain approach is applied to estimate the required sample length L based on the merged sample. Since the convergence is assured, we propose to skip the iterative computation of the 'burn-in' period in the two-state Markov chain approach to maximise the speed-up. The stop criteria for the two-state Markov chain approach becomes that the estimated sample length L is not bigger than the size of the merged sample. If the stop criteria is not satisfied, the multiple chains are extended in parallel to provide a sample of required length. We describe this process in Algorithm 6 and refer to [MPY16d] for a detailed description of this combination. Note that merging is performed in a CPU and no synchronization is required. We show in Figure 6.2 the workflow for computing steady-state probabilities based on trajectory-level parallelisa-

Algorithm 6 The Parallelised two-state Markov chain approach

procedure ESTIMATEINPARALLEL($\omega, \psi_0, \epsilon, r, s$)
  $(chains, \psi) := generateConvergedChains(\omega, \psi_0)$;
  $n := 0$; $extend\_by := \psi$; $monitor :=$ FALSE; $ab\_sample :=$ NULL;
  repeat
    repeat
      $chains :=$ Extend in parallel each chain in chains by $extend\_by$;
      $sample := chains(1 \dots \omega, (n + \psi + 1) \dots (n + \psi + extended\_by))$;
      $ab\_sample :=$ abstract $sample$ and combine with ab_sample ;
      $n := n + extend\_by$; $sample\_size := \omega \cdot n$;
      Estimate α, β from ab_sample;
      Compute $\mathcal{N}$ as $n(\alpha, \beta)$ in Equation 5.2;
      $extend\_by := \lceil (sample\_size - N)/\omega \rceil$;
    until $extend\_by < 0$
    Compute $\mathcal{M}$ as $m(\alpha, \beta)$ in Equation 5.1;
    if $M \geq \psi$ **then**
      $extend\_by := \psi - \mathcal{M}$; $\psi := \mathcal{M}$; $monitor :=$ TRUE;
    end if
  until $monitor$
  Estimate the prob. of meta state 1 from ab_sample;
end procedure

tion.

Each blue box represents a kernel to be parallelised in a GPU. The first and second blue boxes perform the same task except that trajectories in the first blue box are abandoned while those in the second blue box are stored in global memory. This is due to the requirement of the Gelman & Rubin method [GR92] that only the second half samples are used for computing steady-state probabilities. Based on the last k samples simulated in the second blue box, the third blue box computes the *meta state* information required by the two-state Markov chain approach [MPY17]. The two-state Markov chain approach determines whether the samples are large enough based on the meta state information. If not enough, the last, fourth kernel is called repeatedly to extend samples; otherwise, the steady-state probability is computed.

The key part of the four kernels is the simulation process. We describe in Algorithm 7

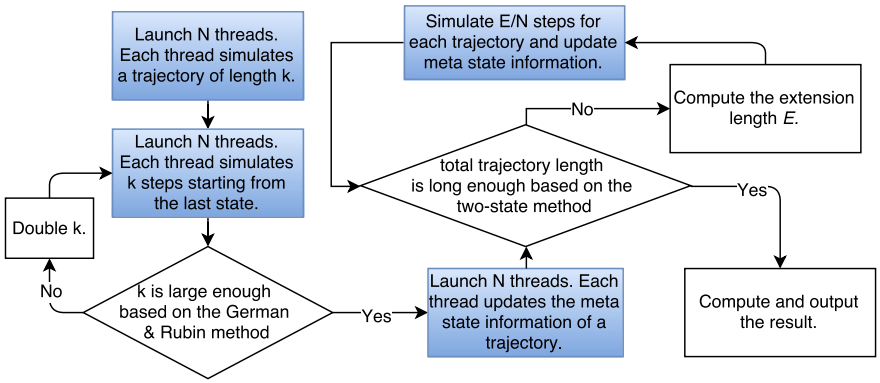


Figure 6.2: Workflow of steady-state analysis using trajectory-level parallelisation.

the process for simulating one step of a PBN in a GPU. The four inputs of this algorithm are respectively the number of nodes n, the Boolean functions F, the extra Boolean functions $extraF$, and the current state S. The extra Boolean functions are generated due to that we optimise the storage of Boolean functions and split them into two parts in order to save memory (see Section 6.2.3 for details). Due to this optimisation, an 'if'

Algorithm 7 Simulate one step of a PBN in a GPU

```
1:  procedure SIMULATEONESTEP(n, F, extraF, p, S)
2:      perturbed := false;
3:      for (i := 0; i < n; i++) do
4:          if rand() < p then perturbed := true; S[i/32] := S[i/32] ⊕ (1 ≪ (i%32));
5:          end if
6:      end for
7:      if perturbed then return S;
8:      else
9:          set array nextS to 0;
10:         for (i := 0; i < n; i++) do
11:             index := nextIndex(i);// Sample the Boolean function index for node i
12:             compute the entry of the Boolean function based on index and S;
13:             v := F[index];
14:             if entry > 31 then                          //entry starts with 0
15:                 get index of the Boolean function in extraF;    //See Section 6.2.3
16:                 v := extraF[index]; entry := entry%32;
17:             end if
18:             v := v ≫ entry; nextS[i/32] := nextS[i/32] | ((v&1) ≪ (i%32));
19:         end for
20:     end if
21:     S := nextS; return S.
22: end procedure
```

sentence (lines 14 to 17) has to be added. This 'if' sentence fetches the Boolean function stored in the second part ($extraF$). The probability that this sentence is executed is very small due to the way we split the Boolean functions and the time cost of executing this sentence is also very small. Therefore, by paying a small penalty in terms of computational time, we are able to store Boolean functions in fast memory and in total gain significant speedups.

6.2.2 Data Arrangement

As mentioned in Section 6.1, suitable strategy for hiding latency should be carefully considered for a GPU program. Since the simulation process requires accessing the PBN information (in a random way) in each simulation step and the latency cost for frequently accessing data in slow memory is huge, caching these information in fast and special memory results in a more efficient computation compared to allowing more active warps. Therefore, we first try to arrange all frequently accessed data in fast and special memory as much as possible; then, based on the remaining resources we calculate the optimal number of threads and blocks to be launched. Since the size of fast memory is limited and the memory required to store a PBN varies from PBN to PBN,

data	data type	stored in
random number generator	CUDA built in	registers
node number	integer	constant memory
perturbation rate	float	constant memory
cumulative number of functions	short array	constant memory
selection probabilities of functions	float array	constant memory
indices of positive nodes	integer array	constant memory
indices of negative nodes	integer array	constant memory
cumulative number of parent nodes	short array	shared memory
Boolean functions	integer array	shared memory
indices for extra Boolean functions	short array	shared memory
parent nodes indices for each function	short array	shared/texture memory
current state	integer array	registers/global memory
next state	integer array	registers/global memory

Table 6.1: Frequently accessed data arrangement.

a suitable data arrangement policy is necessary. In this section, we discuss how we dynamically arrange the data in different GPU memories for different PBNs.

In principle, frequently accessed data should be put in fast memory. We list all the frequently used data and how we arrange them in GPU memories in Table 6.1. As the size of the fast memory is limited and has different advantages for different data accessing modes, we save different data in different memories. Namely, the read-only data that are always or most likely accessed simultaneously by all threads in a warp, are put in constant memory; other read-only data are put in shared memory if possible; and the rest of the data are put in registers if possible. Since the memory required to store the frequently used data varies a lot from PBN to PBN, we propose to use a dynamic decision process to determine how to arrange some of the frequently accessed data, i.e., the data shown in the last four rows of Table 6.1. The dynamic process calculates the memory required to store all the data for a given PBN and determines where to put them based on their memory size. If the shared memory and registers are large enough, all the data will be stored in these two fast memories. Otherwise, they will be placed in the global memory. For the data stored in the global memory, we use two ways to speed up their access. One way is to use texture memory to speed up the access for read-only data, e.g., the parent node indices for each function. The other way is to optimise the data structure to allow a coalesced accessing pattern, e.g., the current state. We explain this in details in Section 6.2.3. This dynamical arrangement of data allows our program to explore the computational power of a GPU as much as possible, leading to larger speedups for relatively small sparse networks.

6.2.3 Data Optimisation

As mentioned in Section 6.1, a GPU usually has a very limited size of fast memory and the latency can vary significantly depending on how the memory is accessed, e.g., accessing shared memory with or without bank conflict. Therefore, we optimise the data structures for two important pieces of data, i.e., the Boolean functions (stored as truth tables) and the states of a PBN, to save space and to maximise the access speed.

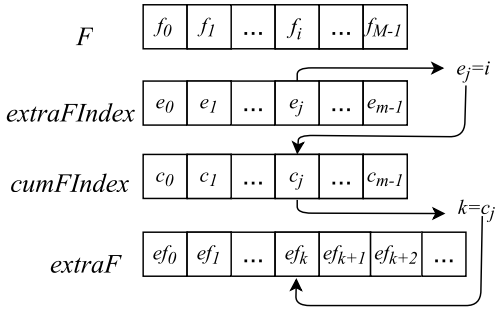


Figure 6.3: Demonstration of storing Boolean functions in integer arrays.

Optimisation on Boolean functions. A direct way to store a truth table is to use a Boolean array, which consumes one byte to store each element. Accessing an element of the truth table can be directly made by providing the index of the Boolean array. Instead, we propose to use a primitive 32-bit integer (4 bytes) type to store the truth table. Each bit of an integer stores one entry of the truth table and hence the memory usage can be reduced by 8 in maximum: 4 bytes compared to 32 bytes of a Boolean array. A 32-bit integer can store a truth table of at most 32 elements, corresponding to a Boolean function with max. 5 parent nodes. Since for real biological systems the number of parent nodes is usually small [LHSYH06], in most cases one integer is enough for storing the truth table of one Boolean function. In the case of a truth table with more than 32 elements, additional integers are needed. In order to save memory and quickly locate a specific truth table, we save the additional integers in a separate array. More precisely, we use a 32-bit integer array F of length M to store the truth tables for all the M Boolean functions and the ith ($i \in [0, M-1]$) element of F stores only the first 32 elements of the ith truth table. If the ith truth table contains more than 32 elements, the additional integers are stored in an extra integer array $extraF$. In addition, two index arrays $extraFIndex$ and $cumExtraFIndex$ are needed to store the index of the ith truth table in $extraF$. Each element of $extraFIndex$ stores one index value of the truth table which requires additional integers. The length of $extraFIndex$ is at most M. Each element of $cumExtraFIndex$ stores the cumulative number of additional required integers for all the truth tables whose indices are stored in $extraFIndex$.

As an example, we show how to store a truth table with 128 elements in Figure 6.3. We assume that this 128-element truth table is the ith one among all M truth tables and that it is the jth one among those m truth tables that require additional integers to store. Therefore, its first 32 (0-31th) elements are stored in the ith element of F and its index i is stored in the jth element of $extraFIndex$, denoted as e_j. The jth element of $cumExtraFIndex$, denoted as c_j, stores the total number of additional integers required to store the $j-1$ truth tables whose indices are stored in the first $j-1$ elements of $extraFIndex$. Let $cumExtraFIndex[j] = k$. The kth, $(k+1)$th, and $(k+2)$th elements of $extraF$ store the 32-127th elements of the ith truth table. After storing the truth tables in this way, accessing the tth element of the ith truth table can be performed in the following way. When $t \in [0, 31]$, $F[i]$ directly provides the information and when $t \in [32, 127]$, three steps are required: 1) search the array $extraFIndex$ to find the index j such that $extraFIndex[j]$ equals to i, 2) fetch the jth value of array $cumFIndex$ and let $k = cumFIndex[j]$, 3) the integer $extraF[k + (t-32)\%32]$ contains the required information. Since in most cases the number of parent nodes is very limited, the array $extraFIndex$ is very small. Hence, the search of the index j in the first step can be

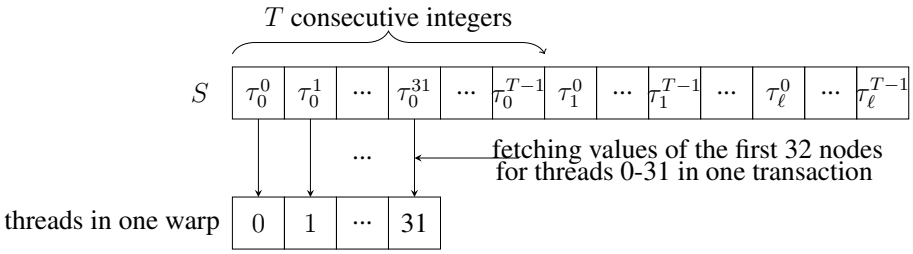


Figure 6.4: Storing states in one array and coalesced fetching for threads in one warp.

finished very quickly. In the rare case where the $extraFIndex$ array would be large, e.g., M is large and the length of $extraFIndex$ would be close to M, it is preferable to store $extraFindex$ as an array of length M and let $extraFindex[i]$ store the entry in $cumFIndex$ for the ith truth table so that the search phase of the first step is eliminated. The required memory for storing this truth table is reduced from 128 bytes (stored as Boolean arrays) to 20 bytes (6 integers to store the truth table and 2 shorts to store the index). In addition to saving memory, the above optimisation can also reduce the chances of bank conflict in shared memory due to the fact that accessing any entry of a truth table is performed by fetching only one integer in array F in most cases. Accessing the elements in $extraFIndex$ requires additional memory fetching; however, as mentioned before, the chance for such cases to happen is very small in a real-life PBN and the gained memory space and improved data fetching pattern can compensate for this penalty.

Optimisation on PBN states. The optimisation of the data structure for states is similar to that for Boolean functions, i.e., states are stored as integers and each bit of an integer represents the value of a node. Therefore, a PBN with n nodes requires $\lceil n/32 \rceil$ integers ($4 * \lceil n/32 \rceil$ bytes) to be stored, compared to n bytes when stored as a Boolean array. During the simulation process, the current state and the next state of a PBN have to be stored. As shown in Table 6.1, the states are put in registers whenever possible, i.e., when the number of nodes is smaller than 129. In the case of a PBN with nodes number equal to or larger than 129, the global memory has to be used due to the limited register size (shared memory are used to store other data and would not be large enough to store states in this case). To reduce the frequency of accessing global memory, one register (32 bits) is used to cache the integer that stores the values of 32 nodes. Updating of the 32 node values is performed via the register and stored in the global memory with a single access only once all the 32 node values are updated in the register. Moreover, states for all the threads are stored in one large integer array S in the global memory and we arrange the content of this array to allow for a coalesced accessing pattern. More specifically, starting from the 0th integer, every consecutive T integers store the values of 32 nodes in the T threads (assuming there are T threads in total). Figure 6.4 shows how to store states of a PBN with n nodes for all the T threads in an integer array S and how the 32 threads in the first warp fetch the first integer in a coalesced pattern. We denote τ_i^j as the ith integer to store values of 32 nodes for thread j and let $\ell = \lceil n/32 \rceil$. For threads in one warp, accessing the values of the same node can be performed via fetching the adjacent integers in the array S. This results in a coalesced accessing pattern of the global memory. Hence, all the 32 threads in one warp can fetch the data in a single data transaction.

6.2.4 Node-reordering for Large and Dense Networks

The above mentioned data arrangements and optimisation methods work quite well if the network is relatively sparse or small. However, when the network is both large and dense, the space required for storing the Boolean functions becomes so huge that they cannot be handled by the fast memories. Moreover, when the network is too dense, the number of parent nodes for each Boolean function is very likely to exceed 5. As a result, the Boolean function requires extra integers to be stored as discussed in Section 6.2.3, leading to inefficient access of Boolean functions.

To overcome the above mentioned problem, we propose a reorder-and-split method to handle the Boolean functions and their parent node indices for a large and dense network. The method consists of the following two steps. First, we reorder the nodes in an ascending order based on the number of their Boolean functions. Secondly, we split the ordered nodes into two parts. This split is based on the available amount of shared memory, i.e., the first part contains the first m nodes, where m is the maximum number of nodes whose Boolean functions and parent node indices can be stored in the fast memory. By reordering the nodes, we put the nodes with fewer Boolean functions in the fast memory. As a result, we can put more nodes in fast memory. Therefore, accessing slow memory in each simulation step is reduced. By splitting, we maximise the usage of the fast memory to store the Boolean functions so that the access to slow memory is minimised. Besides, since the chance that a Boolean function may have more than five parent nodes becomes higher in a dense network, it is very likely that $extraF$ is required to store a Boolean function if the Boolean functions are stored as discussed in Section 6.2.3. As GPU instructions in a warp are performed simultaneously, even if only one out of 32 threads is accessing the $extraF$, the other 31 threads have to wait for this access. Therefore, the advantage for the optimisation of storing Boolean functions in Section 6.2.3 disappears or even becomes a disadvantage. Instead of the above optimisation, we propose to store a Boolean function in consecutive elements of the same array when the elements of a Boolean function is more than 32. In this way, we only need two arrays to store the Boolean function information: one array F to store the Boolean functions and one array $startIndexF$ to store the starting index of each Boolean function.

6.3 Strongly Connected Component (SCC)-based Network Reduction

The set of states whose steady-state probability is to be computed is usually specified by specifying the values for a subset of node, referred to as the nodes of interest. The values of the remaining nodes are not considered when computing the steady-state probability. If a non-interest node is not an ancestor node of any node of interest, then such a node would not affect the values of the nodes of interest. We call such a node as an *irrelevant* node. Removing the irrelevant nodes will not affect the computation of steady-state probabilities if the perturbations of these irrelevant nodes are considered. In [MPY16b], a leaf-based network reduction method was proposed to remove the irrelevant nodes and hence to reduce the amount of computations required for PBN simulation. In this section, we present a strongly connected component (SCC)-based network reduction technique to improve the performance. Our method differs with the leaf-based network

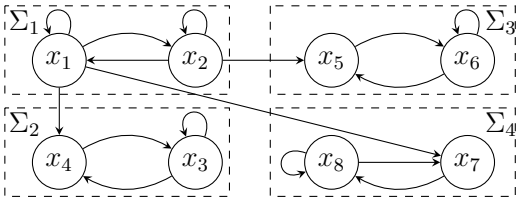


Figure 6.5: SCC-based network reduction.

reduction method by removing not only the leaf nodes, but also any other node that does not affect the nodes of interest. In other words, our method can remove all the irrelevant nodes. We first give the standard graph-theoretical definition of an SCC:

Definition 6.3.1 (SCC)**.** *Let $\mathcal{G}$ be a directed graph and $\mathcal{V}$ be its vertices. A strongly connected component (SCC) of $\mathcal{G}$ is a maximal set of vertices $C \subseteq \mathcal{V}$ such that for every pair of vertices u and v, there is a directed path from u to v and vice versa.*

We take a PBN G and convert it, i.e., its network structure, to a graph $\mathcal{G}$ by taking the nodes of G as the vertices in $\mathcal{G}$ and by drawing edges from the parent nodes to the child nodes in each of the Boolean functions. We then detect SCCs for the graph $\mathcal{G}$. By treating the SCCs as new vertices, we obtain a new graph which is in fact a directed acyclic graph (DAG). In this DAG, we keep only the following two types of SCCs: either an SCC that contains nodes of interest or an SCC that is an ancestor of a first type SCC. Nodes in the remaining SCCs are removed.

Example 6.3.1. *Figure 6.5 shows the graph of a BN with 8 nodes $x_1, x_2, \ldots, x_8$. The BN is decomposed into four SCCs $\Sigma_1, \Sigma_2, \Sigma_3$, and Σ_4. Assume only node x_7 is of interest, then the nodes in the SCC Σ_2 and Σ_3 can be removed since these two SCCs neither contain the nodes of interest nor are ancestors of an SCC with nodes of interest. Notably, the leaf-based network reduction method will not remove any node in this graph since there is no leaf node in this graph.*

Let us call the nodes removed by the above mentioned SCC-based network reduction technique as *redundant nodes*. Since those redundant nodes do not affect the nodes of interest, the simulation of the nodes of interest will not be affected in a PBN without perturbations after applying this network reduction technique. In the case of a PBN with perturbations, perturbations of the redundant nodes need to be considered. Updating states with Boolean functions will only be performed when there is no perturbation in both the redundant nodes and the non-redundant nodes. Perturbations of the redundant nodes can be checked in constant time irrespective of the number of redundant nodes as describe in Algorithm 9. The input p is the perturbation probability for each node and ℓ is the number of redundant nodes in the PBN. Then, the probability that no perturbation happens in all the redundant nodes is given by $t = (1-p)^{\ell}$. With the consideration of their perturbations, the redundant nodes can be removed without affecting the simulation of the non-redundant nodes also in a PBN with perturbations. Since the redundant nodes are not of interest, results of analyses performed on the simulated trajectories of the reduced network, i.e., containing only non-redundant nodes, will be the same as performed on trajectories of the original network, i.e., containing all the nodes.

Algorithm 8 Checking perturbations of redundant nodes in a PBN

```
1: procedure CHECKREDUNDANTNODES(p, ℓ)
2:     t = pow(1 − p, ℓ);
3:     if rand() > t then return true;
4:     else return false;
5:     end if
6: end procedure
```

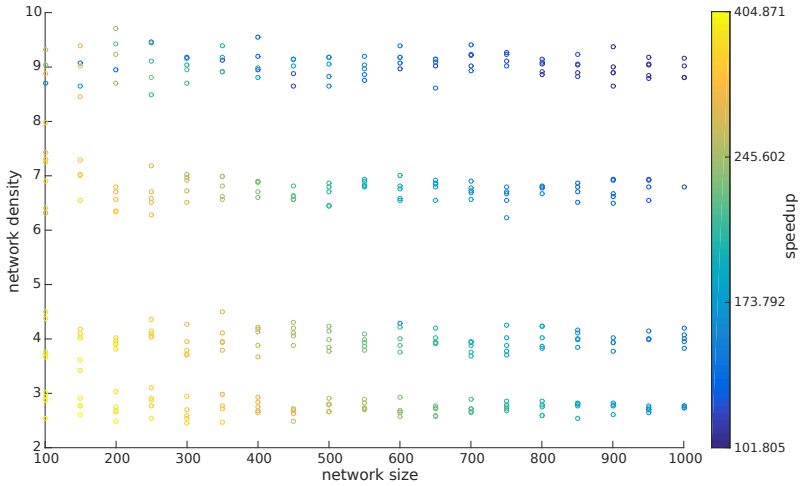


Figure 6.6: Speedups of GPU-accelerated steady-state computation.

6.4 Evaluation

We evaluate our GPU-based parallelisation framework for computing steady-state probabilities of PBNs on both randomly generated networks and on a real-life biological network. The evaluation contains three parts. We first evaluate the performance of our framework on randomly generated networks in Section 6.4.1. This evaluation includes the performance for relatively sparse networks as well as dense networks. Then, we demonstrate the performance of our SCC-based network reduction technique in Section 6.4.2. Lastly, we evaluate our framework on a real-life biological network. All the experiments are performed on high performance computing (HPC) machines, each of which contains a CPU of Intel Xeon E5-2680 v3 @ 2.5 GHz and an NVIDIA Tesla K80 Graphic Card with 2496 cores @824MHz. The program is written in a combination of both Java and C, and the initial and maximum Java virtual machine heap sizes are set to 4GB and 11.82GB, respectively. The C language is used to program operations on GPUs due to the fact that no suitable Java library is currently provided for programming operations on NVIDIA GPUs. When launching the GPU kernels, the kernel configurations (the number of threads and blocks) are dynamically determined as mentioned in Section 6.2.2.

6.4.1 Randomly Generated Networks

We first evaluate our framework on relatively sparse networks. This evaluation is performed on 380 PBNs, which are generated using the tool ASSA-PBN [MPY15, MPY16b]. The node numbers of these networks are form the set $\{100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000\}$. For each of the 380 networks, we compute one steady-state probability using both the sequential two-state Markov chain approach and our GPU-accelerated parallelisation framework. We set the three precision requirements of the two-state Markov chain approach, i.e., the confidence level s, the precision r, and the steady-state convergence parameter ϵ to 0.95, 5×10^{-5}, and 10^{-10} respectively. The computation time limit is set to 10 hours. In the end, we obtain 366 pairs of valid results. The 14 invalid pairs are due to the time out of the sequential version of the two-state Markov chain approach (the parallel version is not). Among the 366 results, 355 (96.99%) are comparable, i.e., the computed probabilities satisfy the specified precision requirement. This result meets our confidence level requirement.

We compute the speedups of the GPU-accelerated parallelisation framework with respect to the sequential two-state Markov chain approach for those 366 valid results with the formula $speedup = \frac{s_{pa}/t_{pa}}{s_{se}/t_{se}}$, where s_{pa} and t_{pa} are respectively the sample size and time cost of the parallelisation framework, and s_{se} and t_{se} are respectively the sample size and time cost of the sequential approach. The speedups are ploted in Figure 6.6. As can be seen from this figure, we obtain speedups approximately between 102- and 405-fold. There are some small gaps in the densities of the generated networks, e.g., there are no networks with density between 5 and 6. These gaps are due to the way the networks are randomly generated, i.e., one cannot force the ASSA-PBN tool to generate a PBN with a fixed density, but can only provide the following information to affect the density: the number of nodes, the maximum (minimum) number of functions for each node, and the maximum (minimum) number of parent nodes for each function. However, even with the gaps, the tendency of the changes of speedups with respect to densities can be well observed. In fact, this observation is similar to that for the network size. With the network size decreasing and the density decreasing, our GPU-accelerated parallelisation framework gains higher speedups. This is due to our dynamic way of arranging data for different size PBNs: data for relatively small[1] and sparse networks can be arranged in the fast memory alone.

To present the details on the obtained results, we select 8 pairs among the 366 results and show in Table 6.2 the computed probabilities, the sample size (in millions), and the time cost (in seconds) for computing the steady-state probabilities using both the sequential two-state Markov chain approach and the GPU-accelerated parallelisation framework. Note that the results of the two methods are shown in columns titled "s." and "–"; the columns titled "+" are used for demonstrating results of the network reduction technique discussed in the next section. The speedup of the GPU-accelerated parallelisation framework with respect to the sequential method is shown in the column titled "–". The two approaches generated comparable results using similar length of samples while our GPU-accelerated parallelisation framework shows speedups of more than two orders of magnitude. All detailed results for the 380 networks can be found at

[1] In fact all the networks used in this subsection should be called large-size PBNs since the state space of the network with the smallest size has $2^{100} \approx 10^{30}$ states.

# node	# re.	den.	probability s.	probability −	probability +	sample size (million) s.	sample size (million) −	sample size (million) +	time (s) s.	time (s) −	time (s) +	speedup −	speedup +
100	36	2.53	0.24409	0.24401	0.24407	350	367	367	2637.06	6.84	4.56	405	608
100	31	7.31	0.08831	0.08830	0.08830	150	152	151	939.70	4.24	2.89	224	328
400	131	7.14	0.20528	0.20528	0.20529	494	492	492	9825.44	62.89	38.75	155	252
400	148	2.75	0.12003	0.12002	0.12004	316	318	317	7615.72	26.77	15.26	286	501
700	231	7.08	0.13707	0.13708	0.13708	540	542	542	14758.95	120.60	80.18	123	185
700	269	2.64	0.05800	0.05794	0.05795	259	261	260	8567.52	39.27	22.59	220	381
1000	331	7.09	0.17795	0.17797	0.17797	988	998	993	28639.01	327.98	214.55	88	134
1000	388	2.73	0.14675	0.14673	0.14678	838	839	849	30626.44	184.44	108.24	166	287

Table 6.2: Speedups of GPU-accelerated steady-state computation of 8 randomly generated networks. "# re." is short for the number of redundant nodes; "s." is short for the sequential two-state Markov chain approach; "–" means the GPU-accelerated parallel approach without the network reduction technique applied; and "+" means the GPU-accelerated parallel approach with the network reduction technique applied.

# node	density	probability −	probability +	sample size (million) −	sample size (million) +	time (s) −	time (s) +	speedup
500	16.93	0.10273	0.10279	325	325	486.11	80.26	6.05
600	16.71	0.08988	0.08992	312	313	611.92	94.27	6.50
700	16.26	0.13131	0.13127	528	528	1308.04	190.34	6.87
800	16.46	0.09157	0.09154	439	440	1440.05	185.13	7.81
900	16.39	0.12738	0.12737	665	665	2664.85	321.59	8.28
1000	16.87	0.16530	0.16528	934	932	4789.25	512.42	9.32

Table 6.3: Speedups of GPU-accelerated steady-state computation with the reorder-and-split method applied. "+" means with the reorder-and split method applied; while "–" menas without the method applied.

`http://satoss.uni.lu/software/ASSA-PBN/benchmark/`.

We continue to demonstrate the performance of our framework on large and dense networks. Using the tool ASSA-PBN, we generate 30 large and dense networks whose nodes number are in the set $\{500, 600, 700, 800, 900, 1000\}$. In this evaluation, we compare how our reorder-and-split method as discussed in Section 6.2.4 performs compared to the cases when the reorder-and-split method is not applied. Therefore, for each of the 30 networks, we compute one steady-state probability using both the GPU-accelerated parallelisation framework with and without the reorder-and-split method applied. The three precision parameters were kept the same as in the previous evaluation. In the end, we get a pair of valid results for each of the 30 networks. We select 6 out of the 30 results and show them in Table 6.3. It is obvious from this table that applying our reorder-and-split method can improve the performance of the GPU-accelerated parallelisation framework by several times. Moreover, the improved performance of the reorder-and-split method increases with the number of nodes. This reflects the fact that the advantages of our reorder-and-split method become more pronounced with the network size increased.

steady-state	probability			sample size (million)			time (s)			speedup	
R C F	s.	–	+	s.	–	+	s.	–	+	–	+
0 1 1	0.003236	0.003237	0.003242	589.05	590.77	594.02	3866.04	9.28	5.89	417.81	661.98
1 1 1	0.990053	0.990046	0.990050	1809.27	1811.71	1811.44	11476.00	28.08	17.43	409.20	659.06
1 0 1	0.005592	0.005590	0.005586	1015.95	1021.07	1055.17	6662.26	15.89	10.13	421.47	682.98
1 1 0	0.001082	0.001080	0.001080	197.80	200.12	203.81	1281.45	3.27	1.96	396.60	673.29
* 1 1	0.993289	0.993288	0.993283	1222.83	1241.06	1235.52	7967.42	19.30	11.91	418.99	676.02
* 1 0	0.001082	0.001087	0.001090	197.29	206.37	201.08	1096.90	3.36	1.98	341.62	566.05
* 0 1	0.005614	0.005624	0.005619	1021.87	1039.35	1035.13	6725.25	16.17	9.95	422.98	684.60

Table 6.4: Speedups of GPU-accelerated steady-state computation of a real-life apoptosis network. "s." represents the sequential two-state Markov chain approach; "–" represents the GPU-accelerated parallel approach without applying the network reduction technique; and "+" represents the GPU-accelerated parallel approach with the network reduction technique applied.

6.4.2 Performance of SCC-based Network Reduction

In this section, we evaluate the performance of our SCC-based network reduction technique. We use the 8 selected networks shown in Table 6.2 to perform this evaluation. We calculate 8 steady-state probabilities of the 8 networks using the GPU-accelerated parallelisation framework with the SCC-based network reduction technique applied and show the results in columns with title "SCC.". The speedup of the parallelisation framework with the SCC-based network reduction technique applied with respect to the sequential two-state Markov chain approach is calculated based on the formula $speedup_{SCC} = \frac{s_{SCC}/t_{SCC}}{s_{se}/t_{se}}$, where s_{SCC} and t_{SCC} are respectively the sample size and time cost of the parallelisation framework with the SCC-based network reduction technique applied and s_{se} and t_{se} are respectively the sample size and time cost of the sequential approach. The results in Table 6.2 show that, by applying our SCC-based network reduction technique, the performance of our GPU-accelerated framework can be further improved and the improvement strongly depends on the percentage of redundant nodes.

6.4.3 An Apoptosis Network

We have analysed a PBN model of an apoptosis network using the sequential two-state Markov chain approach in [MPY17]. The apoptosis network was originally published in [SSV+09] as a BN model and cast into the PBN framework in [TMP+14]. The PBN model (as shown in Figure 3.8) contains 91 nodes and 107 Boolean functions.

The selection probabilities of the Boolean functions were fitted to experimental data in [TMP+14]. We took the 20 best fitted parameter sets and performed the influence analyses for them. Although we managed to finish this analysis in an affordable amount of time due to an efficient implementation of a sequential PBN simulator, the analysis was still very expensive in terms of computation time since the required trajectories were very long and we needed to compute steady-state probabilities for a number of different states.

In this work, we re-perform part of the influence analyses from [MPY17] using our

GPU-accelerated parallel two-state Markov chain approach. In the influence analysis, we consider the PBN with the best fitted values and we aim to compute the *long-term influences* on complex2 from each of its parent nodes: RIP-deubi, complex1, and FADD, in accordance with the definition in [SDKZ02]. In order to compute this long-term influence, seven different steady-state probabilities are required. We show in the first column of Table 6.4 the values of the nodes of interest for seven steady-state probabilities. The three numbers or "*" with two numbers respectively represent the values of the three genes RIP-deubi, complex1, and FADD: 0 represents active; 1 represents inactive; and "*" represents irrelevant. We compute the seven different steady-state probabilities using three different methods: the sequential two-state Markov chain approach, the GPU-accelerated parallelisation framework without the SCC-based network reduction technique applied, and the GPU-accelerated parallelisation framework with the SCC-based network reduction technique applied. In this network, there are 36 redundant nodes for all the seven steady-state probabilities. We show in Table 6.4 the computed steady-state probabilities, the sample size (in millions), the time cost (in seconds), and the speedups we obtain for this computation. The confidence level s, precision r, and the steady-state convergence parameter ϵ of this computation are set to 0.95, 5×10^{-6}, and 10^{-10} respectively. The density of the network is approximately 1.78. The three approaches compute comparable steady-state probabilities with similar trajectory lengths; while our two GPU-accelerated parallelisation frameworks reduce the time cost by approximately 400 and 600 times, respectively. The total time cost for computing the seven probabilities is reduced from about 11 hours to approximately 1.5 min. for the parallel framework without the network reduction technique applied and to less than 1 min. for the parallel framework with the network reduction technique applied.

6.5 Conclusion and Discussions

In this study, we have proposed a trajectory-level parallelisation framework to accelerate the computation of steady-state probabilities for large PBNs with the use of GPUs. Our work contributes in four aspects of maximising the performance of a GPU when computing the steady-state probabilities. First, we reduce the time consuming synchronisation cost between GPU cores by allowing each core to simulate all nodes of one trajectory. Secondly, we propose a dynamical data arrangement mechanism for handling different size PBNs with a GPU. Specially, we take care of both large and dense networks and develop a reorder-and-split method to handle it efficiently. Thirdly, we develop a specific way of storing predictor functions of a PBN and the state of the PBN in the GPU memory to save space and to accelerate the memory access. Last but not least, we have developed an SCC-based network reduction technique, leading to a great improvement in the computation speed of steady-state probabilities. We show with experiments that our GPU-based parallelisation gains a speedup of more than two orders of magnitudes. Evaluation on a real-life apoptosis network shows that our GPU-based parallelisation obtains a speedup of approximately 600 times.

In addition to multiple CPU or GPU cores, grid computing is another parallel technique that worth to explore. Indeed, many national wide grid computing centres have been established for this purpose, e.g., the CNGrid in China and the Grid5000 in France. Grid computing uses computer resources from multiple locations to reach a common goal. It can be seen as a special type of parallel computing, which relies on complete computers

connected together to a single network with a conventional network interface like Ethernet. This is different to parallel computing based on a conventional supercomputer, e.g., the multiple CPU computer, as a supercomputer is in fact one machine. In the literature, Grid computing has been expoled for the purpose of parameter estimation with PBNs [TMP+14]. Their parallel pipeline works in three steps: pre-processing, grid-based execution and post-processing. The pre-processing is run on a single machine and divides the parameter estimation task into sub-tasks. The sub-tasks can then be performed in the grid computers. After the execution finishes in the second step, the results are collected and processed in a single machine. There is a good potential to extend our parallel framework for steady-state computation in grid computing as well. Since our framework is based on trajectory-level parallelisation, the task of each machine in a grid can be simulating one trajectory. The simulated trajectories can then be collected and used for estimating the probabilities.

7

Structure-based Parallel Steady-state Computation

In the previous chapter, we discussed how to speedup the simulation of a PBN with the use of multiple cores. In this chapter, we continue to discuss the speedup techniques. Instead of using multiple cores, we make use of memory and propose a structure-based method to speed up the simulation process. The method is based on analysing the structure of a PBN and consists of two key ideas: first, it removes the unnecessary nodes in the network to reduce its size; secondly, it divides the nodes into groups and performs simulation for nodes in a group simultaneously. The grouping of nodes requires additional memory usage but result in a much faster simulation speed. We show with experiments that our structure-based method can significantly reduce the computation time for approximate steady-state analyses of large PBNs. To the best of our knowledge, our proposed method is the first one to apply structure-based analyses for speeding up the simulation of a PBN.

7.1 Structure-based Parallelisation

The simulation method described in the above section requires to check perturbations, make a selection and perform updating a node for n times in each step. In the case of large PBNs and huge trajectory (sample) size, the simulation time cost can become prohibitive. Intuitively, the simulation time can be reduced if the n-time operations can be speeded up, for which we propose two solutions. One is to perform *network reduction* such that the total number of nodes is reduced. The other is to perform *node-grouping* in order to parallelise the process for checking perturbations, making selections, and updating nodes. For the first solution, we analyse the PBN structure to identify those nodes that can be removed and remove them to reduce the network size; while for the second solution, we analyse the PBN structure to divide nodes into groups and perform the operations for nodes in a group simultaneously. We combine the two solutions together and refer to this simulation technique as *structure-based parallelisation*. We formalise the two solutions in the following three steps: the first solution is described in Step 1 and the second solution is described in Steps 2 and 3.

Step 1. Remove unnecessary nodes from the PBN.
Step 2. Parallelise the perturbation process.
Step 3. Parallelise updating a PBN state with predictor functions.

We describe these three steps in the following subsections.

Algorithm 9 Checking perturbations of leaf nodes in a PBN

```
1: procedure CHECKLEAFNODES(p, ℓ)
2:     t = pow(1 − p, ℓ);    // The probability that no perturbation happens in leaves
3:     if rand() > t then return true;
4:     else return false;
5:     end if
6: end procedure
```

7.1.1 Removing Unnecessary Nodes

We first identify those nodes that can be removed and perform network reduction. When simulating a PBN without perturbations, if a node does not affect any other node in the PBN, the states of all other nodes will not be affected after removing this node. If this node is *not of interest* of the analysis, e.g., we are not interested in analysing its steady-state, then this node is dispensable in a PBN without perturbations. We refer to such a dispensable node as a leaf node in a PBN and define it as follow:

Definition 7.1.1 (Leaf node)**.** *A node in a PBN is a* leaf node *(or* leaf *for short) if and only if either (1) it is not of interest and has no child nodes or (2) it is not of interest and has no other children after iteratively removing all its child nodes which are leaf nodes.*

According to the above definition, leaf nodes can be simply removed without affecting the simulation of the remaining nodes in a PBN without perturbations. In the case of a PBN with perturbations, perturbations in the leaf nodes need to be considered. Updating states with Boolean functions will only be performed when there is no perturbation in both the leaf nodes and the non-leaf nodes. Perturbations of the leaf nodes can be checked in constant time irrespective of the number of leaf nodes as describe in Algorithm 9. The input p is the perturbation probability for each node and ℓ is the number of leaf nodes in the PBN. Then, the probability that no perturbation happens in all the leaf nodes is given by $t = (1-p)^{\ell}$. With the consideration of their perturbations, the leaf nodes can be removed without affecting the simulation of the non-leaf nodes also in a PBN with perturbations. Since the leaves are not of interest, results of analyses performed on the simulated trajectories of the reduced network, i.e., containing only non-leaf nodes, will be the same as performed on trajectories of the original network, i.e., containing all the nodes.

7.1.2 Performing Perturbations in Parallel

The second step of our method speeds up the process of determining perturbations. Normally, perturbations are checked for nodes one by one. In order to speed up the simulation of a PBN, we perform perturbations for k nodes simultaneously instead of one by one. For those k nodes, there are 2^k different perturbation situations. We calculate the probability for each situation and construct an alias table based on the resulting distribution. With the alias table, we make a choice c among 2^k choices and perturb the corresponding nodes based on the choice. The choice c is an integer in $[0, 2^k)$ and for the whole network the perturbation can then be performed k nodes by k nodes using the logical bitwise *exclusive or* operation, denoted $|$. To save memory, the alias table can be reused for all the groups since the perturbation probability p for each node is the same.

It might happen that the number of nodes in the last perturbation round will be less than k nodes. Assume there is k' nodes in the last round and $k' < k$. For those k' nodes, we can reuse the same alias table to make the selection in order to save memory. After getting the choice c, we perform $c = c\&m$, where $\&$ is a bitwise *and* operation and m is a mask constructed by setting the first k' bits of m's binary representation to 1 and the remaining bits to 0.

Theorem 7.1.1. *The above process for determining perturbations for the last k' nodes guarantees that the probability for each of the k' nodes to be perturbed is still p.*

Proof. Without loss of generality, we assume that in the last k' nodes, t nodes should be perturbed and the positions of the t nodes are fixed. The probability for those t fixed nodes to be perturbed is $p^t(1-p)^{k'-t}$. When we make a selection from the alias table for k nodes, there are $2^{k-k'}$ different choices corresponding to the case that t fixed position nodes in the last k' nodes are perturbed. The sum of the probabilities of the $2^{k-k'}$ different choices is $[p^t(1-p)^{k'-t}] \cdot \sum_{i=0}^{k-k'} p^i(1-p)^{k-k'-i} = p^t(1-p)^{k'-t}$. □ □

We present the procedures for constructing groups and performing perturbations based on the groups in Algorithm 10, where n is the given number of nodes,[1] k is the maximum number of nodes that can be perturbed simultaneously and s is the PBN's current state which is represented by an integer. To obtain more balanced groups, k can be decreased in line 2. As perturbing one node equals to flipping one bit of s, perturbing nodes in a group is performed via a logical bitwise *exclusive or* operation, denoted $\oplus$ (see line 13 of Algorithm 10). Perturbing k nodes simultaneously requires 2^k double numbers to store the probabilities of 2^k different choices. The size of k is therefore restricted by the available memory.[2]

7.1.3 Updating Nodes in Parallel

The last step to speed up PBN simulation is to update a number of nodes simultaneously in accordance with their predictor functions. For this step, we need an initialisation process to divide the n nodes into m groups and construct *combined predictor functions* for each group. After this initialisation, we can select a combined predictor function for each group based on a sampled random number and apply this combined function to update the nodes in the group simultaneously.

We first describe how predictor functions of two nodes are combined. The combination of functions for more than two nodes can be performed iteratively. Let x_α and x_β be the two nodes to be considered. Their predictor functions are denoted as $F_\alpha = \{f_1^{(\alpha)}, f_2^{(\alpha)}, \ldots, f_{\ell(\alpha)}^{(\alpha)}\}$ and $F_\beta = \{f_1^{(\beta)}, f_2^{(\beta)}, \ldots, f_{\ell(\beta)}^{(\beta)}\}$. Further, the corresponding selection probability distributions are denoted as $C_\alpha = \{c_1^{(\alpha)}, c_2^{(\alpha)}, \ldots, c_{\ell(\alpha)}^{\alpha}\}$ and $C_\beta = \{c_1^{(\beta)}, c_2^{(\beta)}, \ldots, c_{\ell(\beta)}^{\beta}\}$. After the grouping, due to the assumed independence, the number of combined predictor functions is $\ell(\alpha) * \ell(\beta)$. We denote the set of combined predictor functions as $\bar{F}_{\alpha\beta} = \{f_1^{(\alpha)} \cdot f_1^{(\beta)}, f_1^{(\alpha)} \cdot f_2^{(\beta)}, \ldots, f_{\ell(\alpha)}^{(\alpha)} \cdot f_{\ell(\beta)}^{(\beta)}\}$, where for

[1] In our methods, it is clear that Step 2 and Step 3 are independent of Step 1. Thus, we consistently use n to denote the number of nodes in a PBN.

[2] For the experiments, we set k to 16 and k could be bigger as long as the memory allows. However, a larger k requires larger table to store the 2^k probabilities and the performance of a CPU drops when accessing an element of a much larger table due to the large cache miss rate.

Algorithm 10 The group perturbation algorithm

```
1: procedure PREPAREPERTURBATION(n, k)
2:     g = ⌈n/k⌉; k = ⌈n/g⌉; k' = n − k ∗ (g − 1);
3:     construct the alias table 𝔸_p; mask = 0; i = 0;
4:     repeat mask = mask | (1 ≪ i); i + +;
5:     until i = k';
6:     return [𝔸_p, mask];
7: end procedure
8: procedure PERTURBATION(𝔸_p, mask, s)
9:     perturbed = false;
10:    for (i = 0; i < g; i + +) do
11:        c = Next(𝔸_p);          //Next(𝔸_p) returns a random integer based on 𝔸_p
12:        if c ≠ 0 then
13:            s = s ⊕ (c ≪ (i ∗ k));          //Shift c to flip only the bits (nodes)
14:            perturbed = true;                // of the current group
15:        end if
16:    end for
17:    c = Next(𝔸_p) & mask;
18:    if c ≠ 0 then
19:        s = s ⊕ (c ≪ (i ∗ k)); perturbed = true;
20:    end if
21:    return [s, perturbed];
22: end procedure
```

$i \in [1, \ell(\alpha)]$ and $j \in [1, \ell(\beta)]$, $f_i^{(\alpha)} \cdot f_j^{(\beta)}$ is a combined predictor function that takes the input nodes of functions $f_i^{(\alpha)}$ and $f_j^{(\beta)}$ as its input and combines the Boolean output of functions $f_i^{(\alpha)}$ and $f_j^{(\beta)}$ into integers as output. The combined integers range in $[0, 3]$ and their 2-bit binary representations (from right to left) represent the values of nodes x_α and x_β. The selection probability for function $f_i^{(\alpha)} \cdot f_j^{(\beta)}$ is $c_i^{(\alpha)} * c_j^{(\beta)}$. It holds that $\sum_{i=1}^{\ell(\alpha)} \sum_{j=1}^{\ell(\beta)} c_i^{(\alpha)} * c_j^{(\beta)} = 1$. With the selection probabilities, we can compute the alias table for each group so that the selection of combined predictor function in each group can be performed in constant time.

We now describe how to divide the nodes into groups. Our aim is to have as few groups as possible so that the updating of all the nodes can be finished in as few rounds as possible. However, fewer groups lead to many more nodes in a group, which will result in a huge number of combined predictor functions in the group. Therefore, the number of groups has to been chosen properly so that the number of groups is as small as possible, while the combined predictor functions can be stored within the memory limit of the computer performing the simulation. Besides, nodes with only one predictor function should be considered separately since selections of predictor functions for those nodes are not needed. In the rest of this section, we first formulate the problem for dividing nodes with more than one predictor function and give our solution afterwards; then we discuss how to treat nodes with only one predictor function.

Problem formulation. Let S be a list of n items $\{\mu_1, \mu_2, \ldots, \mu_n\}$. For $i \in [1, n]$, item μ_i represents a node in a PBN with n nodes. Its weight is assigned by a function $\omega(\mu_i)$, which returns the number of predictor functions of node μ_i. We aim to find

Algorithm 11 The greedy algorithm

```
1: procedure FINDPARTITIONS(S, m)
2:     sort S with descending orders based on the weights of items in S;
3:     initialise A, an array of m lists;          // Initially, each A[i] is an empty list
4:     for (j = 0; j < S.size(); j++) do //S.size() returns the number of items in S
5:         among the m elements of A,    // The weight of A[i] is w_i = ∏_{μ_j ∈ A[i]} ω(μ_j)
6:             find the one with the smallest weight and add S[j] to it;
7:     end for
8:     return A;
9: end procedure
```

a minimum integer m to distribute the nodes into m groups such that the sum of the combined predictor functions numbers of the m groups will not exceed a memory limit θ. This is equivalent to finding a minimum m and an m-partition $S_1, S_2, \ldots, S_m$ of S, i.e., $S = S_1 \cup S_2 \cup \cdots \cup S_m$ and $S_k \cap S_\ell = \emptyset$ for $k, \ell \in \{1, 2, \ldots, m\}$, such that $\sum_{i=1}^{m} \left(\prod_{\mu_j \in S_i} \omega(\mu_j)\right) \leq \theta$.

Solution. The problem in fact has two outputs: an integer m and an m-partition. We first try to estimate a potential value of m, i.e., the lower bound of m that could lead to an m-partition of S which satisfies $\sum_{i=1}^{m} \left(\prod_{\mu_j \in S_i} \omega(\mu_j)\right) \leq \theta$. With this estimate, we then try to find an m-partition satisfying the above requirements.

Denote the *weight* of a sub-list S_i as w_i, where $w_i = \prod_{\mu_j \in S_i} \omega(\mu_j)$. The inequality in the problem description can be rewritten as $\sum_{i=1}^{m} w_i \leq \theta$. We first compute the minimum value of $\hat{m}$, denoted as $\hat{m}_{min}$, satisfying the following inequality:

$$\hat{m} \cdot \sqrt[\hat{m}]{\prod_{i=1}^{n} \omega(\mu_i)} \leq \theta. \tag{7.1}$$

Theorem 7.1.2. *$\hat{m}_{min}$ is the lower bound on m that allows a partition to satisfy $\sum_{i=1}^{m} w_i \leq \theta$.*

Proof. We proceed by showing that for any $k \in \{1, 2, \ldots, \hat{m}_{min} - 1\}$, $\hat{m}_{min} - k$ will make the inequality unsatisfied, i.e., $\sum_{i=1}^{\hat{m}_{min}-k} w_i^{'} > \theta$, where $w_i^{'}$ is the weight of the ith sub-list in an arbitrary partition of S into $\hat{m}_{min} - k$ sub-lists. Since $\hat{m}_{min}$ is the minimum value of $\hat{m}$ that satisfies Inequality (7.1), we have $(\hat{m}_{min} - k) \cdot \sqrt[(\hat{m}_{min}-k)]{\prod_{i=1}^{n} \omega(\mu_i)} > \theta$. Hence,

$$(\hat{m}_{min} - k) \cdot \sqrt[(\hat{m}_{min}-k)]{\prod_{i=1}^{\hat{m}_{min}-k} w_i^{'}} > \theta. \tag{7.2}$$

Based on the inequality relating arithmetic and geometric means, we have

$$\sum_{i=1}^{\hat{m}_{min}-k} w_i^{'} \geq (\hat{m}_{min} - k) \cdot \sqrt[(\hat{m}_{min}-k)]{\prod_{i=1}^{\hat{m}_{min}-k} w_i^{'}}. \tag{7.3}$$

Combining Inequality (7.2) with Inequality (7.3) gives $\sum_{i=1}^{\hat{m}_{min}-k} w_i^{'} > \theta$. □ □

Starting from the lower bound, we try to find a partition of S into m sub-lists that satisfies $\sum_{i=1}^{m} w_i \leq \theta$. Since the arithmetic and geometric means of non-negative real

Algorithm 12 Partition n nodes into groups.

```
1: procedure PARTITION(G, θ)
2:     compute lists S and S' based on G;          // S' contains nodes with 1 function
3:     compute the lower bound m̂ according to Inequality (7.1);  m = m̂;
4:     repeat
5:         A_1 = FINDPARTITIONS(S, m);
6:         sum = Σ_{i=1}^{m} ( Π_{μ_j ∈ A_1[i]} ω(μ_j) );        // Compute the sum of weights
7:         m = m + 1;
8:     until sum < θ;
9:     divide S' into A_2;   // Using modified Algorithm 11: in each iteration, a node is
10:                          // put in a list which shares most common parent nodes with this node
11:    merge A_1 and A_2 into A;
12:    return A;
13: end procedure
```

numbers are equal if and only if every number is the same, we get the heuristic that the weight of the m sub-lists should be as equal as possible so that the sum of the weights is as small as possible. Our problem then becomes similar to the NP-hard multi-way number partition problem: to divide a given set of integers into a collection of subsets, so that the sum of the numbers in each subset are as nearly equal as possible. We adapt the greedy algorithm (see Algorithm 11 for details) for solving the multi-way number partition problem, by modifying the sum to multiplication, in order to solve our partition problem.[3] If the m-partition we find satisfies the requirement $\sum_{i=1}^{m} w_i \leq \theta$, then we get a solution to our problem. Otherwise, we need to increase m by one and try to find a new m-partition. We repeat this process until the condition $\sum_{i=1}^{m} w_i \leq \theta$ is satisfied. The whole partition process for all the nodes is described in Algorithm 12.

Nodes with only one predictor function are treated in line 10. We divide such nodes into groups based on their parent nodes, i.e., we put nodes sharing the most common parents into the same group. In this way, the combined predictor function size can be as small as possible such that the limited memory can handle more nodes in a group. The number of nodes in a group is also restricted by the combined predictor function size, i.e., the number of parent nodes in this group.[4] The partition is performed with an algorithm similar to Algorithm 11. The difference is that in each iteration we always add a node into a group which shares most common parent nodes with this node.

7.1.4 The New Simulation Method

We describe our new method for simulating PBNs in Algorithm 13. The procedure PREPARATION describes the whole preparation process of the three steps (network reduction for Step 1, and node-grouping for Step 2 and Step 3). The three inputs of the procedure PREPARATION are the PBN network G, the memory limit θ, and the maxi-

[3] There exist other algorithms to solve the multi-way number partition problem and we choose the greedy algorithm for its efficiency.

[4] In our experiments, the maximum number of parent nodes in one group is set to 18. Similar to the value of k in Step 2, the number can be larger as long as the memory can handle. However, the penalty from large cache miss rate will diminish the benefits by having fewer groups when the number of parent nodes is too large.

mum number k of nodes that can be put in a group for perturbation. The PREPARATION procedure performs network reduction and node grouping. The reduced network and the grouped nodes information are then provided for the PARALLELSIMULATION procedure via seven parameters: $\mathbb{A}_p$ and $mask$ are the alias table and mask used for performing perturbations of non-leaf nodes as explained in Algorithm 10; l is the number of leaf nodes; p is the perturbation rate; $\mathbb{A}$ is an array containing the alias tables for predictor functions in all groups; F is an array containing predictor functions of all groups; and cum is an array storing the cumulative number of nodes in each group, i.e., $cum[0] = 0$ and $cum[i] = \sum_{j=0}^{i-1} \tau_j$ for $i \in [1, m]$, where m is the number of groups and τ_j is the number of nodes in group j. Procedure PARALLELSIMULATION simulates one step of a PBN by first checking perturbation and then updating PBNs with combined predictor functions. Perturbations for leaf nodes and non-leaf nodes have been explained in Algorithms 9 and 10. We now explain how nodes in a group are simultaneously updated with combined predictor function. It is performed via the following three steps: 1) a random combined predictor function is selected from F based on the alias table A; 2) the output of the combined predictor function is obtained according to the current state s; 3) the nodes in this group are updated based on the output of the combined predictor function. To save memory, states are stored as integers and updating a group of nodes is implemented via a logical bitwise *or* operation. To guarantee that the update is performed on the required nodes, a shift operation is needed on the output of the selected function (line 22). The number of bits to be shifted for the current group is in fact the cumulative number of nodes of all its previous groups, which is stored in the array cum.

7.2 Evaluation

The evaluation of our new simulation method is performed on both randomly generated networks and a real-life biological network. All the experiments are performed on high performance computing (HPC) machines, each of which contains a CPU of Intel Xeon X5675 @ 3.07 GHz. The program is written in Java and the initial and maximum Java virtual machine heap size is set to 4GB and 5.89GB, respectively.

7.2.1 Randomly Generated Networks

With the evaluation on randomly generated networks, we aim not only to show the efficiency of our method, but also to answer how much speedup our method is likely to provide for a given PBN.

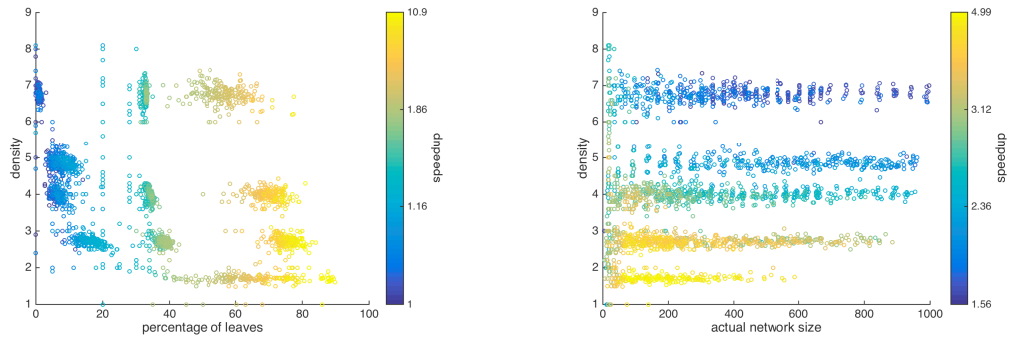
The first step of our new simulation method performs a network reduction technique, which is different from the node-grouping techniques in the later two steps. Therefore, we evaluate the contribution of the first step and the other two steps to the performance of our new simulation method separately. We consider the original simulation method as the reference method and we name it $Method_{ref}$. The simulation method applying the network reduction technique is referred to as $Method_{reduction}$ and the simulation method applying both the network reduction and node-grouping techniques as $Method_{new}$. $Method_{reduction}$ and $Method_{new}$ require pre-processing of the PBN under study, which leads to a certain computational overhead. However, the proportion of the pre-processing time in the whole computation decreases with the increase of the sample

Algorithm 13 Structure-based PBN simulation.

procedure PREPARATION(G, θ, k)
  perform network reduction for G and store the reduced network in G';
  get the number of nodes n and perturbation probability p from G;
  get the number of nodes n' from G'; $\ell = n - n'$;
  $[\mathbb{A}_p, mask]$ =PREPAREPERTURBATION(n', k);
  PA =PARTITION(G', θ);
  for each group in PA, compute its combined functions and put them as a list in array F,and compute its alias table in array $\mathbb{A}$;
  compute cum as $cum[0] = 0$ and $cum[i] = \sum_{j=0}^{i-1} \tau_j$ for $i \in [1, m]$, where m is the number of groups in PA and τ_j is the number of nodes in group j;
  return $[\mathbb{A}_p, mask, \ell, p, \mathbb{A}, F, cum]$;
end procedure
procedure PARALLELSIMULATION($\mathbb{A}_p, mask, \mathbb{A}, F, cum, \ell, p, s$)
  $[s, perturbed]$ =PERTURBATION($\mathbb{A}_p, mask, s$); *// Perturb by group*
  if *perturbed* || CHECKLEAFNODES(p, ℓ) **then** *// Check perturbations of leaves*
    return s;
  else $s' = 0$; $count = size(\mathbb{A})$; *// $size(\mathbb{A})$ is the # of elements in array $\mathbb{A}$*
    for $(i = 0; i < count - 1; i++)$ **do**
      $index = Next(\mathbb{A}[i])$; *// Select a random integer*
      $f = F[i].get(index)$; *// Obtain the predictor function at the given index*
      $v = f[s]$; *// $f[s]$ returns the integer output of f based on state s*
      $s' = s' \mid (v \ll cum[i])$; *// Bit shift v to update only nodes in*
    end for *// the current group*
  end if
  return s';
end procedure

size. In our evaluation, we first focus on comparisons without taking pre-processing into account to evaluate the maximum potential performance of our new simulation method; we then show how different sample sizes will affect the performance when pre-processing is considered.

How does our method perform? Intuitively, the speedup due to the network reduction technique is influenced by how much a network can be reduced and the performance of node-grouping is influenced by both the density and size of a given network. Hence, the evaluation is performed on a large number of randomly generated PBNs covering different types of networks. In total, we use 2307 randomly generated PBNs with different percentages of leaves ranging between 0% and 90%; different densities ranging between 1 and 8.1; and different network sizes from the set $\{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000\}$. The networks are generated randomly using the tool ASSA-PBN [MPY15], by providing the following information: the number of nodes, the maximum (minimum) number of predictor functions for the nodes, and the maximum (minimum) number of parent nodes for the predictor functions. Thus, the generation of these networks' density and percentage of leaves cannot be fully controlled. In other words, density and percentage of leaves for these 2307 PBNs are not uniformly distributed. We simulate 400 million steps for each of the 2307 PBNs with the three different simulation methods and compare their time costs. For the



(a) Simulation time of $Method_{reduction}$ over simulation time of $Method_{ref}$.

(b) Simulation time of $Method_{new}$ over simulation time of $Method_{reduction}$.

Figure 7.1: Speedups obtained with network reduction and node-grouping techniques. The pre-processing time is excluded from the analysis.

network reduction technique the speedups are calculated as the ratio between the time of $Method_{reduction}$ and the time of $Method_{ref}$, where the pre-processing time of the former method is excluded. The obtained speedups are between 1.00 and 10.90. For node-grouping, the speedups are calculated as the ratio between the time of $Method_{new}$ and the time of $Method_{reduction}$ without considering the required pre-processing times. We have obtained speedups between 1.56 and 4.99. We plot in Figure 7.1 the speedups of the network reduction and node-grouping techniques with respect to their related parameters. For the speedups achieved with network reduction, the related parameters are the percentage of leaves and the density. In fact, there is little influence from density to the speedup resulting from network reduction as the speedups do not change much with the different densities (see Figure 7.1a). The determinant factor is the percentage of leaves. The more leaves a PBN has, the more speedup we can obtain for the network. For the speedups obtained from node-grouping, the related parameters are the density and the network size after network reduction, i.e., the number of non-leave nodes. Based on Figure 7.1b, the speedup with node-grouping is mainly determined by the network density: a smaller network density could result in a larger speedup contributed from the node-grouping technique. This is mainly due to the fact that sparse network has a relatively small number of predictor functions in each node and therefore, the nodes will be partitioned into fewer groups. Moreover, while the performance of network reduction is largely influenced by the percentage of leaves, the node-grouping technique tends to provide a rather stable speedup. Even for large dense networks, the technique can reduce the time cost almost by half.

The combination of these two techniques results in speedups (time of $Method_{new}$ over time of $Method_{ref}$) between 1.74 and 41.92. We plot in Figure 7.2 the speedups in terms of the percentage of leaves and density. The figure shows a very good performance of our new method on sparse networks with large percentage of leaves.

What is the influence of sample size? We continue to evaluate the influence of sample size on our proposed new PBN simulation method. The pre-processing time for the network reduction step is relatively very small. Therefore, our evaluation focuses on the influence of the total pre-processing time of all the three steps on the speedup of $Method_{new}$ with respect to $Method_{ref}$. We select 9 representative PBNs from the above 2307 PBNs, with respect to their densities, percentages of leaves and the speedups we

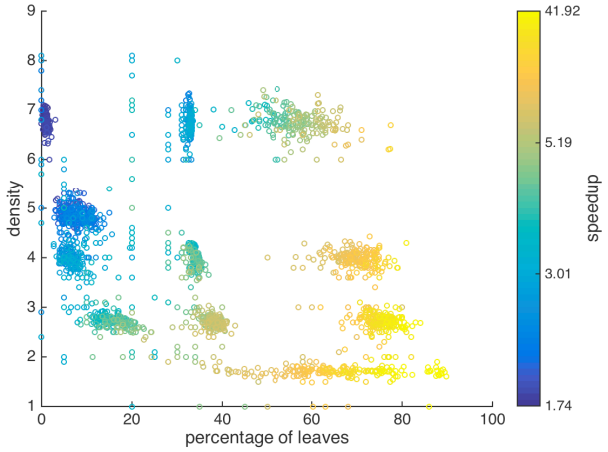


Figure 7.2: Speedups of $Method_{new}$ with respect to $Method_{ref}$.

network #	size	percentage of leaves	density	average p.-p. time (s)	speedup with different sample sizes (million) 1	10	100	400
1	900	1.11	6.72	28.12	0.65	1.49	1.71	1.73
2	950	0.84	6.96	32.35	0.59	1.47	1.73	1.75
3	1000	0.30	7.00	33.72	0.58	1.45	1.71	1.73
4	600	67.83	4.25	162.21	0.13	1.08	4.51	6.89
5	800	68.38	3.94	43.17	0.66	3.05	6.75	7.69
6	900	68.00	3.89	36.58	0.69	3.56	6.90	7.70
7	450	89.78	1.60	0.23	21.44	37.59	41.62	41.84
8	550	88.55	1.72	0.24	20.26	35.94	36.47	36.62
9	1000	89.10	1.75	1.08	10.04	31.83	35.09	37.19

Table 7.1: Influence of sample sizes on the speedups of $Method_{new}$ with respect to $Method_{ref}$. In the fifth column, p.-p. is short for pre-processing and the time unit is second.

have obtained. We simulate the 9 PBNs for different sample sizes using both $Method_{ref}$ and $Method_{new}$. We show the average pre-processing time of $Method_{new}$ and the obtained speedups with $Method_{new}$ (taking into account pre-processing time costs) with different sample sizes in Table 7.1. As expected, with the increase of the sample size, the influence of pre-processing time becomes smaller and the speedup increases. In fact, in some cases, the pre-processing time is relatively so small that its influence becomes negligible, e.g., for networks 7 and 8, where the sample size is equal or greater than 100 million. Moreover, often with a sample size larger than 10 million, the effort spent in pre-processing can be compensated by the saved sampling time (simulation speedup).

Performance prediction. To predict the speedup of our method for a given network, we apply regression techniques on the results of the 2307 PBNs to fit a prediction model. We use the normalised percentage of leaves and the network density as the predictor variables and the speedup of $Method_{new}$ with respect to $Method_{ref}$ as the response variable in the regression model. We do not consider network size as based on the plotted figures it does not directly affect the speedup. In the end, we obtain a polynomial

#	$Method_{ref}$ sample size (million)	time (m)	probability	$Method_{new}$ p.-p. time (s)	sample size (million)	total time (m)	probability	speedup
1	147.50	9.51	0.003243	4.57	147.82	1.05	0.003236	9.09
2	452.35	28.65	0.990049	3.10	452.25	2.79	0.990058	10.28
3	253.85	14.88	0.005583	3.42	253.99	1.74	0.005587	8.54
4	49.52	2.96	0.001087	3.38	50.39	0.36	0.001078	8.31
5	315.06	17.73	0.993293	4.40	305.43	2.05	0.993298	8.39
6	62.22	3.69	0.001088	3.13	50.28	0.39	0.001087	7.67
7	255.88	16.74	0.005621	4.01	256.61	1.70	0.005623	9.88

Table 7.2: Performance of $Method_{ref}$ and $Method_{new}$ on an apoptosis network.

regression model shown in Equation (7.4), which can fit 90.9% of the data:

$$y = b_1 + b_2 * x_1 + b_3 * x_1^2 + b_4 * x_2 + b_5 * x_2^2, \tag{7.4}$$

where $[b_1, b_2, b_3, b_4, b_5] = [2.89, 2.71, 2.40, -1.65, 0.71]$, y represents the speedup, x_1 represents the percentage of leaves and x_2 represents the network density. The result of a 10-fold cross-validation of this model supports this prediction rate. Hence, we believe this model does not overfit the given data. Based on this model, we can predict how much speedup is likely to be obtained with our proposed method for a given PBN.

7.2.2 An Apoptosis Network

In this section, we evaluate our method on a real-life biological network, i.e., an apoptosis network of 91 nodes [SSV+09]. This network has a density of 1.78 and 37.5% of the nodes are leaves, which is suitable for applying our method to gain speedups. The network has been analysed in [MPY17]. In one of the analyses, i.e., the long-term influences [SDKZ02] on complex2 from each of its parent nodes: RIP-deubi, complex1, and FADD, seven steady-state probabilities of the network need to be computed. In this evaluation, we compute the seven steady-state probabilities using our proposed structure-based simulation method ($Method_{new}$) and compare it with the original simulation method ($Method_{ref}$). The precision and confidence level of all the computations, as required by the two-state Markov chain approach [RL92], are set to 10^{-5} and 0.95, respectively. The results of this computation are shown in Table 7.2. The computed probabilities using both methods are comparable, i.e., for the same set of states, the differences of the computed probabilities are within the precision requirements. The sample sizes required by both methods for computing the same steady-state probabilities are very close to each other. Note that the speedups are computed based on the accurate data, which are slightly different from the truncated and rounded data shown in Table 7.2. We have obtained speedups ($Method_{new}$ with respect to $Method_{ref}$) between 7.67 and 10.28 for computing those seven probabilities. In total, the time cost is reduced from 1.5 hours to about 10 minutes.

7.3 Conclusion

We propose a structure-based method for speeding up simulations of PBNs. Using network reduction and node-grouping techniques, our method can significantly improve the simulation speed of PBNs. We show with experiments that our method is especially efficient in the case of analysing sparse networks with a large number of leaf nodes.

The node-grouping technique gains speedups by using more memory. Theoretically, as long as the memory can handle, the group number can be made as small as possible. However, this causes two issues in practice. First, the pre-processing time increases dramatically with the group number decreasing. Second, the performance of the method drops a lot when operating on large memories due to the increase of cache miss rate. Therefore, in our experiments we do not explore all the available memory to maximise the groups. Reducing the pre-processing time cost and the cache miss rate would be two future works to further improve the performance of our method. We plan to apply our method for the analysis of real-life large biological networks.

Part III

The Tool for Steady-state Analysis

8

ASSA-PBN: a Software Tool for Probabilistic Boolean Networks

After providing the theoretical solutions for the two research problems in Chapters 3-7, we now present ASSA-PBN, a toolbox in which we have implemented the above discussed algorithms and heuristics. ASSA-PBN is a software toolbox designed for modelling, simulation and analysis of PBNs. For modelling, ASSA-PBN has provided a high-level PBN description file which can easily modelling a real-life biological network. Besides, it also supports loading and saving PBNs in Matlab-PBN-toolbox format. In addition, users can generate random PBNs according to their requirements. In terms of simulation, ASSA-PBN provides an efficient simulator, which can overcome the network size limitation. The analyser module of ASSA-PBN provides steady-state probability computation, parameter estimation, long-run influence analysis, long-run sensitivity analysis, and computation and visualisation of one-parameter profile likelihoods to explore the characteristics of PBNs. Computation of steady-state probabilities plays a major role among all the analysis methods, since it is the basis of other methods. In particular, ASSA-PBN implements numerical methods for exact analysis of small PBNs and statistical methods for approximate analysis of large PBNs. The current version supports three different statistical methods, i.e. the perfect simulation algorithm [VM04], the two-state Markov chain approach [RL92, MPY17], and the Skart method [TWLS08]. To speed up the computation process, additionally ASSA-PBN provides three parallel techniques: structure-based parallelisation, CPU-based parallelisation and GPU-based parallelisation. This makes ASSA-PBN capable of handling the steady-state computations that require generation of long trajectories consisting of billions of states. Experimental results show that ASSA-PBN is capable of analysing PBNs with thousands of nodes.

The usability of existing methods/tools for PBNs, such as optPBN [TMP+14] and the BN/PBN Toolbox for MATLAB® created by Lähdesmäki and Shmulevich [LS09], is restricted by the network size. For instance, optPBN can only analyse parts of the 96-node PBN due to its computational efficiency issues, leaving some hypotheses regarding the network characteristics unverified [TMP+14]. The PBN Matlab toolbox applies numerical methods for computing steady-state probabilities for PBNs (see more detailed discussions in Section 8.4), which are not scalable and are impracticable for the analysis of large biological networks. ASSA-PBN [MPY17, MPY15] is however, capable of solving these problems with several efficient methods for analysing large PBNs as discussed in previous chapters.

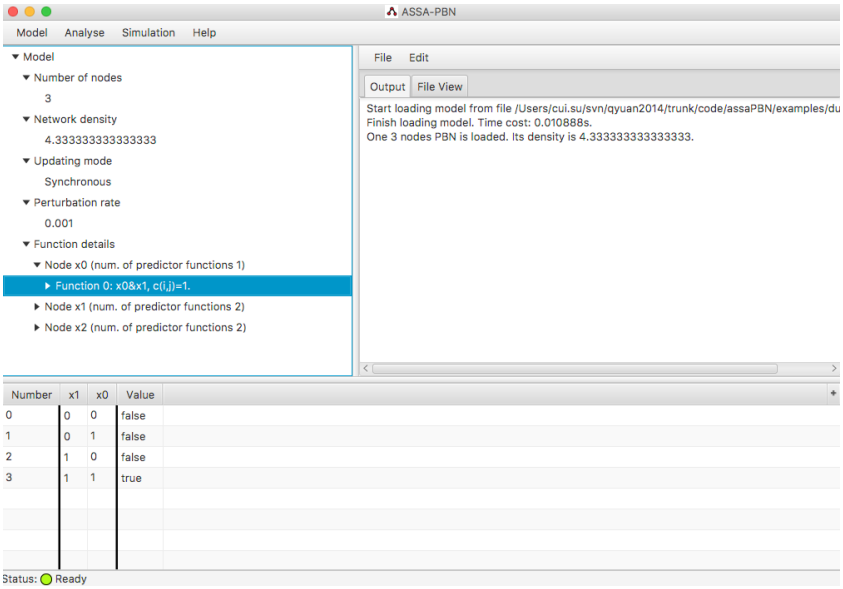


Figure 8.1: Interface after loading a PBN into ASSA-PBN.

8.1 Toolbox Architecture

ASSA-PBN provides both a graphical user interface (GUI) and a command line interface. As shown in Figure 8.1, the interface is divided into three parts: the menu bar, three panels and the status bar. The panels are used to display PBN specification and the results of simulation and analysis.

The architecture of ASSA-PBN consists of three main modules, i.e. a modeller, a simulator, and an analyser, as shown in Figure 8.2. The three modules allow users to construct, simulate and analyse a PBN model, respectively.

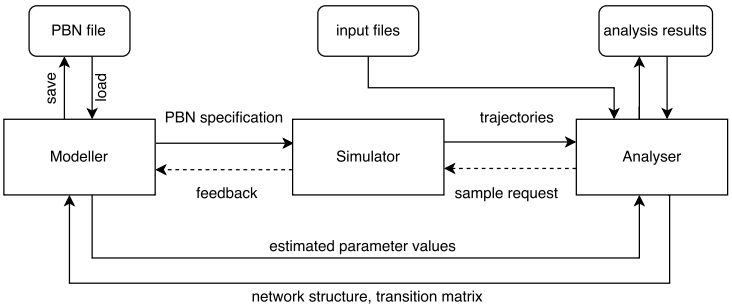


Figure 8.2: The architecture of ASSA-PBN.

The main function of the modeller is to load a PBN model from a given input file and to create its internal representation in memory or to save a PBN model in an output file. In addition, the modeller facilitates the generation of a random PBN in accordance with a user's requirements. ASSA-PBN supports the loading and saving of PBN models in either the ASSA-PBN format or the BN/PBN Toolbox format.

The simulator produces trajectories (also called samples) of the loaded/generated PBN. Since this process is not based on the transition matrix of the loaded PBN, it does not suffer from the state-space explosion problem even for large PBNs. The produced tra-

jectories are presented to the modeller and/or serve as input for further analysis.

The analyser provides several functionalities for analysis of PBNs. Its core function is the computation of the steady-state probability of a subset of states which is specified in a property file. The computation can be performed in either a numerical manner, suitable for small PBNs, or in a statistical manner, appropriate for large PBNs. The numerical methods are based on the state transition matrix supplied by the modeller; while the statistical methods take as input trajectories produced by the simulator. The statistical methods operate in an iterative way and extensions of the trajectories are requested from the simulator in each iteration until the sample size is large enough to obtain results satisfying the specified precision requirements.

Steady-state probabilities can be utilised by the analyser to estimate selection probability parameters of a PBN model to make it fit experimental steady-state measurements. Optimised parameter values are further returned to the modeller. Moreover, the analyser facilitates the evaluation of long-run influences and sensitivities of the PBN. The analysis results can be used to verify and optimise the original model. The details of the three modules are described in the next three sections.

8.2 Modeller

The modeller of ASSA-PBN provides two ways for model construction: loading a PBN from a model definition file or generating a random PBN (e.g., for benchmarking and testing purposes) complying with users' requirements [MPY15].

Users can load a PBN from a file either in ASSA-PBN model definition format or BN/PBN Toolbox format. The ASSA-PBN model definition file provides various information on a PBN , including the update mode, the number of nodes, the Boolean functions for each node, the selection probability for each predictor function and the perturbation rate. ASSA-PBN supports the synchronous update mode and six types of asynchronous update modes. A predictor function can be specified in two ways: either in the form of a truth table or with a high-level PBN definition format, where the predictor function is given as its semantic logical formula. The latter makes the node update semantics more explicit and evident. The GUI of ASSA-PBN also provides means to explore and inspect the information on predictor functions, which allows users to check the details of the model structure and semantics.

Figure 8.1 demonstrates the interface showing that a PBN has been loaded into ASSA-PBN. The top-left panel displays general information on the loaded PBN, including its number of nodes, network density, updating mode, perturbation rate, and details on its predictor functions. The function details are shown as a tree structure in the panel. After selecting a predictor function, its truth table is shown in the bottom panel. The top-right panel additionally contains information on the the PBN model loading time.

When generating a random model, users provide the node number as well as some optional parameters including the maximum (minimum) number of predictor functions for nodes, the maximum (minimum) number of parent nodes for the predictor functions and modify the default value for the perturbation rate.

Additionally, ASSA-PBN allows the user to disable perturbations for specified nodes. This feature is needed for the modelling of cellular systems where environmental con-

ditions are kept constant, i.e. model input nodes should have fixed values, or modelling of mutants where certain nodes are inactivated or over-activated. This feature should however be used with care as it may render the PBN's underlying DTMC to become non-ergodic.

ASSA-PBN stores the PBN model in memory with use of dedicated data structures which facilitate efficient simulation.

8.3 Simulator

At present, statistical approaches are practically the only viable option for the analysis of long-run dynamics of large PBNs due to the infamous state space explosion problem. Such methods however necessitate generation of long trajectories. Therefore, the simulator module is designed to efficiently produce trajectories with the given initial states (either provided by the user or randomly set by ASSA-PBN).

The simulation can be performed with a number of different update modes supported by ASSA-PBN , including synchronous, asynchronous ROG, asynchronous RMG and other asynchronous modes. When simulating the next state of a PBN, the simulator first checks whether perturbation should be applied. If yes, the simulator updates the current state according to the perturbation. Otherwise, the simulator updates the state of certain nodes following the update mode. For the synchronous update mode, every node in the PBN is updated: a predictor function of each node is chosen according to its selection probability and the state of the node is updated with the chosen predictor function. For the asynchronous update mode, depending on which submode is chosen, the states of randomly selected nodes are updated. Notice that the state transition matrix is not needed in the simulation process, which makes ASSA-PBN capable of managing large PBNs. The visualisation of the simulation result is supported in ASSA-PBN. Time-course evolution of the values of selected nodes can be displayed.

Figure 8.3 shows the simulator interface. Users can set trajectory length and the initial state. For example, for a three-node PBN the initial state $(x_2 = 0, x_1 = 1, x_0 = 1)$ can be set by typing the space-delimited sequence $0\ 1\ 1$ in the 'Initial State' field. If the checkboxes for update modes are left unchecked, the update mode defined in the definition file is used. By checking the 'Show the simulation graph' box, a graph view of the simulation results is shown in a separate window once a trajectory has been generated.

As mentioned above, the analysis of the long-run dynamics of large PBNs often requires generation of long trajectories. Therefore, efficiency of the simulation is crucial to enable the analysis of large biological networks in a reasonable computational time. To achieve this goal, ASSA-PBN offers several ways to speed up the simulation, including the alias method [Wal77], the multiple-core based parallelisation technique (see Chapter 6), and the structure-based parallelisation technique (see Chapter 7). Note that the structure-based parallelisation technique only works for synchronous PBNs and currently the GPU-based parallelisation technique is only implemented for synchronous PBNs as well.

The consecutive state is obtained by applying properly selected predictor functions to each of the nodes in a PBN. For efficiency reasons, the selection is performed with the alias method. The simulator of ASSA-PBN provides two modes: the *global alias mode*

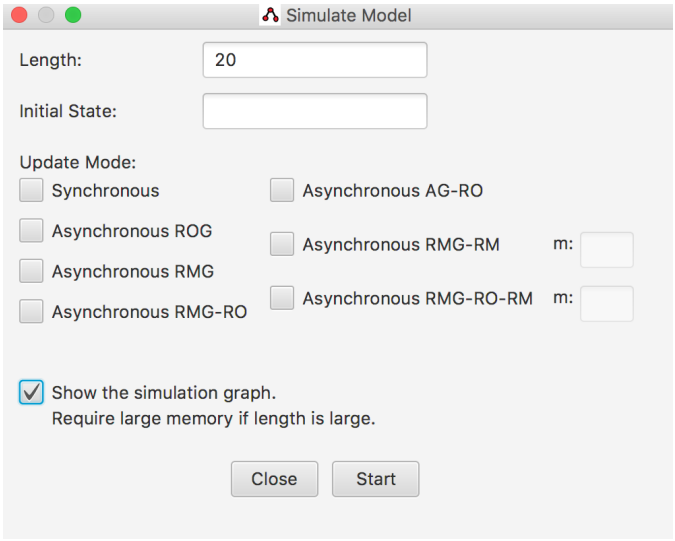


Figure 8.3: Interface of the simulator window in ASSA-PBN.

and the *local alias mode*. In the global mode, a single alias table for the joint probability distribution on the all combinations of predictor functions for all PBN nodes is built. In the local mode, individual alias tables for the distributions on predictor functions for each node are constructed. In both cases it is implicitly assumed that the predictor functions for individual nodes are selected independently of each other. In the global mode, two random numbers are needed to perform predictor functions selection for all the nodes at once, while in the local mode the number of random numbers needed is twice the number of nodes. Compared to the local mode, the simulation with the global mode is faster, but more expensive in terms of memory usage for storage of the large alias table. As a consequence, in general the local mode is recommended for large networks.

Currently, the parallel techniques are only available for the analyser module of ASSA-PBN. Since the computation of steady-state probabilities usually requires long trajectories, the main purpose of the three parallel techniques is to speed up the simulation process greatly. The simulator module is mainly used to generate short trajectories for users to visualise the simulation result and check the correctness of the PBN.

8.4 Analyser

The analyser of ASSA-PBN provides four main functionalities: computation of steady-state probabilities for specified subsets of states, computation of long-run influences and various types of long-run sensitivities, parameter estimation, and computation and visualisation of one-parameter profile likelihoods. Computation of steady-state probabilities forms the basis for the other three tasks. The computed steady-state probabilities and the long-run influences/sensitivities provide insight into the characteristics of a given PBN model, which in turn helps to gain a better understanding of the biological system under study. Parameter estimation optimises the values of estimated parameters of the constructed PBN model to fit steady-state experimental measurements. Finally, one-parameter profile likelihoods provide insight into the structural and practical identifiability of considered model parameters.

8.4.1 Computation of Steady-state Probabilities

In the following, we first describe a few methods that ASSA-PBN implements for computing steady-state probabilities of PBNs. Then, we briefly present three parallel techniques that were recently developed to improve the efficiency of such computations.

Implemented methods. The analyser of ASSA-PBN provides two iterative numerical methods for exact, up to a pre-specified convergence criterion and numerical precision, computation of the steady-state distributions of PBNs, namely the Jacobi method and the Gauss-Seidel method. Moreover, the analyser provides three statistical methods for computation of steady-state probabilities: the perfect simulation algorithm [PW96], the Skart method [TWLS08], and the two-state Markov chain approach [RL92]. Starting from a random initial distribution on the state space of a PBN, iterative numerical methods compute the steady-state distribution by iteratively performing matrix-vector multiplication with use of the state transition matrix. Once the required accuracy threshold is reached, the iterative process terminates and the final steady-state probability distribution is returned. Since iterative numerical methods are based on the state transition matrix, they are expensive both in term of memory and computational time consumption, hence applicable only to *small-size PBNs* (often with less than 20 nodes).

The perfect simulation algorithm [PW96] draws independent samples which are distributed exactly in accordance with the steady-state distribution of a DTMC. In consequence, it avoids problems related to the convergence to the steady-state distribution or non-zero correlation between consecutive states in a trajectory. The current implementation is in-line with the 'Functional backward-coupling simulation with aliasing' algorithm provided in [VM04]. This algorithm shortens the average coupling time significantly when only a subset of states is of interest. Nevertheless, due to the nature of this method, each state of the state space needs to be considered at each step of the coupling scheme. Therefore, this approach only suits *medium-size PBNs* and *large PBNs* are out of its scope. Unfortunately, since PBNs with perturbations are non-monotone systems, the very efficient monotone version of perfect simulation [EP09], in which only a small subset of the whole state space needs to be considered, is of no use in this context.

We refer to Chapter 5 for detailed description of the Skart method and the two-state Markov chain approach.

Parallel computation. To produce trajectories of large synchronous PBNs, the simulator of ASSA-PBN needs to check perturbations, select Boolean functions, and perform state update for n nodes in each step. The simulation time cost can be prohibitive in the case of large PBNs and huge trajectory (sample) size. Therefore, two different techniques to speed up the generation of very long trajectories were proposed and implemented in ASSA-PBN. We refer to Chapters 6 and 7 for detailed description of the two techniques.

Figure 8.4 shows the interface for computing steady-state probabilities with the two-state Markov chain approach in ASSA-PBN. The precision and confidence level are two required parameters of the two-state Markov chain approach. The steady-state convergence parameter ϵ is in the current implementation fixed to 10^{10}. If "Global Alias" is checked, the simulation will be performed with the global alias mode as described in Section 8.3. Checking this box could potentially increase the speed of the two-state

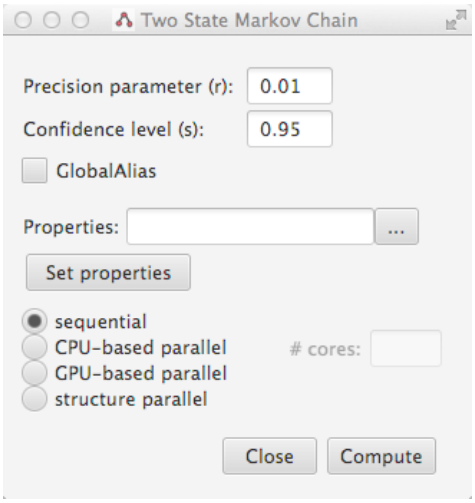


Figure 8.4: Interface of computing steady-state probabilities with the two-state Markov chain approach in ASSA-PBN.

Markov chain approach at the cost of higher memory consumption. "Properties" field allows to provide a file with the specification of a subset of states for which the steady-state probability is to be computed. The four radio selections are used to specify how the simulation should be performed: either in a sequential way or with the use of one of the two different parallel techniques mentioned above. Note that the multiple-core based parallel technique is further divided into two: CPU-based parallel and GPU-based parallel. If "CPU-based parallel" is selected, the gray text field "# cores" will become available and filled with the number of cores on the computer used.

8.4.2 Parameter Estimation

A common task for building a model for a real-life biological system is to optimise the parameters of the model to make it fit experimental data. The analyser provides the parameter estimation functionality to support this task for PBN models. A few algorithms [KE95, MMB03] have been proposed in the literature for parameter estimation of biological systems. ASSA-PBN implements the *particle swarm optimisation (PSO)* and *differential evolution (DE)* algorithms to optimise the specified parameters of PBN models.

PSO is an iterative method to optimise parameters of a model. The set of parameters to be optimised is called a particle. PSO solves the optimisation problem by moving a population of candidate particles around in the search space and by updating the position and speed of the particles according to the considered fitness function. In ASSA-PBN, we use the *mean square error (MSE)*, i.e. $MSE(\theta) = \frac{1}{m \cdot d} \cdot (\sum_{k=1}^{m} \sum_{l=1}^{d} (y_{k,l} - \hat{y}_{k,l}(\theta))^2)$ as the fitness function, where $y_{k,l}$ denotes m steady-state measurements for various mutants of the model for each observable l, i.e. specific subset of states, and $\hat{y}_{k,l}$ is the l-th observable's steady-state probability predicted by the mutant model k with parameters θ. In each iteration, all positions and speed of the particles are updated and verified according to the fitness function. The particle that has the minimum fitness function value is regarded as the optimal particle. Particle values are updated based on the current values and the current best optimal particle values so that each particle is moving towards the direction of the current best optimal particle.

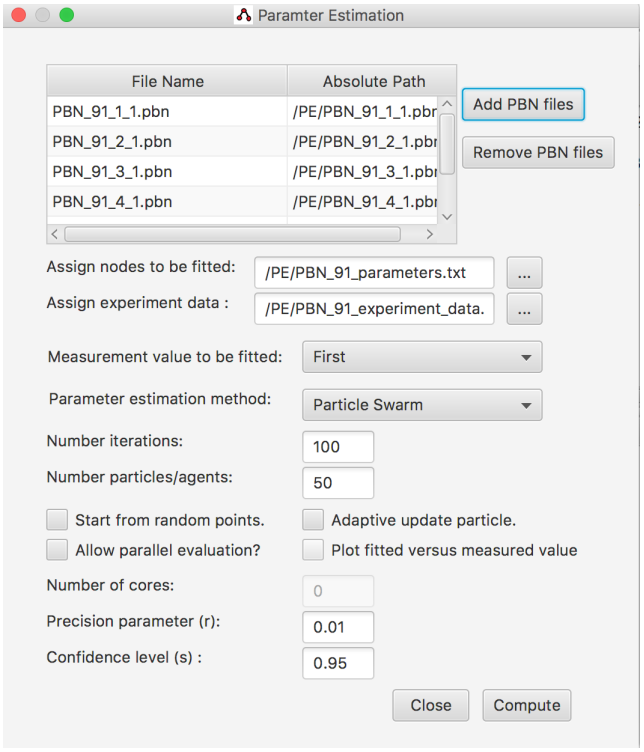


Figure 8.5: Interface of parameter estimation in ASSA-PBN.

DE is a population-based method introduced by Storn and Price in 1996 [Sto96, SP97]. It is developed to optimise real parameters by maintaining a population of candidate solutions that undergo iterations of mutation, recombination and selection. The mutations and recombinations expand the search space by creating new candidate solutions based on the weighted difference between two randomly selected population members added to a third population member. The selection process then keeps the solutions that result in better finesses. In conjunction with the selection, the mutation and recombination self-organise the sampling of the problem space, bounding the search space to known areas of interest.

Note that both PSO and DE are commonly known as meta-heuristics and are capable of exploring a large searching space. However, meta-heuristics such as PSO and DE do not guarantee that an optimal solution is ever found.

The parameter estimation interface is shown in Figure 8.5. The parameter estimation method drop-down list provides two available parameter estimation methods: PSO and DE. If the option "Start from random points" is selected, the parameter estimation will start from randomly generated parameter values. Otherwise, it will use the parameters specified in the first PBN model file (usually all the mutant PBN models should have the same parameter values for the same nodes). The option "Adaptive update particle" is specific to the PSO method. If its checked, PSO will use the adaptive update method to calculate the next position. We refer to [AM11] for more details on the adaptive update method. If the option "Allow parallel evaluation" is checked, the parameter estimation method will be run in parallel, which means that in each iteration x particles will be evaluated in parallel, where x is the number of cores. If the option "Plot fitted versus measured value" is checked, the parameter estimation result plot will be presented in a new window to the user at the end of the estimation. Once the parameter estimation

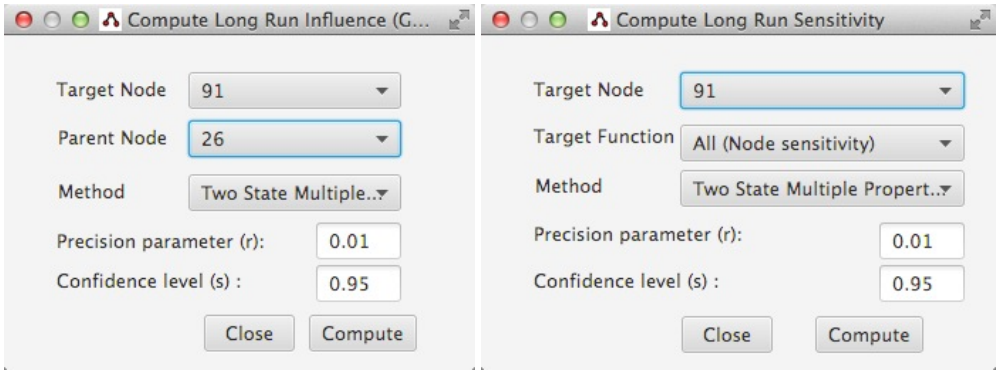
Figure 8.6: The fitness heat map presented after performing parameter estimation in ASSA-PBN.

procedure is finished, a fitness heat map is shown as illustrated in Figure 8.6. The heat map is a graphical representation of the fraction each squared error contributes to the fitness function. The columns represent PBN models under different experimental conditions or model mutants and the rows represent different subsets of states for which the steady-state probabilities are computed and compared with experimental measurements. The vertical colour bar on the right provides a mapping between a colour and corresponding range of percentage values.

8.4.3 Long-run Influence and Sensitivity

In a GRN, it is often important to distinguish which parent gene plays a major role in regulating a target gene. To explore the long-run characteristics of the GRNs, analyser of ASSA-PBN facilitates the computation of long-run influences and sensitivities The long-run influences include the long-run influence of a gene on a specified predictor function and the long-run influence of a gene on another gene. The long-run sensitivities include the average long-run sensitivity of a node, the average long-run sensitivity of a predictor function, the long-run sensitivity of a gene with respect to one-bit function perturbation, and the long-run sensitivity of a gene with respect to selection probability perturbation.

The computations of long-run influences and sensitivities are based on the computations of several steady-state probabilities. Note that ASSA-PBN does not store the generated trajectory for the sake of memory saving. Instead, ASSA-PBN verifies whether the next sampled state of the PBN belongs to the set of states of interest and stores this information only. Therefore, a new trajectory is required when computing the steady-state probability for a new set of states of interest. ASSA-PBN implements computation of steady-state probabilities of several sets of states in parallel with the two-state Markov chain approach [MPY16d], allowing the reuse of a generated trajectory. The crucial idea is that each time the next state of the PBN is generated, it is processed for all state sets of interest simultaneously. Different sets require trajectories of different lengths and the lengths are determined dynamically through an iterative process. Whenever the trajectory is long enough for obtaining the steady-state probability estimate for a certain set of states, the estimate is computed and the set will not be considered in subsequent iterations.



(a) Long-run influence of a gene on another gene interface

(b) Average long-run sensitivity of a node/predictor function interface

Figure 8.7: Interface of long-run analyses in ASSA-PBN.

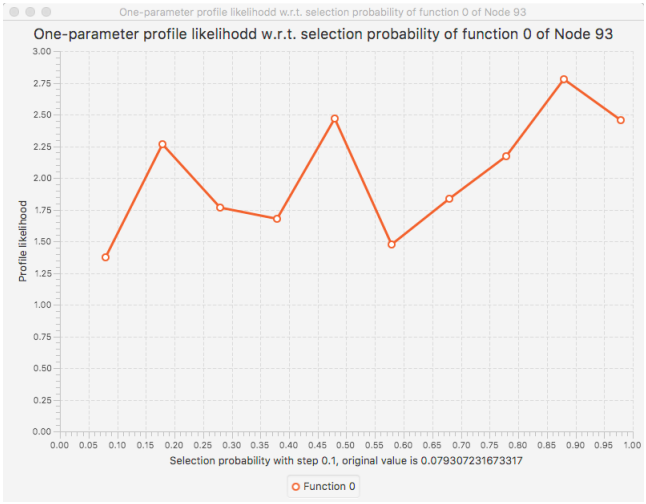


Figure 8.8: Plot of a profile likelihood computed in ASSA-PBN.

Figure 8.7a and Figure 8.7b show the interfaces of long-run influence of a gene on another gene and Average long-run sensitivity of a node/predictor function, respectively. The first two elements of the interfaces allow to specify the details of the analysis to be performed while the other three parameters, i.e. the method, the precision, and the confidence level, govern the computation of the required steady-state probabilities.

8.4.4 Towards Parameter Identifiability Analysis

In the current version ASSA-PBN implements the first part of the general approach of [RKM+09] to analyse arbitrary models for structural and practical identifiability. The approach is based on the concept of *profile likelihood* (PL). In this approach the fit of a model to experimental data is measured by an objective function which is the weighted sum of squared residuals

$$\chi^2(\theta) = \sum_{k=1}^{m} \sum_{l=1}^{d} \left(\frac{y_{k,l} - \hat{y}_{k,l}(\theta)}{\sigma_{k,l}} \right)^2 \tag{8.1}$$

where θ is a vector of model parameter values, $y_{k,l}$ denotes m steady-state measurements for individual mutants of the model for each observable l, $\hat{y}_{k,l}(\theta)$ is the l-th observable

as predicted by the mutant model k with parameter values θ, and $\sigma_{k,l}$ are the corresponding measurement errors. It is further assumed that the parameters are estimated to find $\hat{\theta} = \arg\min[\chi^2(\theta)]$. It can be shown that for normally distributed observational noise this corresponds to the maximum likelihood estimate (MLE) of θ as in this case $\chi^2(\theta) = \text{constant} - 2 \cdot \log(L(\theta))$, where $L(\theta)$ is the likelihood. In [RKM+09], the finite sample confidence intervals are considered, so called *likelihood-based confidence intervals*, defined by a confidence region $\{\theta \mid \chi^2(\theta) - \chi^2(\hat{\theta}) < \Delta_\alpha\}$ with $\Delta_\alpha = \chi^2(\alpha, \text{df})$ whose borders represent confidence intervals [ME95]. In the formula above Δ_α is the α quantile of the χ^2-distribution with df degrees of freedom and represents with $\text{df} = 1$ and $\text{df} = \dim(\theta)$ pointwise and simultaneous, respectively, confidence intervals with confidence level α.

A parameter θ_i is said to be *identifiable*, if the confidence interval $[\sigma_i^-, \sigma_i^+]$ of its estimate $\hat{\theta}_i$ is finite. Two types of parameter non-identifiability are commonly considered. *Structural non-identifiability* arises from a redundant parametrisation manifested as a functional relation between ambiguous parameters that represents a manifold with constant χ^2 value in parameter space. A structural non-identifiability is related to model structure independent of experimental data. For a single parameter it is indicated by flat profile likelihood. Structurally identifiable parameter may still be *practically non-identifiable*, second type of non-identifiability, due to the amount and quality of experimental data. By the definition of [RKM+09], a parameter is practically non-identifiable if the likelihood-based confidence region is infinitely extended, i.e. the increase in χ^2 stays below the threshold Δ_α, in the increasing and/or decreasing direction of θ_i although the likelihood has a unique minimum for this parameter.

The identification of potential structural or practical non-identifiability is based on the exploration of the parameter space in the direction of the least increase in χ^2. For this purpose the profile likelihood χ^2_{PL} is calculated for each parameter individually as $\chi^2_{\text{PL}}(\theta_i) = \min_{\theta_{j \neq i}}[\chi^2_{\text{PL}}]$ by re-optimisation of χ^2 with respect to all other parameters, for each value of θ_i.

Current version of ASSA-PBN facilitates the computation and visualisation of the profile likelihood for a specified parameter. However, since information on measurement errors is not considered in the current version, all $\sigma_{k,l}$ are set to 1 and the finite likelihood-based confidence intervals are not computed. The still missing elements are planned to be implemented in the next releases of ASSA-PBN. An example of a profile likelihood plot computed in ASSA-PBN is shown Figure 8.8.

8.5 Multiple Probabilities Problem

The functionalities provided by the analyser usually rely on the computation of several steady-state probabilities. For example, in the example figure for parameter estimation (Figure 8.5), the tool needs to compute 18 steady-state probabilities in each of the iterations. Those computations are performed via statistical methods and the precision for each of the computed probabilities is guaranteed by one of the previously mentioned statistical methods, e.g., the two-state Markov chain approach. However, when we are computing the 18 steady-state probabilities (properties) in each iteration, the chance that one of the 18 computed results does not meet the pre-defined precision requirement is increased compared to the case that only one probability is computed. Let r be the pre-

cision and α be the confidence level for approximating the steady-state probability of a set of states of a PBN with the two-state Markov chain approach. When computing m probabilities with the above precision requirement, the probability that at least one result does not meet the requirement is $1-(1-\alpha)^m$. This issue is known as the multiple comparisons problem in statistics [Jr.81, Ben10]. In our tool, we use the Bonferroni correction [Dun58, Dun61] to counteract this issue. Instead of computing the probability at a confidence level of α, the Bonferroni correction requires a confidence level of α/m. By providing a smaller confidence level α/m, it guarantees that the probability that at least one computed result does not meet the requirement is still smaller than α when m results are computed.

9

Conclusion and Future Work

9.1 Conclusion

This thesis studies the problem of dealing with the computational complexity of analysing long-run dynamics of large biological networks. This type of analysis is crucial in many contexts, e.g., in identifying cellular functional states. When the network is not very large, their dynamics can be analysed easily with several different methods. However, it often arises in the study of biological systems that the network to be analysed is so huge that the utilization of traditional methods is essentially prohibited. We take on this challenge in this thesis and propose several methods to handle the long-run dynamics analysis for large networks.

Fine-grained formalisms can easily result in a prohibitively complex model when used for modelling large networks; therefore, coarse-grained frameworks remain the only feasible solution for large networks. We focus on probabilistic Boolean networks (PBNs) as formal models of biological networks. PBNs focus on the wiring of a network while ignoring the reaction rate; they not only facilitate the modelling of large biological networks, but also can capture the important long-run dynamics of the modelled networks.

Within the PBN framework, we formulate two research problems with regard to the long-run dynamics analysis. The first is how to detect attractors in a large BN or in each of the constitute BNs of a large PBN effectively; and the second is how to compute steady-state probabilities of a large PBN efficiently.

With regard to the first research problem, we contribute by providing a decomposition method for efficiently identifying attractors in large BNs. This decomposition method works for both synchronous and asynchronous networks. The idea is to decompose a large BN into small sub-networks, detect attractors in the sub-networks, and recover attractors of the original network using attractors in the sub-networks. We prove that our decomposition method can correctly identify all the attractors of a BN. Our experimental results show that the proposed method is significantly faster than the state-of-the-art method. Detailed information on this method is presented in Chapters 3 and 4.

With regard to the second research problem, we contribute in two ways. First, we identify a potential problem in a well-known method called the two-state Markov chain approach and propose some heuristics to avoid it (in Chapter 5). Secondly, we propose several algorithms for improving the efficiency of computing steady-state probabilities of a large PBN. These algorithms include the multiple-core (CPU or GPU) based parallel steady-state computation as discussed in Chapter 6 and the structure-based parallel steady-state computation as discussed in Chapter 7.

Moreover, with the efficient steady-state computation algorithms, we are able to perform parameter estimation of large PBNs. Notably, we take special care of the precision

for estimating multiple steady-state probabilities when performing parameter estimation. We have implemented the above mentioned methods and algorithms in our tool ASSA-PBN. A detailed introduction to the tool, including a case-study demonstrating parameter estimation of a PBN, is provided in Chapter 8.

9.2 Future Work

There are a few interesting related research problems worth investigating, but they are not discussed in our thesis. We present here two of them.

9.2.1 Controllability of BNs

While identifying attractors of a network can be used directly for characterisation of a disease, it does not tell us how to cure the disease. To reach this goal, we need to switch from a diseased status to a healthy status. In mathematical terms, this corresponds to moving from one attractor to another in a PBN. This task is related to the problem of *controllability* of complex biological networks. The ability to control a biological system is the ultimate proof of our understanding of it [LSB11]. Controllability has attracted much interest due to practical applications such as stem cell reprogramming [GMNF14, PT10, You11] and the above mentioned search for therapeutic methods [AZH09, BGL11, WAJ+13].

The control of a BN can be though of driving the system from an initial state to any desired final state within limited time by application of suitable binary inputs. This process is usually referred as external control. Given two states from two different attractors, it is theoretically very easy to transition from one to the other by perturbing individual nodes. However, in practice, not all the nodes can be perturbed/controled due to real-world limitations [MSL+11]. It is more likely that the use of certain drugs will allow the activation or deactivation of only a few selected nodes. Therefore, it is essential to identify a minimum number of nodes sufficient to force the switch from one attractor to another in a BN. A number of computational complexity results have been achieved in the literature for reaching the goal of external control, e.g., [AHCN07, VFC+08]. Since the computational complexity in these cases is double exponential, the existing external control methods can only be applied to BNs with tens of nodes.

Our proposed work for identifying attractors in a decompositional way can be further extended to solve the controllability problem. In particular, it can solve the problem of *target control* [GLDB14], which aims for identifying a set of nodes which can drive the network from a source attractor to a target attractor. To reach the goal of target control, it is enough to control certain nodes such that the state of the system moves into one of the states in the *basin* of the target attractor. We are then guaranteed that the system will eventually evolve to the target attractor by its own dynamics. However, computing the basin of the target attractor will be intractable in the cases of large BNs. To overcome this, the idea of decomposition becomes essential. We decompose the original network into sub-networks based on the decomposition method as proposed in Chapters 3 and 4; the computation of the basin then becomes feasible since the number of nodes in each sub-network is usually significantly reduced. After identifying the nodes to be controlled in each sub-network, we can then recover the nodes to be controlled in the original

network. It is worth mentioning that this idea and the research work presented in this thesis have lead to a new research project: Scalable External Control of Probabilistic Boolean Networks, which is funded by the University of Luxembourg (reference UL-IRP-2015).

9.2.2 Decomposition of BNs

When we discuss the decomposition method in Chapters 3 and 4, we decompose a BN according to the SCCs in the BN structure. In this way, we are guaranteed that there are no feedback loops between different sub-networks. Hence, the attractors in each sub-network will not break the attractor structure in the original network. However, this also poses a limitation to this method, i.e., when there is a huge SCC in the network, the decomposition becomes meaningless since one of the sub-networks is still too large due to the huge SCC. The SCC-based decomposition does not necessary have to be the only way for decomposing a BN. Other ways for decomposing will also work as long as the decomposed sub-networks can preserve the attractor structure of the original BN (see Definition 3.3.6 in Chapter 3 for the concept of preservation of attractors).

A potential direction is to consider the conditions of multistationarity and periodicity in sub-networks [TK01]. The multistationarity and periodicity came from a conjecture of René Thomas in the 1980s [Tho81], which stated the following two rules:

1. the presence of a positive circuit in the interaction graph (i.e., a circuit containing an even number of inhibitions) is a necessary condition for the presence of several stable states in the dynamics;
2. the presence of a negative circuit in the interaction graph is a necessary condition for the presence of an attractive cycle in the dynamics.

This conjecture was later proved in differential frameworks [PMO95, Gou98, Sou03], in Boolean frameworks [RR06, RRT08], and in more general discrete frameworks [RC07]. The conjecture can be applied in two ways to solve the above mentioned issue. First, we may find a new way for decomposing a BN such that the preservation of attractors may be reached by satisfying the multistationarity and periodicity conditions between sub-networks, i.e, a single sub-network may drop the preservation, but combination of several sub-networks can recover the preservation. Secondly, we may reduce the complexity of a network by removing several positive feedbacks at the cost of losing the singleton attractors according to the first rule. However, this does not mean that we cannot identify the singleton attractors. Since identification of singleton attractors is much easier than that of cyclic ones, we can identify the singleton attractors of the original network separately in a very efficient way.

Bibliography

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, volume 736 of Lecture Notes in Computer Science, pages 209–229. Springer, 1993.

[AHCN07] T. Akutsu, M. Hayashida, W.-K. Ching, and M. K. Ng. Control of Boolean networks: Hardness results and algorithms for tree structured networks. *Journal of Theoretical Biology*, 244(4):670–679, 2007.

[Ake78] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 100(6):509–516, 1978.

[AM11] A. Alfi and H. Modares. System identification and control using adaptive particle swarm optimization. *Applied Mathematical Modelling*, 35(3):1210–1221, 2011.

[Ana17] Clarivate Analytics. Metacore. `https://clarivate.com/products/metacore/`, 2017.

[AO03] R. Albert and H. G. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster. *Journal of Theoretical Biology*, 223(1):1–18, 2003.

[AZH09] C. Auffray, C. Zhu, and L. Hood. Systems medicine: The future of medical genomics and healthcare. *Genome Medicine*, 1(1):2, 2009.

[BCB+16] J. Behaegel, J.-P. Comet, G. Bernot, E. Cornillon, and F. Delaunay. A hybrid model of cell cycle in mammals. *Journal of bioinformatics and computational biology*, 14(01):1640001, 2016.

[Ben10] Y. Benjamini. Simultaneous and selective inference: Current successes and future challenges. *Biometrical Journal*, 52(6):708–721, 2010.

[BGL11] A. Barabási, N. Gulbahce, and J. Loscalzo. Network medicine: A network-based approach to human disease. *Nature Reviews Genetics*, 12(1):56, 2011.

[BK96] F. Bause and P. Kritzinger. Stochastic petri nets. *Verlag Vieweg, Wiesbaden*, 26, 1996.

[BL16] E. Bartocci and P. Lió. Computational modeling, formal analysis, and tools for systems biology. *PLoS computational biology*, 12(1):e1004591, 2016.

[BMW06] N. Black, S. Moore, and E. W. Weisstein. Gauss-seidel method. From MathWorld-A Wolfram Web Resource, 2006. `http://mathworld.wolfram.com/Gauss-SeidelMethod.html`.

[BMW14] N. Black, S. Moore, and E. W. Weisstein. Jacobi method. From MathWorld–A Wolfram Web Resource, 2014. `http://mathworld.wolfram.com/JacobiMethod.html`.

[Bor05] S. Bornholdt. Less is more in modeling large genetic networks. *Science*, 310(5747):449–451, 2005.

[CGH06] M. Calder, S. Gilmore, and J.. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Lecture Notes in Computer Science*, 4230:1–23, 2006.

[CH07] I. R. Cohen and D. Harel. Explaining a complex living system: dynamics, multi-scaling and emergence. *Journal of the royal society interface*, 4(13):175–182, 2007.

[CH09] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009.

[CQZ12] J. Cao, X. Qi, and H. Zhao. Modeling gene regulation networks using ordinary differential equations. *Next Generation Microarray Bioinformatics: Methods and Protocols*, pages 185–197, 2012.

[DFTdJV06] S. Drulhe, G. Ferrari-Trecate, H. de Jong, and A. Viari. Reconstruction of switching thresholds in piecewise-affine models of genetic regulatory networks. In *International Workshop on Hybrid Systems: Computation and Control*, pages 184–199. Springer, 2006.

[DHRK07] A. S. Dhillon, S. Hagan, O. Rath, and W. Kolch. MAP kinase signalling pathways in cancer. *Oncogene*, 26:3279–3290, 2007.

[dJR06] H. de Jong and D. Ropers. Strategies for dealing with incomplete information in the modeling of molecular interaction networks. *Briefings in bioinformatics*, 7(4):354–363, 2006.

[DMS+04] P. Dhar, T. C. Meng, S. Somani, L. Ye, A. Sairam, M. Chitre, Z. Hao, and K. Sakharkar. Cellwarea multi-algorithmic software for computational systems biology. *Bioinformatics*, 20(8):1319–1321, 2004.

[DPR08] L. Dematté, C. Priami, and A. Romanel. The Beta Workbench: a computational tool to study the dynamics of biological systems. *Briefings in Bioinformatics*, 9(5):437–449, 2008.

[DT11] E. Dubrova and M. Teslenko. A SAT-based algorithm for finding attractors in synchronous Boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1393–1399, 2011.

[DTM05] E. Dubrova, M. Teslenko, and A. Martinelli. Kauffman networks: Analysis and applications. In *Proc. 2005 IEEE/ACM International Conference on Computer-Aided Design*, pages 479–484. IEEE CS, 2005.

[Dun58] O. J. Dunn. Estimation of the means of dependent variables. *The Annals of Mathematical Statistics*, pages 1095–1111, 1958.

[Dun61] O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

[EP09] D. El Rabih and N. Pekergin. Statistical model checking using perfect simulation. In *Proc. 7th Symposium on Automated Technology for Verification and Analysis*, volume 5799 of *LNCS*, pages 120–134, 2009.

[FH07] J. Fisher and T. A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, 2007.

[FH10] J. Fisher and D. Harel. *On Statecharts for Biology*. Jones & Bartlett Publishers, 2010.

[FHL+04] A. Finkelstein, J. Hetherington, L. Li, O. Margoninski, P. Saffrey, R. Seymour, and A. Warner. Computational challenges of systems biology. *Computer*, 37(5):26–33, 2004.

[FMKT03] A. Funahashi, M. Morohashi, H. Kitano, and N. Tanimura. Celldesigner: a process diagram editor for gene-regulatory and biochemical networks. *Biosilico*, 1(5):159–162, 2003.

[Gat10] D. Gatherer. So what do we really mean when we say that systems biology is holistic? *BMC Systems Biology*, 4(1):22, 2010.

[GCBP+13] L. Grieco, L. Calzone, I. Bernard-Pierrot, F. Radvanyi, B. Kahn-Perles, and D. Thieffry. Integrative modelling of the influence of MAPK network on cancer cell fate decision. *PLOS Computational Biology*, 9(10):e1003286, 2013.

[GDCX+08] A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, and G. De Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925, 2008.

[GLDB14] J. Gao, Y.-Y. Liu, R. M. D'Souza, and A.-L. Barabási. Target control of complex networks. *Nature Communications*, 5:5415, 2014.

[GMNF14] E. Garreta, E. Melo, D. Navajas, and R. Farr. Low oxygen tension enhances the generation of lung progenitor cells from mouse embryonic and induced pluripotent stem cells. *Physiological Reports*, 2(2), 2014.

[Gou98] J. L. Gouzé. Positive and negative circuits in dynamical systems. *Journal of Biological Systems*, 6(01):11–15, 1998.

[GPPQ09] M. L. Guerriero, D. Prandi, C. Priami, and P. Quaglia. Process calculi abstractions for biology. *Algorithmic Bioprocesses*, pages 463–486, 2009.

[GR92] A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992.

[GTT03] R. Ghosh, A. Tiwari, and C. Tomlin. Automated symbolic reachability analysis; with application to delta-notch signaling automata. *Hybrid Systems: Computation and Control*, pages 233–248, 2003.

[GXMD07] A. Garg, L. Xenarios, L. Mendoza, and G. DeMicheli. An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments. In *Proc. 11th Annual Conference on Research in Computational Molecular Biology*, volume 4453 of *LNCS*, pages 62–76. Springer, 2007.

[GYW+14] W. Guo, G. Yang, W. Wu, L. He, and M. Sun. A parallel attractor finding algorithm based on Boolean satisfiability for genetic regulatory networks. *PLOS ONE*, 9(4):e94258, 2014.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[HK09] A. P. Heath and L. E. Kavraki. Computational challenges in systems biology. *Computer Science Review*, 3(1):1–17, 2009.

[HKN+08] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.

[Hop08] A. L. Hopkins. Network pharmacology: The next paradigm in drug discovery. *Nature Chemical Biology*, 4(11):682–690, 2008.

[HSG+06] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI: a COmplex PAthway SImulator. *Bioinformatics*, 22(24):30673074, 2006.

[Hua99] Sui Huang. Gene expression profiling, genetic networks, and cellular states: An integrating concept for tumorigenesis and drug discovery. *Journal of Molecular Medicine*, 77(6):469–480, 1999.

[Hua01] Sui Huang. Genomics, complexity and drug discovery: Insights from Boolean network models of cellular regulation. *Pharmacogenomics*, 2(3):203–222, 2001.

[IGH01] T. Ideker, T. Galitski, and L. Hood. A new approach to decoding life: Systems biology. *Annual Review of Genomics and Human Genetics*, 2(1):343–372, 2001.

[IM04] N. T. Ingolia and A. W. Murray. The ups and downs of modeling the cell cycle. *Current Biology*, 14(18):R771–R777, 2004.

[Iro06] D. J. Irons. Improving the efficiency of attractor cycle identification in Boolean networks. *Physica D: Nonlinear Phenomena*, 217(1):7–21, 2006.

[Jen87] K. Jensen. Coloured petri nets. In *Petri nets: central models and their properties*, pages 248–299. Springer, 1987.

[Jr.81] R. G. Miller Jr. *Simultaneous Statistical Inference*. Springer, New York, NY, 1981.

[Kau69a] S. A. Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224:177–178, 1969.

[Kau69b] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[Kau93] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[KBS15] H. Klarner, A. Bockmayr, and H. Siebert. Computing maximal and minimal trap spaces of Boolean networks. *Natural Computing*, 14(4):535–544, 2015.

[KBSK09] J. Kielbassa, R. Bortfeldt, S. Schuster, and I. Koch. Modeling of the U1 snRNP assembly pathway in alternative splicing in human cells using Petri nets. *Computational biology and chemistry*, 33(1):46–61, 2009.

[KCH01] N. Kam, I. R. Cohen, and D. Harel. The immune system as a reactive system: Modeling T-cell activation with statecharts. In *Proc. Human-Centric Computing Languages and Environments*, pages 15–22. IEEE, 2001.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[KIM03] S. Y. Kim, S. Imoto, and S. Miyano. Inferring gene networks from time series microarray data using dynamic Bayesian networks. *Briefings in bioinformatics*, 4(3):228–235, 2003.

[Kit02] H. Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[KJH04] I. Koch, B. H. Junker, and M. Heiner. Application of Petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber. *Bioinformatics*, 21(7):1219–1226, 2004.

[KN08] M. Krishna and H. Narang. The complexity of mitogen-activated protein kinases (MAPKs) made simple. *Cellular and Molecular Life Sciences*, 65(22):3525–3544, 2008.

[KPQ11] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Proc. 41st IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 359–370. IEEE, 2011.

[Lee59] C.-Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

[LHSYH06] H. Lähdesmäki, S. Hautaniemi, I. Shmulevich, and O. Yli-Harja. Relationships between probabilistic Boolean networks and dynamic Bayesian networks as models of gene regulatory networks. *Signal Processing*, 86(4):814–834, 2006.

[LLL+04] F. Li, T. Long, Y. Lu, Q. Ouyang, and C. Tang. The yeast cell-cycle network is robustly designed. *Proceedings of the National Academy of Sciences of the United States of America*, 101(14):4781–4786, 2004.

[LMP+14] R. Lintott, S. McMahon, K. Prise, C. Addie-Lagorio, and C. Shankland. Using process algebra to model radiation induced bystander effects. In *Proc. 12th International Conference on Computational Methods in Systems Biology*, pages 196–210. Springer, 2014.

[LPW09] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

[LQR15] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: An open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 2015.

[LS09] H. Lähdesmäki and I. Shmulevich. BN/PBN Toolbox. `http://code.google.com/p/pbn-matlab-toolbox`, 2009. Accessed 2017 March 24.

[LSB11] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabasi. Controllability of complex networks. *Nature*, 473(7346):167–173, 05 2011.

[LSG+06] C. Li, S. Suzuki, Q.W. Ge, M. Nakata, H. Matsuno, and S. Miyano. Structural modeling and analysis of signaling pathways based on Petri nets. *Journal of bioinformatics and computational biology*, 4(05):1119–1140, 2006.

[Luc02] R. Luc. Dynamics of Boolean networks controlled by biologically meaningful functions. *Journal of Theoretical Biology*, 218(3):331–341, 2002.

[ME95] W. Q. Meeker and L. A. Escobar. Teaching about approximate confidence regions based on maximum likelihood estimation. *The American Statistician*, 49(1):48–53, 1995.

[MMB03] C. G Moles, P. Mendes, and J. R. Banga. Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003.

[MML09] B. D. MacArthur, A. Ma'ayan, and I. R. Lemischka. Systems biology of stem cell fate and cellular reprogramming. *Nature Reviews Molecular Cell Biology*, 10(10):672–681, 2009.

[MPQY] A. Mizera, J. Pang, H. Qu, and Q. Yuan. Benchmark Boolean networks. `http://satoss.uni.lu/software/ASSA-PBN/benchmark/attractor_syn.xlsx`.

[MPQY18] A. Mizera, J. Pang, H. Qu, and Q. Yuan. Taming asynchrony for attractor detection in large Boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (Special issue of 16th Asia Pacific Bioinformatics Conference - APBC'18)*, 2018.

[MPY15] A. Mizera, J. Pang, and Q. Yuan. ASSA-PBN: An approximate steady-state analyser of probabilistic Boolean networks. In *Proc. 13th International Symposium on Automated Technology for Verification and Analysis*, volume 9364 of *LNCS*, pages 214–220. Springer, 2015.

[MPY16a] A. Mizera, J. Pang, and Q. Yuan. ASSA-PBN 2.0: A software tool for probabilistic Boolean networks. In *Proc. 14th International Conference on Computational Methods in Systems Biology*, volume 9859 of *LNCS*, pages 309–315. Springer, 2016.

[MPY16b] A. Mizera, J. Pang, and Q. Yuan. Fast simulation of probabilistic Boolean networks. In *Proc. 14th International Conference on Computational Methods in Systems Biology*, volume 9859 of *LNCS*, pages 216–231. Springer, 2016.

[MPY16c] A. Mizera, J. Pang, and Q. Yuan. GPU-accelerated steady-state computation of large probabilistic Boolean networks. In *Proc. 2nd International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, volume 9984 of *LNCS*, pages 50–66. Springer, 2016.

[MPY16d] A. Mizera, J. Pang, and Q. Yuan. Parallel approximate steady-state analysis of large probabilistic Boolean networks. In *Proc. 31st ACM Symposium on Applied Computing*, pages 1–8, 2016.

[MPY17] A. Mizera, J. Pang, and Q. Yuan. Reviving the two-state markov chain approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017.

[MQPY17] A. Mizera, H. Qu, J. Pang, and Q. Yuan. A new decomposition method for attractor detection in large synchronous Boolean networks. In *Proc. 3rd International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, 2017.

[MSL+11] F.-J. Müller, A. Schuppert, Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási. Few inputs can reprogram biological networks/liu et al. reply. *Nature*, 478(7369):E4, 2011.

[Nor98] J. R. Norris. *Markov Chains*. Cambridge University Press, 1998.

[NRTC11] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*, 412(21):2207–2218, 2011.

[OALH06] J. P. Overington, B. Al-Lazikani, and A. L. Hopkins. How many drug targets are there? *Nature Reviews Drug Discovery*, 5(12):993–996, 2006.

[OGP02] I. M. Ong, J. D. Glasner, and D. Page. Modelling regulatory pathways in E. coli from time series expression profiles. *Bioinformatics*, 18(suppl_1):S241–S248, 2002.

[Pea14] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[PMO95] E. Plahte, T. Mestl, and S. W. Omholt. Feedback loops, stability and multistationarity in dynamical systems. *Journal of Biological Systems*, 3(02):409–413, 1995.

[PR08] C. A. Petri and W. Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008. revision #91646.

[PRSS01] C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information processing letters*, 80(1):25–31, 2001.

[PT10] M. F. Pera and P. P. Tam. Extrinsic regulation of pluripotent stem cells. *Nature*, 465(7299):713, 2010.

[PW96] J. G. Propp and D. B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures & Algorithms*, 9(1):223–252, 1996.

[QD09] X. Qian and E. R. Dougherty. On the long-run sensitivity of probabilistic Boolean networks. *Journal of Theoretical Biology*, 257(4):560–577, 2009.

[RC07] A. Richard and J.-P. Comet. Necessary conditions for multistationarity in discrete dynamical systems. *Discrete Applied Mathematics*, 155:2403–2413, 2007.

[RKM+09] A. Raue, C. Kreutz, T. Maiwald, J. Bachmann, M. Schilling, U. Klingmller, and J. Timmer. Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. *Bioinformatics*, 25(15):1923–1929, 2009.

[RL92] A. E. Raftery and S. Lewis. How many iterations in the Gibbs sampler? *Bayesian Statistics*, 4:763–773, 1992.

[Roh13] C. Rohr. Simulative model checking of steady state and time-unbounded temporal operators. *Transactions on Petri Nets and Other Models of Concurrency*, 8:142–158, 2013.

[RR06] É. Remy and P. Ruet. On differentiation and homeostatic behaviours of Boolean dynamical systems. *Transactions Computational Systems Biology VIII*, 4230:153–162, 2006.

[RRT08] É. Remy, P. Ruet, and D. Thieffry. Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. *Advances in Applied Mathematics*, 41(3):335–350, 2008.

[SAA10] A. Saadatpour, I. Albert, and R. Albert. Attractor analysis of asynchronous Boolean models of signal transduction networks. *Journal of Theoretical Biology*, 266:641–656, 2010.

[SBSW06] L. J. Steggles, R. Banks, O. Shaw, and A. Wipat. Qualitatively modelling and analysing genetic regulatory networks: a petri net approach. *Bioinformatics*, 23(3):336–343, 2006.

[SD10] I. Shmulevich and E. R. Dougherty. *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*. SIAM Press, 2010.

[SDKZ02] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.

[SDZ02a] I. Shmulevich, E. R. Dougherty, and W. Zhang. Control of stationary behavior in probabilistic Boolean networks by means of structural intervention. *Journal of Biological Systems*, 10(04):431–445, 2002.

[SDZ02b] I. Shmulevich, E. R. Dougherty, and W. Zhang. From Boolean to probabilistic Boolean networks as models of genetic regulatory networks. *Proceedings of the IEEE*, 90(11):1778–1792, 2002.

[SG01] R. Somogyi and L. D. Greller. The dynamics of molecular networks: Applications to therapeutic discovery. *Drug Discovery Today*, 6(24):1267–1277, 2001.

[SGH+03] I. Shmulevich, I. Gluhovsky, R. F. Hashimoto, E. R. Dougherty, and W. Zhang. Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks. *Comparative and Functional Genomics*, 4(6):601–608, 2003.

[SHF07] M. A. Schaub, T. A. Henzinger, and J. Fisher. Qualitative networks: a symbolic approach to analyze biological signaling networks. *BMC systems biology*, 1(1):4, 2007.

[Shi09] Y. Shinya. Elite and stochastic models for induced pluripotent stem cell generation. *Nature*, 460(7251):49, 2009.

[SHK06] A. Sackmann, M. Heiner, and I. Koch. Application of Petri net based analysis techniques to signal transduction pathways. *BMC bioinformatics*, 7(1):482, 2006.

[SN10] Y.-J. Shin and M. Nourani. Statecharts for gene network modeling. *PLoS One*, 5(2):e9376, 2010.

[SNC+17] E. Scott, J. Nicol, J. Coulter, A. Hoyle, and C. Shankland. Process algebra with layers: Multi-scale integration modelling applied to cancer therapy (forthcoming). In *Proc. 13th International Conference on Computational Intelligence methods for Bioinformatics and Biostatistics*. Springer, 2017.

[Som15] F. Somenzi. CUDD: CU decision diagram package - release 2.5.1. `http://vlsi.colorado.edu/~fabio/CUDD/`, 2015.

[Sou03] C. Soulé. Graphic requirements for multistationarity. *ComPlexUs*, 1(3):123–133, 2003.

[SP97] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[SSV+09] R. Schlatter, K. Schmich, I. A. Vizcarra, P. Scheurich, T. Sauter, C. Borner, M. Ederer, I. Merfort, and O. Sawodny. ON/OFF and beyond - a boolean model of apoptosis. *PLOS Computational Biology*, 5(12):e1000595, 2009.

[Sto96] R. Storn. On the usage of differential evolution for function optimization. In *Proc. Biennial Conference of the North American Fuzzy Information Processing Society*, pages 519–523, 1996.

[SVA05] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *Proc. 17th Conference on Computer Aided Verification*, volume 3576 of *LNCS*, pages 266–280. Springer, 2005.

[Tho81] R. Thomas. On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. *Springer series in Synergetics*, 9:180–193, 1981.

[TK01] R. Thomas and M. Kaufman. Multistationarity, the basis of cell differentiation and memory. ii. logical analysis of regulatory networks in terms of feedback circuits. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 11(1):180–195, 2001.

[TMD+11] K. Tun, M. Menghini, L. D'Andrea, P. Dhar, H. Tanaka, and A. Giuliani. Why so few drug targets: A mathematical explanation? *Current Computer-aided Drug Design*, 7(3):206–213, 2011.

[TMP+13] P. Trairatphisan, A. Mizera, J. Pang, A.-A. Tantar, J. Schneider, and T. Sauter. Recent development and biomedical applications of probabilistic Boolean networks. *Cell Communication and Signaling*, 11:46, 2013.

[TMP+14] 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.

[TWLS08] 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. 2008 Winter Simulation Conference*, pages 387–395, 2008.

[VCCW12] N. X. Vinh, M. Chetty, R. Coppel, and P. P. Wangikar. Gene regulatory network modeling via global optimization of high-order Dynamic bayesian network. *BMC bioinformatics*, 13(1):131, 2012.

[VFC+08] G. Vahedi, B. Faryabi, J.-F. Chamberland, A. Datta, and E. R. Dougherty. Intervention in gene regulatory networks via a stationary mean-first-passage-time control policy. *IEEE Transactions on Biomedical Engineering*, 55(10):2319–2331, 2008.

[VM04] 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.

[Wad57] C. H. Waddington. *The strategy of the genes*. George Allen & Unwin, London, 1957.

[WAJ+13] O. Wolkenhauer, C. Auffray, R. Jaster, G. Steinhoff, and O. Dammann. The road from systems biology to systems medicine. *Pediatric Research*, 73(2):502–7, 2013.

[Wal77] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 3(3):253–256, 1977.

[WMG08] S. Watterson, S. Marshall, and P. Ghazal. Logic models of pathway biology. *Drug discovery today*, 13(9):447–456, 2008.

[WSA12] R.-S. Wang, A. Saadatpour, and R. Albert. Boolean modeling in systems biology: An overview of methodology and applications. *Physical Biology*, 9(5):055001, 2012.

[You11] R. A. Young. Control of the embryonic stem cell state. *Cell*, 144(6):940–954, 2011.

[YQPM16] Q. Yuan, H. Qu, J. Pang, and A. Mizera. Improving BDD-based attractor detection for synchronous Boolean networks. *Science China Information Sciences*, 59(8):080101, 2016.

[YS02] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. 14th Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.

[ZC04] M. Zou and S. D. Conzen. A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, 21(1):71–79, 2004.

[ZKF13] Y. Zhao, J. Kim, and M. Filippone. Aggregation algorithm towards large-scale Boolean network analysis. *IEEE Transactions on Automatic Control*, 58(8):1976–1985, 2013.

[ZOS03] I. Zevedei-Oancea and S. Schuster. Topological analysis of metabolic networks based on Petri net theory. *In silico biology*, 3(3):323–345, 2003.

[ZYL+13] D. Zheng, G. Yang, X. Li, Z. Wang, F. Liu, and L. He. An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. *PLOS ONE*, 8(4):e60593, 2013.

Curriculum Vitae

2014 – 2018	Ph.D. student, University of Luxembourg.
2009 – 2012	Master of Computer Science, Shandong University, China.
2010 – 2011	Master of Computer Science, University of Luxembourg.
2005 – 2009	Bachelor of Software Engineering, Shandong University, China

Born on December 28, 1986, Rizhao, China.