Faculty of Science and Engineering

Masters Thesis

A Network Model towards Strong Controllability of Biological Networks

Author:
Patric Gustafsson,
38847

Supervisor:
Prof. Ion Petre

2019

Contents

Abstract

In systems biology, we study structures and networks with the goal of achieving an understanding of the system as a whole. The networks we study are often biological networks, say cancer networks. When we understand how these systems evolve over time and what their characteristics are, we can think about controlling the network to a specific state. Identification of new drugs and treatments is one application of network controllability where, if we can show how to control the system into a healthy state, we can then develop a drug which mimics that behavior.

In this thesis, we extend this idea of network controllability. We introduce a model with the goal of achieving something we call *strong controllability*. The idea is that unanimous influences are conserved over long paths which results in stronger controllability, hence the name. With unanimous influences we mean that there is no disagreement between the influences, such as receiving both an activation signal and an inhibition signal. The case of contradictory influences is handled in a special way in order to ensure that only the unanimous influences win.

The model we introduce is called *strong regulatory graphs*. Our model is defined on the already well-established *gene-regulatory* networks. We show that the model is well defined and produces non-trivial state spaces. We apply the model on a small cancer network and compare the state space to a majority-voting based Boolean network model.

We also introduce a concept called *target attractors* that serve as this thesis's major contribution towards *strong controllability*. Since much theory has been developed for Boolean networks, we also introduce a transformation from our model into a Boolean network that preserves the transitions and attractors. Finally, we present some open research problems related to strong controllability that can be good venues for further research.

In addition to the theoretical part, we also introduce a software tool that we have developed, which can help with the analysis of the models. The tool can show, simulate, analyze and control networks. The software is multi-platform and can be run on any modern computer without the need to install additional software.

Chapter 1

Motivation

Biology and computer science might seem like an unlikely pair. Biology is an experimental science where things are discovered and verified through experiments, whereas computer science, much like mathematics, deals with absolute truths. Perhaps surprisingly, biology and computer science are today working together more than ever, as is evident from the relatively new discipline called *bioinformatics*. The explosion of biological data has developed a need for new tools and algorithms to handle that data. Projects such as the Human Genome Project would certainly not have been completed without the development of new sequencing algorithms. However, it is not only biology that has benefited from computer science, the other way around is also true. The field of machine and deep-learning was originally inspired by how the neurons in the human brain function.

A field where both computer scientists and biologists cooperate on a large scale is *systems biology*. Here we try to understand how biological systems work by studying them in a systems approach. It is very much a holistic approach to biology which has traditionally had a more reductionist approach where smaller parts were studied in-depth in the hope that understanding the smaller parts would help in understanding the bigger picture as well. In systems biology, the belief is that the knowledge of the system as a whole is needed before the individual properties can be understood. Even if the approach is more abstract the networks and structures that are studied come from reality, for example from experiments done by biologists. The networks often come in the form of *gene-regulatory* networks. In these networks, every gene either activates or inhibits some other genes. With the help of these networks it is possible to, for example, make predictions about possible drug treatments for illnesses, or study the development of cancer in cells [14, 7]. Why is computer science needed in this field? The reason is that the structures that are studied in systems biology have given rise to new computational problems that are computationally difficult and say something interesting about nature and reality itself.

There exist many frameworks of study within systems biology, ranging from discrete to continuous to a mix of both [26]. Most continuous models, such as differential equation models, rely on quantitative and qualitative data about the networks being modeled. This data is often not available [6] and even if it is, the continuous model presents additional challenges, such as the need for accurate parameters [20]. When modeling discretely, for example with Boolean networks,

in which we only work with the values of 1 and 0 indicating if a gene is active or not, these biochemical values need not be known since they are not taken into account into the model. What matters more for logical modeling is that we have access to the topology of the network, that is, the interactions between the nodes in the network, such as who inhibits who and similar for activation. Such networks are available from the KEGG PATHWAY [21] database and other similar databases.

At a first glance, it might seem unreasonable, careless even, to try to model any kind of reality with such a coarse model as Boolean network or any other kind of logical modeling. Is reality in itself not a continuous process, especially when we are talking about biology? As a matter of fact, logical modeling has been used successfully for many real biological applications. One such example is the identification of drug targets and treatments [6]. They are not without faults, however. A common approach for Boolean networks is to use a modeling technique called majority voting. In majority voting, the gene's influence which has the most representative, a majority, determines the state of the gene. This is not a biologically faithful model. The strengths of the interactions are not taken into account. For example, it is often the case that inhibition is seen as the stronger influence even if it has fewer representatives [41]. In many cases the strengths of the interactions are simply unknown or difficult to measure [39].

In this thesis we take a new approach. We will consider the unanimous influences on a node as a starting point for our model. With unanimous we mean influences that are in agreement. Consider for example a node that has only inhibition influences. This node, we can be sure, will always be inactive as soon as it receives a signal. Now consider instead a node that has both activation and inhibition influences. What should we do if both of them are active at the same time, if we do not know the strengths of the interactions? In our case, we will consider this to be an *ambiguous* situation. If a node is not in an ambiguous state then we can in some sense say that the state is *strong*, meaning that the node really should be in that state, otherwise it would have been ambiguous. We will develop this much more in-depth in the coming chapters.

We will also focus on a concept called *network controllability*. Network controllability means that we take control over some nodes and try to guide the system from one state to another. This could, for example, be a system that has entered an unhealthy state and we would like to guide it to a healthy state. We will consider network controllability in the form of *strong controllability* where we want to guide our systems into specific stable states that have certain properties that we will describe later. How we can find these stable states will be the main focus of the thesis.

In Chapter 2, we will start the thesis with a review of some mathematical concepts that are needed for the thesis. In Chapter 3, we will introduce our own model that is based on the intuition given above and give a few examples of how the model behaves in some examples. In Chapter 4, we present an algorithm for finding interesting stable state based on unanimity between the nodes. Most of the time will be spent in this chapter, since we need to reason extensively and introduce new ideas. We also introduce a transformation from our model into a Boolean network. We also introduce two open problems related to *strong controllability*. Finally, we introduce a tool that we have developed for the purpose of analyzing and working with our models.

Chapter 2

Mathematical Preliminaries

This chapter is a short introduction to the necessary mathematical structures and concepts that are used in this thesis. We start with a short overview of some basic concepts from algorithm analysis and complexity theory. Then we introduce the notion of Boolean networks, which is a widely used framework for modeling biological networks. After that we discuss some general issues about *network controllability* and how that more specifically applies to discrete logic-based models such as Boolean networks. We also mention the complexity of some well-known problems related to Boolean networks and controllability.

2.1 Algorithm analysis

There is a need in computer science to categorize problems into classes so that it is possible to easily know if a problem is, in some sense, easy or difficult. In the field of algorithm analysis, we look closer at the performance of algorithms, often stated with pseudocode. We assume there is some abstract model of computation that can run this pseudocode and all operations take a set amount of time. Thus, we can ignore differences between different machines and architectures and focus only on what matters, which is the run-time and memory usage of our algorithms.

A useful tool for analyzing algorithms is the so-called "Big-Oh" notation, popularized among computer scientists in the 1970's by Donald Knuth [24]. The idea is that we can analyze algorithms in a machine-independent way. Constants and other factors are ignored even though they could have some difference in a real-life implementation. We only care about the operations that take the longest amount of time. An algorithm's running time is often defined as a function of the size of the input, which will be denoted by n in this section. We can see that an algorithm that has a running time of n^2 will be faster than one with 2^n since already for small n the latter expression grows much faster. We formalize this idea as follows:

Definition 1. (Upper bound) $\mathcal{O}(f(n))$ is the set of all $g(n)$ such that there exist constants $c \geq 0$ and $n_0 \geq 0$ with the property that $|g(n)| \leq cf(n)$, for all $n \geq n_0$.

If we say that $g(n) = \mathcal{O}(f(n))$, then we can intuitively think about it as f being a strict upper bound on g, meaning that when $n \geq n_0$, f will always grow

faster than g. There is a similar notion for a function that works as a lower bound:

Definition 2. (Lower bound) $\Omega(f(n))$ is the set of all $g(n)$ such that there exist constants $c \geq 0$ and $n_0 \geq 0$ with the property that $|g(n)| \geq cf(n)$, for all $n \geq n_0$.

Similarly to the upper bound we can now think of $g(n) = \Omega(f(n))$ as saying that $g(n)$ always grows faster than $f(n)$ when $n \geq n_0$. Combining the definitions we have a definition for a strict bound:

Definition 3. (Exact bound) $\theta(f(n))$ is the set of all functions $g(n)$ such that $g(n) = \Omega(f(n))$ and $g(n) = \mathcal{O}(f(n))$.

Here we see that the statement $g(n) = \theta(f(n))$ implies that $g(n)$ grows at exactly the same rate as $f(n)$.

When doing algorithm analysis we are often interested in the worst-case running time. We often try to find an upper bound for some algorithm and state that in the worst case, the algorithm will always have a running time below some upper bound. It is often easy to find a simple upper bound, yet finding an exact one can be more challenging. In this thesis, we will deal mostly with bounds that are quadratic or exponential. Quadratic means that the running times grow by the square of the input, such as n^2 while exponential means that there is some exponential relationship with the input, such as 2^n.

2.2 Computational complexity

The goal of computational complexity theory is to categorize different problems into classes, where each class represents a different degree of difficulty. The field is relatively new compared to other fields of science. It began to manifest itself among researches in the 1970's due to Karp's paper [22] where he showed that 21 well-known problems related to graph-theory were in fact $\mathcal{NP}$-complete. A recent textbook about computational complexity which this section is based on is [3]. Although there exists many complexity classes, we will only focus on the two perhaps most famous ones, $\mathcal{P}$ and $\mathcal{NP}$. Informally, the difference between these two classes can be explained by an example.

Consider the problem called the *Traveling salesman*-problem. The goal of the salesman is to visit every city on a given map only once and, in addition, he wants to return to his original starting city. Now, given a list of visited cities $(c_1, c_2, \cdots, c_n)$ it is easy to verify that he indeed visited every city once and that $c_1 = c_n$. However, without this list how should we propose a route for him to take? There exists an enormously large number of different paths. The total number of routes are $n!$ in the case of a fully connected graph. The class of $\mathcal{NP}$ can be described as problems where given a solution, it is easy to verify that solution, but finding a solution is much more difficult. Formally, complexity classes are often defined in terms of a Turing Machine (TM). The reason is that the TM is the simplest form of computation that is still equivalent to most other notions of a computational device.

We first define the class of $\mathcal{P}$ in terms of decision problems. A decision problem is a problem that has a *yes* or *no* answer. The problem of determining whether a number is prime or not is a well-known decision problem.

Definition 4. (Decision Problem) A decision problem is a set $L_f = \{x : f(x) = 1\}$, where f is a Boolean function, giving 0 or 1 as an answer. The problem is: given an instance x of a problem, decide if $f(x) = 1$ or equivalently $x \in L_f$.

We look at Boolean functions more closely in the next section in the context of Boolean networks. Here it is only relevant to know that they give either 0 or 1 as an output.

Definition 5. (Class $\mathcal{P}$) The class of $\mathcal{P}$ is the set of all decision problems which can be decided in polynomial time, that is $\mathcal{O}(n^k)$, for some $k \geq 0$.

Problems that belong to $\mathcal{P}$ are usually thought of as problems that can be solved efficiently. We now move on to the class of $\mathcal{NP}$. In modern literature, this class is often defined in terms of a verifier, which we also use here. The intuition behind the verifier was explained in the traveling salesman example above. That is, given a solution to a problem, we can easily check that the solution is correct.

Definition 6. (Class $\mathcal{NP}$) The class of $\mathcal{NP}$ is the set of all decision problems which have a verifier that runs in polynomial time. That is, given a solution y to a problem L_f, the verifier outputs:

1. "Yes" if y is a solution, that is, $y \in L_f$;
2. "No" if y is not a solution, that is, $y \notin L_f$.

Note that the definition does not include any information about actually obtaining the solution. From the definition we see that $\mathcal{P} \subseteq \mathcal{NP}$, since having an algorithm that produces a solution in polynomial time is essentially our verifier for problems in $\mathcal{P}$. Problems that are in $\mathcal{NP}$ are often thought of as problems where finding a solution that runs in polynomial time is impossible. Within $\mathcal{NP}$ there is a class of problems known as $\mathcal{NP}$-complete problems. They are defined as follows:

Definition 7. ($\mathcal{NP}$-completeness) A decision problem X is said to be $\mathcal{NP}$-complete if

1. X is in $\mathcal{NP}$;
2. Every other problem in $\mathcal{NP}$ is reducible to X in polynomial time.

Reducible essentially means that we have some algorithm that can transform an instance of a problem X, into another instance of a problem Y, in polynomial time. A common way to show that a specific problem Y is $\mathcal{NP}$-complete is to reduce another $\mathcal{NP}$-complete problem X to Y. If we can find a polynomial time solution to any problem that is $\mathcal{NP}$-complete, then we have found a solution to every problem in $\mathcal{NP}$ and this would imply that $\mathcal{NP} = \mathcal{P}$. Most researchers are of the opinion that $\mathcal{NP} \neq \mathcal{P}$ [13], however no proof of this exists.

The following famous problem is an example of a $\mathcal{NP}$-complete problem [22].

Problem: 3SAT

Input:	A logical formula L that consists of k clauses and where each clause C_i contains a disjunction of three variables.
Output:	"Yes" if there exists a satisfying assignment to the formula. "No" otherwise.

Example 2.2.1. *The following is an instance of 3SAT:*

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee \neg x_6).$$

The above formula has two clauses, each with three variables. This instance has a solution. The partial assignment $(x_1, x_4) = (1, 1)$ *satisfies the formula.*

Reducing from 3SAT is a common way to prove $\mathcal{NP}$-completeness of some problems. The reason is that many problems can be naturally encoded as a binary choice, which corresponds to assigning the value of 1 or 0 to a variable x_i in the formula for a 3SAT instance.

2.3 Boolean networks

This section is a short review of terms needed for working with Boolean networks. Boolean networks were introduced in 1969 by Kauffman [23]. A Boolean network is a graph where each node can be either expressed or not corresponding to the Boolean values of true and false, denoted respectively by 0 and 1. A more recent text on Boolean networks is [1] and includes the theory on which this chapter is based.

Each node in the network has an activation function. For Boolean networks this function is a Boolean function. A Boolean function takes as input Boolean variables and gives a Boolean output, 0 or 1.

Definition 8. (Boolean function.) A Boolean function with n inputs is a function $f(x_1, \cdots, x_n) : \{0, 1\}^n \to \{0, 1\}$.

Using vectors the notation can be simplified. Let $\boldsymbol{x} = (x_1, \cdots, x_n)$ where each x_i is a Boolean variable. Then the Boolean function $f(x_1, \cdots, x_n)$, can be written as $f(\boldsymbol{x})$.

All Boolean functions can be constructed by using one or more of the *fundamental* Boolean functions. The fundamental Boolean functions are:

1. The constant function $f = 0$ and $f = 1$;
2. The identity function $f(x) = x$, maps an element to itself;
3. Negation $f(x) = \neg x$, flips the truth value;
4. Conjunction $f(x, y) = x \wedge y$;
5. Disjunction $f(x, y) = x \vee y$;
6. Implication $f(x, y) = x \to y$.

Example 2.3.1. *Consider the following Boolean function* $f(x, y, z) = (x \vee (y \wedge z))$. *This function has three Boolean variables* $x, y,$ *and* z. *It uses two of the fundamental Boolean functions. The assignment* $\boldsymbol{a_1} = (1, 0, 1)$ *gives* $f(\boldsymbol{a_1}) = 1$, *while the assignment* $\boldsymbol{a_2} = (0, 0, 1)$ *gives* $f(\boldsymbol{a_2}) = 0$.

We are now able to define a Boolean network formally.

Definition 9. (Boolean Network) A Boolean network (BN), is a graph G, with a set of nodes $V = \{x_1, x_2, \cdots, x_n\}$, and a list of Boolean functions $F = (f_1, f_2, \cdots, f_n)$. The list of functions defines the edges. If a node x_j appears in the formula for a node x_i, then there is an edge from x_j to x_i. The edges are directed, so pairs of edges are ordered, that is (x_i, x_j) is an edge from x_i to x_j. We will denote BNs in this text as $G(V, F)$, where V is a set of nodes and F is a list of Boolean functions, one for each node.

We illustrate the definition by showing an example of a BN which we will use as a running example in this section.

Example 2.3.2. *Consider a BN that consists of the following three nodes x_1, x_2, and x_3, then $V = \{x_1, x_2, x_3\}$. To know which nodes have edges between them, we need to define a Boolean function for each node. Assume they are defined as follows:*

$$x_1(t+1) = x_1(t) \vee \neg x_3(t)$$
$$x_2(t+1) = x_1(t) \wedge x_3(t)$$
$$x_3(t+1) = x_2(t).$$

From this equation, we can now state the edges of the BN. Recall from Definition 9 that there is an edge from node x_i to x_j, if x_j appears in the formula for $x_i(t)$. Following this reasoning, we see that the set of edges is $E = \{(x_1, x_1), (x_1, x_2), (x_2, x_3), (x_3, x_1), (x_3, x_2)\}$. The whole BN can be seen in Figure 2.1.

2.4 Dynamical systems of Boolean networks

By considering how a BN evolves over time we get a *discrete dynamical system*. This system only has a finite amount of states, since each Boolean variable can only take on two values. For a network consisting of n nodes, we have 2^n different states the network can be in. We now go through two well-known updating schemes called the *synchronous* and the *asynchronous* updating scheme [33].

1. *Synchronous.* This model was introduced in [23]. Its main feature is that all nodes are updated simultaneously. This is the model that we will focus on in this thesis, mainly because it has been studied extensively and is simpler to reason about than the *asynchronous* model. Even though the synchronous update rule can be seen as simple from a biological point of view, it has been used recently to predict oncogenesis, which is how cells turn into cancer cells [14]. Another use case is for development of personalized medicine [43].

2. *Asynchronous.* In the *asynchronous* model the nodes are updated in a random order and at different timescales. The way this is usually done is to have a permutation π, and apply this to the vector $(x_1, \cdots, x_n)$ which gives $(x_{\pi(1)}, \cdots x_{\pi(n)})$. A new permutation is generated each timestep and $x_{\pi(i)}$ means that node x_i gets updated as the i:th node in that round of updates. This model was introduced in [37]. The idea of the model is that there is more choice at each state and it should, therefore, be able to model more complex interactions. For example, cases where the interactions can

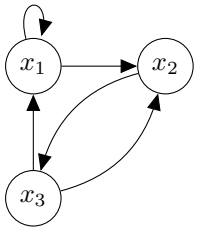


Figure 2.1: Boolean network with three nodes. Adapted from [40].

be non-deterministic, which is usually the case in biological systems. Since each state is no longer exactly determined by the previous state, there will be multiple ways to traverse the state space of the dynamics. Reasoning about the dynamics becomes more difficult in the asynchronous case, since updating is no longer completely deterministic.

In this thesis we only use the synchronous updating scheme, however, the asynchronous one is useful to know about since they are closely related.

2.4.1 Formal definition of synchronous updating

Now we formally define how to reason about the dynamics of a BN.

Definition 10. (Configuration) First denote the number of nodes in a given BN by m, then a *configuration* of a BN is a vector:

$$\boldsymbol{x} \in \{1, 0\}^m.$$

That is, $\boldsymbol{x} = (x_1, x_2, \cdots, x_m)$ where $x_i \in \{1, 0\}$ for all $1 \leq i \leq m$. A configuration describes the state of a BN completely at any time t.

At each timestep t the network changes configuration. A *transition* in the network from time t to $t + 1$ is given by:

$$\boldsymbol{x}(t + 1) = \boldsymbol{f}(\boldsymbol{x}(t)).$$

Here $\boldsymbol{x} = (x_1, x_2, \cdots, x_m)$ and $\boldsymbol{f} = (f_1, f_2, \cdots, f_m)$, where each f_i is a Boolean function assigned to each node. A transition for a single node x is written as $x(t + 1) = f(\boldsymbol{x}(t))$. To visualize how the dynamics of the systems evolve over time we can use a *state-transition table* (STT) and a *state-transition graph* (STG). In the STT we enumerate all the states the BN can be in. Each row includes a state: $\boldsymbol{s} \in \{1, 0\}^m$, and the corresponding next state: $f(\boldsymbol{s})$.

The STG is a graph where each node is one of the states $\boldsymbol{s}$. There is an edge between two nodes if both states appear at the same row in the STT. The number of nodes in the STG and the number of rows in the STT are the same: 2^m, since we enumerate each state.

Example 2.4.1. *Consider the same network as in Example 2.3.2. In Table 2.1, we have created the STT for the BN. We can see that since the network consists of three nodes that the total amount of rows in the STT is* $2^3 = 8$.

$x_1(t)$	$x_2(t)$	$x_3(t)$	$x_1(t+1)$	$x_2(t+1)$	$x_3(t+1)$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

Table 2.1: The state-transition diagram for the BN in Figure 2.1. We can see that it has $8 = 2^3$ rows. One for each configuration of the network.

2.4.2 Attractors and their basins

Starting from an initial configuration in a BN one of the following two things are bound to happen: either a fixed point will be reached in which the network will stay, or it will encounter a previously seen node and stay in a cycle of those nodes. The fixed points and cycles of states are called *attractors* of the BN. An attractor is a set $\{A_1, A_2, \cdots, A_n\}$, where each state follows from the previous one, except when $i = n$, then we "wrap around" to the first state. Formally we can state this as:

Definition 11. (Attractor) An attractor of a BN is a set of states

$$\{A_1, A_2, \cdots, A_n\},$$

where each A_i for $2 \leq i \leq n-1$ can be calculated as:

$$A_i = \boldsymbol{f}(A_{i-1}).$$

The first entry $i = 1$ is calculated in a special way $A_1 = \boldsymbol{f}(A_n)$. This is so that we "wrap around" correctly and complete the cycle.

The size of the attractor set is called the *period* or length of the attractor. The attractors that have period $p = 1$ are called *singleton attractors*, and those that have period $p \geq 1$ are called *periodic attractors*. Note that singleton attractors are just a special case of the periodic attractors. In this thesis, when we mention periodic attractors we will always mean those that have $p \geq 2$, as to distinguish them from the singleton attractors.

Another way to define attractors is as cycles in the STG. Each connected component of the graph contains one attractor. The set of states that eventually lead to an attractor is called the *basin* of the attractor. There exist different interpretations of how these attractors correspond to real biological functions of the cell. One interpretation [19] is that singleton attractors correspond to cell differentiation or apoptosis. Cell differentiation means that the cell becomes more specialized at doing a certain task. Apoptosis is programmed cell death [10], it is considered a vital part of many normal human processes such as embryo development, however, it can also be a factor in neurodegenerative diseases as well as many types of cancers, if an abnormal amount of apoptosis takes place.

A periodic attractor could correspond to different phases of the cell cycle. During various phases of the cell cycle, similar patterns of gene activity can

be seen in the cell as in the periodic attractor, thus making this a compelling interpretation. We now show an example of some attractors.

Example 2.4.2. *The attractors of the BN from Example 2.3.2 can be found by creating the STG. To create the STG, we first have to create the STT. For each row, we create a new node that corresponds to that state. We then add an edge between the node for the state at time t and the state at time $t+1$. If the two states are the same, then we have found a singleton attractor. Using this method we find that the BN has two singleton attractors:* $\{(1,1,1)\}$, $\{(1,0,0)\}$, *and also a periodic attractor:* $\{(1,0,1),(1,1,0)\}$

One observation is that if the basin of attraction is large, then many mutations can occur in the initial state of the cell and it will still end up in the same stable state at the end. The largest attractor in the network can thus be seen as the normal function of the cell and considerable mutation is needed in order to reach any of the other stable states, assuming that they have a much smaller basin of attraction [19]. Even though it is generally assumed that the most important cell states manifest themselves as the attractors with the largest basin, this is not always the case. In [12], they argue that some attractors which have small basins can be important biologically.

From these arguments, it follows that we would often be interested in knowing what kind of attractors there are in our network. The most obvious way to find them is to enumerate the whole STG. This is only possible for small networks ($n \leq 20$), since the STG grows exponentially, on the order of $\Omega(2^n)$. Consider now the problem of determining if a given BN contains a singleton attractor:

Problem: SingletonAttractor

Input:	Given a BN $G(V,F)$, determine if it has a singleton attractor.
Output:	"Yes" if the BN contains a singleton attractor. "No" otherwise.

Proposition 1. SingletonAttractor is $\mathcal{NP}$-complete, if we restrict the indegree of each node to at most k.

Proof. The proof can be found in [1]. □

The reason we restrict the indegree to some k, is that the problem is not $\mathcal{NP}$-complete for unrestricted k. In the case of unrestricted k, the problem becomes $\mathcal{NP}$-hard.

We might also be interested in finding not only singleton attractors, but also periodic attractors. The problem can be stated formally as:

Problem: PeriodicAttractor

Input:	A BN $G(V,F)$ and a period p.
Output:	"Yes" if the BN has a periodic attractor with period p, "No" otherwise.

The complexity of this problem is not known. It is believed to be $\mathcal{PSPACE}$-complete [1]. In both cases, singleton and periodic, we should not expect any polynomial time algorithms to be available. Attractor finding is closely related to the satisfiability problem, due to this, some algorithms try to take advantage

of that. One such algorithm is based on model checking and SAT-solvers [9]. It is capable of handling some hundreds of nodes. A drawback of the algorithm is that it only seems effective for networks that have their indegree restricted to some $k \leq 2$.

2.5 Network controllability

Consider a dynamical system that has entered an unhealthy state. This could, for example, be a cell or some small system inside the cell. What we would be interested in, is to guide or *control* the network away from the unhealthy state to a healthy one. The *control* in controllability, refers to the fact that we take control of some nodes and control their states through different mechanisms. A common approach is to choose a subset of the nodes, called driver nodes, formally denoted by D [18].

These driver nodes can be chosen in multiple ways. They can either be nodes from the original network, then $D \subseteq V$, or they are extra nodes added to the network. If the nodes are chosen from the original graph, then they are called *internal nodes*, otherwise we call them *external nodes* [2].

The most general form of controllability states that we should be able to guide the network from any initial state into any final state. Formally we can state this as:

Problem: AnyStateControl

Input:	A BN $G(V, F)$, an initial state $\boldsymbol{x_0}$, and a final state $\boldsymbol{x_m}$.
Output:	A minimal set of driver nodes, which can steer the BN from the initial state to the final state in m timesteps.

Proposition 2. AnyStateControl is $\mathcal{NP}$-hard.

Proof. The proof can be found in [18]. □

However, instead of allowing the final state to be an arbitrary state, it might be more reasonable to restrict the target state to attractors [17]. As mentioned earlier, attractors often correspond to biologically significant states. For this reason, we also focus on controllability to attractors in this thesis. This perspective has the other advantage that it is enough to only reach the *basin* of the attractor that we are interested in, since by reaching the basin we will eventually fall into the attractor. This gives us more possible states as control targets while still reaching the attractor. Another good reason to consider control methods when the target is an attractor, is due to a recent result regarding controllability to attractors that was shown in [17]. They showed that if we limit controllability to only attractor states, then the expected number of driver nodes for a given BN is $\mathcal{O}(\log_2 m + \log_2 n)$, where n is the number of nodes and m is the number of attractors in the network. This is a significant result, since even with large networks the expected number of attractors is small for BNs. We now state two problems that are related to the type of controllability problems we will consider when introducing our own model.

Problem: AttractorDependentControl

Input:	A BN $G(V, F)$, an initial state $\boldsymbol{x_0}$, an attractor A.
Output:	A minimal set of driver nodes that can drive the network from the initial state into the attractor.

Problem: AttractorIndependentControl

Input:	A BN $G(V, F)$, an initial state $\boldsymbol{x_0}$.
Output:	A minimal set of driver nodes that can drive the network to any attractor in the network.

In the first problem, we have an initial state and want to go from that state to the attractor. The second problem is a more general version, where we do not have a specific attractor as the goal, rather we want to find a set of nodes, such that, they allow us to control the network into any attractor. For each of these two problems, it is enough that we simply reach the basin of the target state. The complexity of these problems, to the author's best knowledge, is still unknown. After introducing our own model, in Section 4.1 we will study some special controllability problems in the context of our model.

Chapter 3

The SRG Model

In this chapter, we introduce our own model for modeling biological systems. The model is similar to the *logical regulatory graph* model defined in [32]. There a node can have one of m different expression levels. The expression level of a node is given by a function that is dependent on the incoming edges, called *regulators* to the node. In order to differentiate our model from the already quite well-established regulatory graph model, we will call our model *Strong Regulatory Graphs* (SRG). The name symbolizes that we want the nodes of the network to be in some kind of agreement amongst each other. The agreement should be *strong* in the sense that there should be as little *ambiguity* between the nodes as possible. One easy example to see where ambiguity could arise is to consider a node that is being activated and inhibited at the same time. In what way should the node be updated, if we have no way of knowing the strengths of the interactions, which is often the case [39]? In our model, we consider this kind of situation to be a *contradiction* and, thus, assign a special value to that node.

In the first part of this chapter, we introduce and define what an SRG consists of in graph-theoretic terms. The structure of the network is actually the same as used in other models that make use of regulatory graphs; the difference comes when we define the dynamics of our model later on. This gives the advantage that we can use existing regulatory networks for our model without needing to change them. In the second part, we formally define the dynamics of the SRG model. We first introduce a definition that is based on our intuition of how the model should behave, and later show that the definition can be shortened considerably. The shorter definition is easier to implement and to reason about.

The final two sections are where we show some examples of how the SRG model behaves, first on an artificial example, and then on a small, but real cancer network. By real, we mean that the network has been constructed by looking at experimental data from the real world and that something can be said about reality by studying it. We compare our SRG model to the majority voting model that the network was originally studied with.

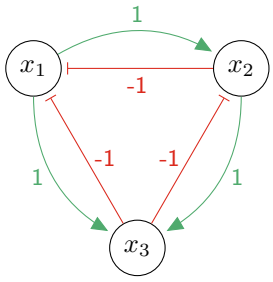


Figure 3.1: A labeled-directed graph with three nodes: x_1, x_2, and x_3. Each edge has been labeled with its corresponding number, where 1 means that the edge is an *activation* edge, while -1 means that the edge is an *inhibition* edge. The numbers will be omitted from now on, since the shape of the edges already gives its function.

3.1 Strong regulatory graphs

We begin this section with a formal definition of what an SRG is in terms of graph theory.

Definition 12. (Strong Regulatory Graphs) An SRG is an edge-labeled graph [15] SRG(V, E, ψ, F) where

1. V is a finite set of nodes $\{x_1, x_2, \cdots, x_n\}$, where n is the number of nodes in the graph.
2. E is a set of ordered pairs (e_1, e_2) where each element denotes a node. Formally, $E = \{(e_1, e_2) : (e_1, e_2) \in V \times V\}$.
3. ψ is a function that labels the edges, in this case with either $-$ or $+$. Formally, $\psi : E \to \{+, -\}$. Here $+$ means the edge is an *activation* edge, while $-$ means that the edge is an *inhibition* edge.
4. F is a set of n three-valued functions, each denoted by f_i. Each f_i is assigned to a node. Each function is defined in terms of the structure of the network. Each f_i is defined as $f_i : V \to \{1, -1, 0\}$. These values will be explained more in depth later.

Example 3.1.1. *An example of an SRG can be seen in Figure 3.1. The graph has three nodes: x_2, x_2, and x_3, so $V = \{x_1, x_2, x_3\}$. It also has six edges so $E = \{(x_1, x_2), (x_1, x_3), (x_2, x_1), (x_2, x_3), (x_3, x_1), (x_3, x_2)\}$. Three of the edges have the label $+$, that is, $\psi(x_1, x_3) = \psi(x_2, x_3) = \psi(x_1, x_2) = +$. The remaining three edges have $-$ as a label.*

In the text we will omit ψ and F when stating that we have an SRG. Instead we just write SRG(V, E). The functions in F can be derived from the edges so for that reason we also omit F.

3.2 Dynamics of the SRG model

As mentioned earlier, the nodes of a BN can be in only one of two states, 0 or 1. In this section, we expand on this idea and introduce the dynamics of the SRG

model. In the SRG model, each node can take one of three values from the set $\{1, -1, 0\}$. We also introduce a fixed update rule that apply to every node in the network. This is slightly different from the BN case where we considered arbitrary Boolean functions as update rules. The notation we use is the same as for BNs, that is $x_i(t)$ is the state of node i at timestep t, since we have three values for each $x_i \in \{1, -1, 0\}$. Similarly to BNs, a configuration is now a vector $\boldsymbol{v} \in \{1, -1, 0\}^{|V|}$. It can be noted here that the STG when using the SRG model will have $3^{|V|}$ nodes.

Each state is now described in depth. The state 1 corresponds to an active node, then it will try to either make its neighbors active or inactive depending on the outgoing edges. If the state is -1 then the node is said to be inactive, essentially having no effect on its neighbors. The state 0 means that the node is in an *ambiguous state*. This ambiguity might spread to other nodes under some circumstances. At this point it is worth elaborating on where the intuition and idea of this ambiguity comes.

The notion of ambiguity is really a consequence of what we are trying to achieve with our model. Consider for example a node that has contradicting influences, meaning that it is both being activated and inhibited at the same time. Usually we do not have the strengths of the interactions available to us, we will consider this situation to be *ambiguous*. The node could be active or inactive depending on which influence is stronger. For this reason, we introduce the additional state of *ambiguous* into our model. Now we can also see that a node will only be active or inactive if there is no ambiguity with respect to the node's influences. Intuitively this should mean that when a node is in a non-ambiguous state, then it really *should* be in that state, since otherwise the node would have been in the ambiguous state.

The update function is defined for each node v in terms of the state of the adjacent nodes and the state of node v itself. The set of adjacent nodes for activating as well as inhibiting nodes are denoted as ρ^+ and ρ^- respectively. Formally we can define these sets as:

Definition 13. (Adjacent nodes) Consider a node $v \in V$. Then the set of active nodes $\rho^+(v)$, and the set of inhibiting nodes, $\rho^-(v)$ are defined as:

$$\rho^+(v) = \{u \in V : (u, v) \in E \wedge \psi(u, v) = +\}$$
$$\rho^-(v) = \{u \in V : (u, v) \in E \wedge \psi(u, v) = -\}$$

Next we illustrate the usage of this definition by an example.

Example 3.2.1. *As an example, consider the SRG in Figure 3.1, then the sets $\rho^+(v)$ and $\rho^-(v)$ can be seen in Table 3.1.*

v	ρ^+	ρ^-
x_1	$\emptyset$	$\{x_2, x_3\}$
x_2	$\{x_1\}$	$\{x_3\}$
x_3	$\{x_1, x_2\}$	$\emptyset$

Table 3.1: Table showing the adjacent nodes for each node in the SRG in Figure 3.1.

We will also need access to the state of each node at a given time t. This we will do by defining a new set that will contain the states of the adjacent nodes. This can be done as follows:

Definition 14. (Adjacent nodes state) Consider a node $v \in V$. The state of the adjacent active nodes, $\rho_t^+(v)$, and the state of the adjacent inhibiting nodes $\rho_t^-(v)$, is defined as:

$$\rho_t^+(v) = \{u(t) : u \in \rho^+(v)\}$$
$$\rho_t^-(v) = \{u(t) : u \in \rho^-(v)\}$$

Having defined the concept of activating and inhibiting adjacent nodes as well as how to access their states, we are now ready to formally define the first version of our update function. As explained earlier, this function is the same for all nodes in the network. If we compare this with the BN case where each update function defines the edges of the network, here it is the opposite. The structure of the network defines what kind of update function each node will have.

Definition 15. (Network update (1)) Consider an arbitrary node v at time t in the network, then the next state of the node, that is, $v(t+1)$ is

- 1 (active), if
  - $\rho_t^+(v) \ni 1 \wedge \rho_t^-(v) \subseteq \{-1\}$ OR
  - $v(t) = 1 \wedge \rho_t^-(v) \subseteq \{-1\}$.
- -1 (inactive), if
  - $\rho_t^-(v) \ni 1 \wedge \rho_t^+(v) \subseteq \{-1\}$ OR
  - $v(t) = -1 \wedge \rho_t^+(v) \subseteq \{-1\}$.
- 0 (ambiguous), if
  - $\rho_t^+(v) \cap \{1, 0\} \neq \emptyset \wedge \rho_t^-(v) \cap \{1, 0\} \neq \emptyset$ OR
  - $v(t) = 1 \wedge \rho_t^-(v) \subseteq \{0, -1\} \wedge \rho_t^-(v) \ni 0$ OR
  - $v(t) = -1 \wedge \rho_t^+(v) \subseteq \{0, -1\} \wedge \rho_t^+(v) \ni 0$ OR
  - $v(t) = 0 \wedge \rho_t^+(v) \subseteq \{0, -1\} \wedge \rho_t^-(v) \subseteq \{0, -1\}$

The intuition for each case is given now. The activation case says that if a node is being activated, then it should become active, as long as every other node that is inhibiting it is inactive. The other case is that a node is already active, and all nodes that say otherwise are inactive, then the node simply remain active. The inactive case follows exactly the same line of reasoning, but instead of being activated the node is being inhibited.

The last case is more involved. The first clause represents a *contradiction* or disagreement between the nodes. The node is being activated as well as inhibited. This also includes *ambiguous* nodes as can be seen from the definition. The second clause says that if a node is currently active and a node that is inhibiting it is in the *ambiguous* state, then the node should also become ambiguous. The

reason is that the node's state *could* change, however, since the inhibiting node is ambiguous, no definite answer about the next possible state can be deduced. The same situation is mirrored for the inactive case in the third clause.

The final clause says that a node should remain ambiguous, as long as there is no new information given from the other nodes. This ensures that the network remains ambiguous until given a clear activation signal.

We could now show an example of how Definition 15 would be used in practice. The example would be long and not that pedagogic. Instead, we will now show that there is a more succinct and equivalent way to define the update functions. We first state the alternative definition, and then we prove that it is equivalent to our previously introduced definition.

Definition 16. (Network update (2)) Consider an arbitrary node v in the network at time t. The new state for v, at time $t+1$, that is $v(t+1)$ is calculated according to Table 3.2. The table should be read so that the leftmost column corresponds to the current value of the node at time t. Each rule (column two and three) is checked and if any of them apply, then the node is given the value in the rightmost column.

$v(t)$	$\rho_t^+(v)$	$\rho_t^-(v)$	$v(t+1)$
*	$\subseteq \{-1\}$	$1 \in \rho_t^-(v)$	-1
-1	$\subseteq \{-1\}$	$\subseteq \{-1, 0\}$	-1
*	$1 \in \rho_t^+(v)$	$\subseteq \{-1\}$	1
1	$\subseteq \{-1, 0\}$	$\subseteq \{-1\}$	1
Otherwise			0

Table 3.2: Table of rules defining how updates are done in the network. The leftmost column corresponds to the value of the node at time t, while the last column is the new value at time $t + 1$.

We now prove that Definition 15 and Definition 16 of the update function are equivalent. The proof is essentially by construction where we enumerate all possibilities for the update function and notice that many cases collapse into more simple statements. The shorter form of the update function is stated in Definition 16. The proof of Lemma 1 shows that both definitions are equivalent.

Lemma 1. The update function in Definition 15 and Definition 16 are equivalent.

Proof. We now prove Lemma 1. First consider an arbitrary node v in some SRG with incoming activation and inhibition edges. See Figure 3.2 for a graphical representation of this situation. Consider the set $\rho_t^+(v) \subseteq \{1, -1, 0\}$, we can enumerate all possible combinations of activation edges by considering the power set of $\{1, -1, 0\}$, namely $\mathcal{P}(\{1, -1, 0\})$. The size of this set is $|\mathcal{P}(\{1, -1, 0\})| = 2^{|\{1,-1,0\}|} = 2^3 = 8$. The exact same calculation applies for the inhibiting edges. In total, we have $8 \cdot 8 = 64$ different cases.

For each of the 64 different cases, we also need to take into consideration the value of the node v, since that is used during the update. Now, v can be in three different states, so for each of the 64 cases we need to take these three values into account. In total, we have $64 * 3 = 192$ different cases to consider.

Next we construct a table of all different cases. For each case we consider what value the update function would give. Table 3.3 shows the calculations. The table includes the three additional cases for the values $\{1, -1, 0\}$ on the

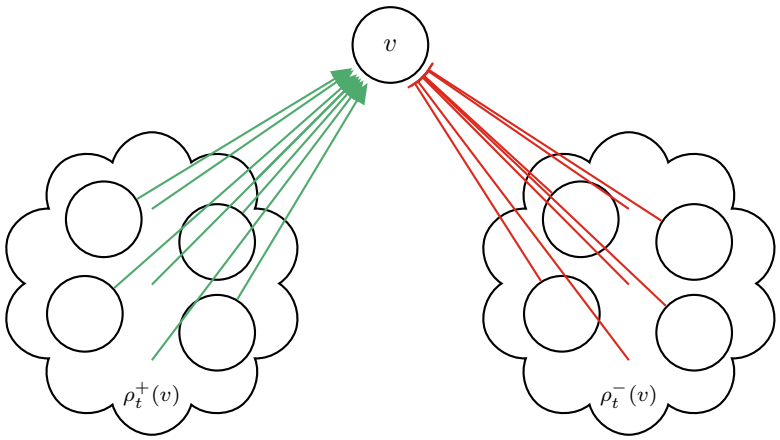


Figure 3.2: The neighborhood of a node v considered in the proof of Lemma 1. The node v is an arbitrary node in the network, $\rho_t^+(v)$ are all its incoming activation edges, and $\rho_t^-(v)$ are all incoming inhibition edges.

same row, that is, even though the table only has 64 rows, all 192 cases are included.

By grouping together all outcomes from Table 3.3 that gives a certain output, we can find common factors among the cases. In Table 3.4, we have collected all rows where $v(t+1)$ is -1. The last row of each table collects the common factors into a single case. Now we have a more compressed version of the update function in the case of -1. A similar table can be made for the cases where the node becomes active. The end result is two new rules for when a node will become active.

By combining these four new rules, we get the first four rows of Table 3.2. All other cases are mapped onto 0, and correspond to the last row of Table 3.2, meaning that if none of the four rules apply, then v will become ambiguous. We have now constructed Table 3.2 (Definition 16) and have thus completed the proof of Lemma 1.

□

What has taken place is that all the four different cases for the ambiguous clause in Definition 15 have collapsed into a single statement. This new statement is only true whenever the statements for activation and inhibition are false. Thus, when updating, we only need to check the conditions for becoming active or inactive and if neither of those are true, then we say that the node will be ambiguous in the next configuration. We will now show, by example, how to construct the set of update functions using Definition 16.

Example 3.2.2. *Consider the SRG in Figure 3.1. We are interested in constructing the set F of update functions using Definition 16. To accomplish this, we first need to construct the two sets that contain the adjacent nodes, namely ρ^+ and ρ^-. We have already done this earlier in Table 3.1. Now we can directly use Definition 16 to define each update function f_i, for each of the nodes in the SRG as follows:*

$\rho_t^+(x_i)$	$\rho_t^-(x_i)$	$v(t) \to v(t+1), \forall v(t) \in \{1,-1,0\}$
$\emptyset$	$\emptyset$	$1 \to 1; -1 \to -1; 0 \to 0$
$\emptyset$	$\{-1\}$	$1 \to 1; -1 \to -1; 0 \to 0$
$\emptyset$	$\{0\}$	$1 \to 0; -1 \to -1; 0 \to 0$
$\emptyset$	$\{1\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\emptyset$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\emptyset$	$\{1,-1\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\emptyset$	$\{1,0\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\emptyset$	$\{1,-1,0\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\{-1\}$	$\emptyset$	$1 \to 1; -1 \to -1; 0 \to 0$
$\{-1\}$	$\{-1\}$	$1 \to 1; -1 \to -1; 0 \to 0$
$\{-1\}$	$\{0\}$	$1 \to 0; -1 \to -1; 0 \to 0$
$\{-1\}$	$\{1\}$	$1 \to 1; -1 \to -1; 0 \to -1$
$\{-1\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{-1\}$	$\{1,-1\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\{-1\}$	$\{1,0\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\{-1\}$	$\{1,-1,0\}$	$1 \to -1; -1 \to -1; 0 \to -1$
$\{0\}$	$\emptyset$	$1 \to 1; -1 \to 0; 0 \to 0$
$\{0\}$	$\{-1\}$	$1 \to 1; -1 \to 0; 0 \to 0$
$\{0\}$	$\{0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{0\}$	$\{1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{0\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{0\}$	$\{1,-1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{0\}$	$\{1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{0\}$	$\{1,-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1\}$	$\emptyset$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1\}$	$\{-1\}$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1\}$	$\{0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1\}$	$\{1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1\}$	$\{1,-1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1\}$	$\{1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\emptyset$	$1 \to 1; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\{-1\}$	$1 \to 1; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\{0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\{1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\{1,-1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{-1,0\}$	$\{1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1\}$	$\emptyset$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1,-1\}$	$\{-1\}$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1,-1\}$	$\{0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1\}$	$\{1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1\}$	$\{1,-1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1\}$	$\{1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1\}$	$\{1,-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,0\}$	$\emptyset$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1,0\}$	$\{-1\}$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1,0\}$	$\{0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,0\}$	$\{1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,0\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,0\}$	$\{1,-1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,0\}$	$\{1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,0\}$	$\{1,-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1,0\}$	$\emptyset$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1,-1,0\}$	$\{-1\}$	$1 \to 1; -1 \to 1; 0 \to 1$
$\{1,-1,0\}$	$\{0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1,0\}$	$\{1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1,0\}$	$\{-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1,0\}$	$\{1,-1\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1,0\}$	$\{1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$
$\{1,-1,0\}$	$\{1,-1,0\}$	$1 \to 0; -1 \to 0; 0 \to 0$

Table 3.3: Table of possible updates for a node. The last column shows the next state, $v(t+1)$, depending on the value of the previous state, $v(t)$.

$\rho_t^+(v)$	$\rho_t^-(v)$	$v(t) \to v(t+1)$
$\emptyset$	$\{1\}$	$* \to -1$
$\emptyset$	$\{1,-1\}$	$* \to -1$
$\emptyset$	$\{1,0\}$	$* \to -1$
$\emptyset$	$\{1,-1,0\}$	$* \to -1$
$\{-1\}$	$\{1\}$	$* \to -1$
$\{-1\}$	$\{1,-1\}$	$* \to -1$
$\{-1\}$	$\{1,0\}$	$* \to -1$
$\{-1\}$	$\{1,-1,0\}$	$* \to -1$
$\subseteq \{-1\}$	$1 \in \rho^-(v)$	$* \to -1$

$\rho_t^+(v)$	$\rho_t^+(v)$	$v(t+1)$
$\{-1\}$	$\emptyset$	$* \to -1$
$\{-1\}$	$\{-1\}$	$* \to -1$
$\{-1\}$	$\{0\}$	$* \to -1$
$\{-1\}$	$\{-1,0\}$	$* \to -1$
$\emptyset$	$\emptyset$	$* \to -1$
$\emptyset$	$\{0\}$	$* \to -1$
$\emptyset$	$\{-1\}$	$* \to -1$
$\emptyset$	$\{-1,0\}$	$* \to -1$
$\subseteq \{-1\}$	$\subseteq \{1,0\}$	-1

Table 3.4: In these two tables we have collected all possibilities where the node value at time $t+1$ is -1. The last row of each table collects the common factors of all the above rows. Thus collapsing eight different cases into a single statement. The notation, $* \to x$ means that, regardless of the initial value of $v(t)$, $v(t+1)$ will always be x in the next state.

$$f_1(x_1) = \begin{cases} -1, & (x_2(t) = -1 \vee x_3(t) = -1) \vee \\ & (x_1(t) = -1 \wedge x_2(t) \subset \{-1,0\} \wedge x_3(t) \subset \{-1,0\}) \\ 1, & x_1(t) = 1 \wedge (x_2(t) = -1 \wedge x_3(t) = -1) \\ 0, & \textit{otherwise} \end{cases}$$

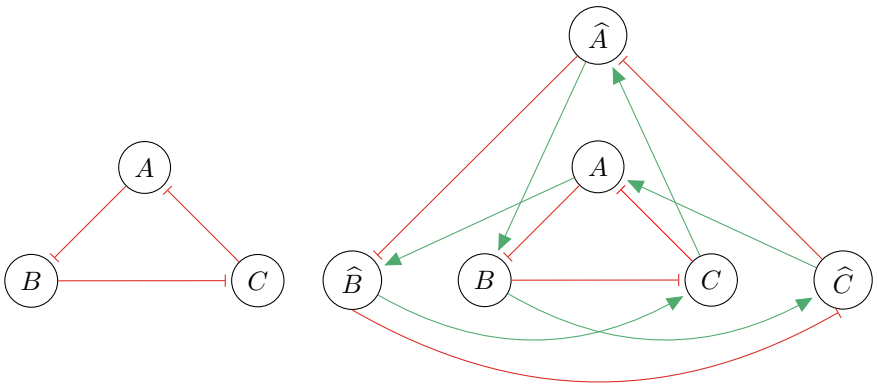
$$f_2(x_2) = \begin{cases} -1, & (x_1(t) = -1 \wedge x_3(t) = 1) \vee \\ & (x_2(t) = -1 \wedge x_1(t) = -1 \wedge x_3(t) \subset \{-1,0\}) \\ 1, & (x_1(t) = 1 \wedge x_3(t) = -1) \vee \\ & (x_2(t) = 1 \wedge x_1(t) \subset \{-1,0\} \wedge x_3(t) = -1) \\ 0, & \textit{otherwise} \end{cases}$$

$$f_3(x_3) = \begin{cases} -1, & (x_3(t) = -1 \wedge x_1(t) = -1 \wedge x_2(t) = -1 \\ 1, & (x_1(t) = 1 \vee x_2(t) = -1) \vee \\ & (x_3(t) = 1 \wedge x_1(t) \subset \{-1,0\} \wedge x_2(t) \subset \{-1,0\}) \\ 0, & \textit{otherwise} \end{cases}$$

Now that we have defined each update function we simply assign them to the nodes. For each configuration we now have a rule for updating each state. Even with our shorter definition, this process takes some time to write down by hand, however, it can easily be implemented on a computer. If a node's adjacent nodes, either the active or the inactive ones, is the empty set, then the update rule becomes shorter, since we can exclude some rules from the definition by concluding that those rules can never be satisfied.

3.3 Modeling examples

In this section, we work through a small example to demonstrate the topics introduced in this chapter. Our modeling example comes from synthetic biology, and is called the repressilator [11]. The repressilator was introduced, because many "simple" biological functions were poorly understood, even though they



(a) The original repressilator model. Each node inhibits the next one in the cycle.

(b) Our modified version of the repressilator.

Figure 3.3: On the left is the original repressilator. On the right is our modified version.

had been extensively researched. With the repressilator, they tried to create one of these systems, by hand, in the hope that it would lend a better understanding of how the real biological counterpart works. The repressilator uses three repression genes in a cycle, as can be seen in Figure 3.3. In the figure, we see that we have three nodes A, B, and C, where each node represses the next one in the cycle. We do not need to consider what genes are actually used in the real biological repressilator, the only important property is that the genes repress each other in a cycle.

The model represented in Figure 3.3 is compatible with the SRG model, but using it directly in the presented way would be incorrect. There is an implicit assumption in the repressilator, which is that if a node is *not* being *actively* repressed, then it should be active. One idea to make this assumption explicit can be seen in Figure 3.3. In the figure, we can see that three additional nodes have been added. The idea of the new nodes is that they should always be in the opposite state to their counterpart node. We make this idea more clear by an example.

Example 3.3.1. *Assume that the node $\widehat{A}$ is in the state -1, then A should be active and in the next state $\widehat{B} = 1$, since A activates that node. Similarly, $B = -1$, since A represses that node. The same reasoning can be applied to C.*

If our model of the repressilator is correct, then we should see some attractors that have a period of length greater than one. This is motivated by the biological function of the repressilator that behaves oscillatory. After simulating the repressilator, we end up with an STG that has 729 different states. The whole STG can be seen in Figure 3.4.

If we analyze the repressilator's attractors, we see that they all have the kind of oscillatory behavior that we described earlier. The attractor

$$\begin{aligned} C = &\{\{1,1,1,-1,-1,-1\},\\ &\{-1,-1,-1,1,1,1\}\} \end{aligned}$$

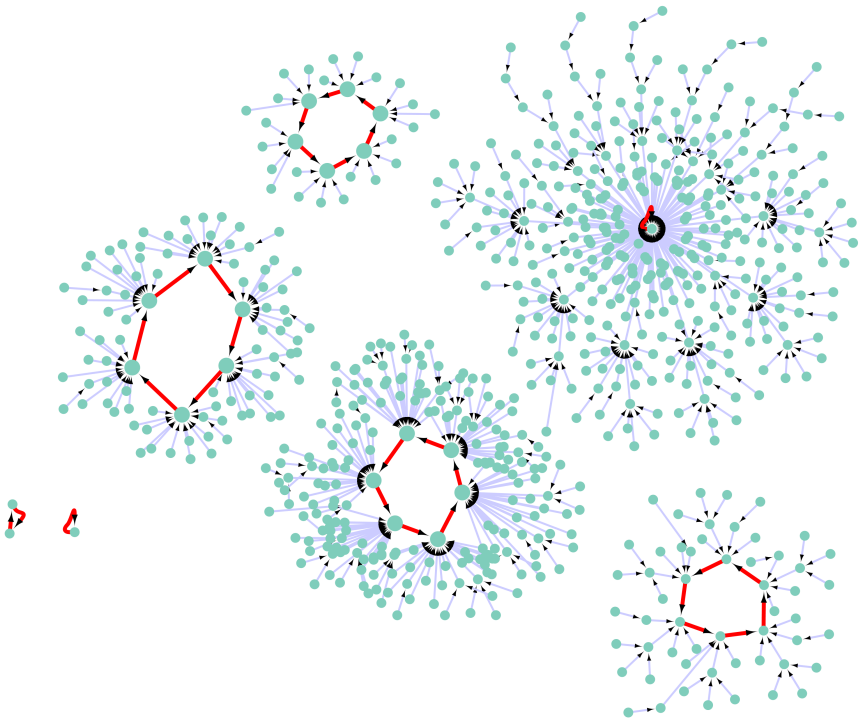


Figure 3.4: The STG for the modified repressilator from Figure 3.3. The attractors can be seen as cycles in the graph. The red edges do not signify inhibition rather they just highlight the attractors.

is perhaps the one where this behavior is the most obvious. This attractor can also be seen on the very left side of Figure 3.4. In this attractor, we see that in the first state, all the nodes $A, B,$ and C are activated, then they become inactive, since they all repress each other. Due to how the network is connected, we see that the three other nodes in the network, $\widehat{A}, \widehat{B},$ and $\widehat{C}$ become active. In the next state, the same behavior occurs, only now the roles of the nodes are switched. This process will continue indefinitely.

3.4 Comparison to other models

In this section, we compare the SRG model to two already existing models. To the author's best knowledge, the concept of unanimous influences has not been studied before. Still, there are some models that bear similarities to our model, both Boolean and multi-valued models.

3.4.1 Majority voting

Majority voting bears some resemblance to our model. In majority voting, a predefined amount of the influences must be in agreement before a change can take place. This type of modeling is often done with the help of *Boolean Threshold Networks* (BTN) [42]. Here, instead of using a different Boolean function for each node, only one function is used. This function is based on the structure of the network, and a predefined *threshold*. The update function is defined as follows:

$$x_i(t) = \boldsymbol{f}\left(\sigma_i^1, \cdots, \sigma_i^j, \cdots, \sigma_i^k\right) = \begin{cases} 1, & \sum_{j=1}^{k_i} a_{ij}\sigma_{ij}(t) \geq \theta_i \\ 0, & \sum_{j=1}^{k_i} a_{ij}\sigma_{ij}(t) \leq \theta_i \\ \sigma_i(t), & \sum_{j=1}^{k_i} a_{ij}\sigma_{ij} = \theta_i. \end{cases}$$

Here, θ_i is called the *threshold* of the network, and σ_{ij} is 1 if the edge between node i and j is an activation edge, and 0 if it is an inhibition edge. The parameter θ_i is adjustable. This value is usually set to 0, then only one activation signal is needed for a node to become active. If there is an equal number of activations and inhibitions, then the state is unchanged. The term "majority voting" stems from the fact that the influences which have majority determine the next state. In some small networks, BTNs can successfully predict the behavior of real biological systems [42]. BTNs are similar to the SRG model, in the sense that the update function is defined by the topology of the network. Still, the BTN model places assumptions on the strengths of the interactions. As an example, assume that a node has three activators and two inhibitors. In biochemistry, inhibition is often seen as the stronger "force" [41], and it could be possible that the node should become inactive instead of active. Our model does not make any assumption about the strengths of the interactions, instead we model it as a lack of data, hence the ambiguous state in our update function.

3.4.2 Multi-valued model

Many multi-valued models have also been introduced for modeling biological networks [38, 8, 4]. The motivation is that the Boolean model is too coarse to

model reality. One such model was introduced in [37]. The model is defined as asynchronous, but can be used in the synchronous framework as well. The network topology that is used is the same as in the SRG model, with activators and inhibitors. The idea is to use multiple logical levels for each node, up to k levels. The difference between this model and the SRG model, is that the levels can only increase and decrease monotonously, that is, decrease or increase by one value at a time. In this model, some edges can also be inactive or active, depending on the current state of the node. This model is similar to ours in the case when $k = 3$, however, there is no way to model unanimity in the sense that we are after. This is because in this model, node values are only updated monotonously, which is incompatible with our definitions.

3.4.3 State-space comparison of a small cancer model

In this section, we look at a specific cancer network and see how the state space of the majority voting scheme compares to the SRG model. The reason we focus only on majority voting is first: the network we are comparing against was originally developed using a majority voting model. Secondly, the multi-valued model introduced above needs to have a specific multi-valued function assigned to each node, thus we cannot simply work on the network structure level if we wish to have a comparison to the multi-valued model as well. For these two reasons, we focus only on comparing the SRG model to the majority voting model.

Network

In order to make this comparison somewhat tractable, we have limited ourselves to a model of modest size. Specifically, we will use a network introduced in [41]. The purpose of the network was to study the impact of the miR-17-92 cluster on the regulation of the network as a whole. The network itself is linked to the G1/S transition in the cell cycle. It has been shown that an accurate transition from the G1 state to the S state in the cell cycle is vital, and misregulation at this stage is linked to cancer formation of the cell, so-called oncogenesis [5]. In fact, they showed with the help of a Boolean model that miR-17-92 is critical in the suppression of the G1/S transition and misregulation of miR-17-92 can result in an increase of cancer proliferation among cells.

The model used in [41] is a specific case of a majority voting model. To be more precise, it has the parameter θ set to 0. The network model itself consists of eight nodes and can be seen in Figure 3.5. The network was deduced from experimental data. Many of the genes in the network are either oncogenes, for example: Cdc25A and Cdk, while others are tumor suppressors, such as: pRb and p25. An oncogene is a gene that might cause cancer, while tumor suppressors on the other hand work against cancer in different ways: they inhibit the growth of cancerous cells, stops them from dividing too quickly, repairs damaged DNA, and etc. [27]

In [41] they found six different attractors, while our model had 60. Due to the size of the networks, 256 nodes for the Boolean model, and 6561 nodes for the SRG model, it is difficult to create a meaningful image of the STG to show. Instead we rank the attractors by size and show them in a table. In Table 3.6, we can see all attractors found by using a BTN model for the cancer network.

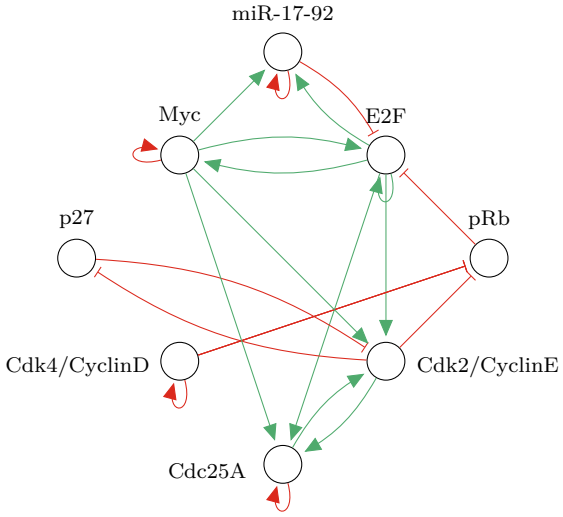


Figure 3.5: The network that we consider in order to make the state-space comparison. The network has eight nodes where green edges represent activation and blue or black edges represent inhibition. The purpose of the network is to model the influence of the miR-17-92 gene on the G1/S transition in the cell cycle.

Basin size	miR-17-92	Myc	E2F	p27	pRb	Cdk4	Cdk2	Cdc25A
184	0	0	0	0	0	0	1	1
48	0	0	0	1	0	0	0	0
16	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0

Table 3.5: The six attractors found by using a majority voting model in a Boolean framework as shown in [41]. All attractors are singletons. Cdk2 and Cdk4 have shortened names in the table for typographical reasons.

The 10 attractors with the largest basin found with the SRG model can be seen in Table 3.6.

We can note some interesting differences between the attractors found by the SRG model and the BTN model. For example, we can see that the node Cdc25A is never active in the SRG model. The reason is that if a node has a negative self-loop, then it can never be active in an attractor. Another difference that can easily be seen, is that the only node that is active at all, in any attractor, is Cdk2. This is partly due to the number of negative self-loops that the network has. However, there are other features of the network that causes this type of behavior in the SRG model. We will go into more detail on this in the next chapter, where we show that in the SRG model some nodes cannot have a certain value in any attractor whatsoever.

At this point, we do not go any deeper into some comparison about the differences between the BTN and SRG model. We show this example here to illustrate that the SRG model does produce different results than some already existing model.

Basin size	miR-17-92	Myc	E2F	p27	pRb	Cdk4	Cdk2	Cdc25A
1472	0	0	0	0	-1	-1	1	0
1152	0	0	0	0	-1	-1	0	0
736	0	0	0	-1	-1	0	1	0
616	0	0	0	-1	-1	-1	0	0
576	0	0	0	0	0	0	0	0
576	0	0	0	0	0	-1	0	0
288	0	0	0	0	-1	0	0	0
272	0	0	0	-1	-1	0	0	0
84	-1	-1	-1	-1	-1	-1	1	0
80	-1	-1	-1	0	-1	-1	0	0

Table 3.6: The top 10 attractors found with the SRG model ranked by their basin size. All attractors are singletons. Cdk2 and Cdk4 have shortened names in the table for typographical reasons.

Chapter 4

Controllability of the Model

In this chapter, we try to answer some questions regarding the controllability of our model. What we will be after, is some special *target attractor* that we want to control the network into. This will be the focus of this chapter.

The first topic we will cover is that of a transformation of a given SRG network into an equivalent Boolean network. This has the benefit of allowing us to apply existing tools and algorithms that have already been developed for BNs. We will show that the transformation can be carried out efficiently. The transformation we show is not a perfect solution for the problems we present in this chapter, however, it is nonetheless a good starting point.

In the second section of this chapter, we present a more efficient approach to finding target attractors, based on inspecting the SRG. We will show that we can determine, purely based on the SRG, if some special *target attractors* can exist. We will be spending most of the time on this topic, since we need to develop some new theory and definitions to explain the concepts in depth.

In the last section, we introduce some special controllability problems. Here, we do not present any complete solutions, rather we show that some existing algorithms and ideas can be applied to these problems. We will show partial solutions, involving the theory developed in this chapter, as well as some previously shown results. The issues we present are still open research problems, so we cannot directly state that any of the given partial solutions are optimal, or even correct.

4.1 Target attractor detection

In this section, we consider a problem that is related to the controllability problem. The problem is, given some set of nodes, called *target nodes* and a *target configuration* for those nodes, does the dynamical system of the SRG have an attractor where this target configuration always holds? An equivalent statement is: does there exist an attractor, where the target values stay constant? If we can answer this question, then the next question would be: how do we control the network to this attractor? It should be noted here that we are considering a more constrained version of controllability. Instead of controlling the network to any given state, we are requiring that the target configuration should belong to an attractor. This is a reasonable requirement, since most states will be

nonsense states that have no real biological meaning, while attractors usually represent some core function of the cell [16]. We formulate the question as a decidability problem, called **UnanimousAttractorDetection** as follows:

Problem: UnanimousAttractorDetection

Input:	A target set $T \subseteq V$, and a target configuration $\alpha = \{1, -1\}^{\lvert T \rvert}$
Output:	Is there an attractor C, such that $C_i\lvert_T = \boldsymbol{\alpha}$, for all $1 \leq i \leq l$, where l is the period of C?

We will call attractors that satisfy **UnanimousAttractorDetection**, *target attractors*.

The first question we should ask ourselves is: does such attractors even exist in our model? The case is certainly true for $l = 1$, since we can choose $\alpha = \{-1\}^{\lvert V \rvert}$ and this is always a singleton attractor in our model. The question is more interesting when $l \geq 2$. Having attractors, where some nodes stay constant for longer periods of time, symbolizes that they have some sort of "agreement" or unanimity between them. We now show with an example that there exist attractors with $l \geq 2$ and $\alpha \neq \{-1\}^{\lvert V \rvert}$. This shows that the problem is well defined and that our search is not meaningless.

Example 4.1.1. *Consider the graph in Figure 4.1. Assume that we have the target set $V = \{x_1\}$ and have $\boldsymbol{\alpha} = \{1\}$, that is, we want to find an attractor where the node x_1 remains active throughout. The network in Figure 4.1 has such an attractor. It consists of the following states:*

$$\begin{aligned} C = &\{\{1, 0, -1, -1, 0, -1, -1\}, \{1, -1, -1, 0, 0, -1, -1\}, \\ &\{1, -1, 0, 0, -1, -1, -1\}, \{1, 0, 0, -1, -1, -1, -1\}\} \end{aligned}$$

It can easily be seen that C is a target attractor, since it satisfies the needed properties, namely that the target value stays constant through the attractor. If we denote the states in C by C_i then we have:

$$\begin{aligned} C_1|_{\{x_1\}} &= 1 \\ C_2|_{\{x_1\}} &= 1 \\ C_3|_{\{x_1\}} &= 1 \\ C_4|_{\{x_1\}} &= 1 \end{aligned}$$

We can see that the target node, x_1, remains constant throughout the attractor, as it should.

One obvious way to solve **UnanimousAttractorDetection** is to create the whole STG and then use a technique such as depth-first search, which is able to find cycles in graphs. The drawback of this method is that generating the STG takes exponential time and memory, namely $\Omega(3^n)$, so something more efficient is desired. We can achieve a marginal improvement by keeping the nodes in the target set constant. This makes the number of possible states we need to check fewer.

Proposition 3. UnanimousAttractorDetection can be solved in $\mathcal{O}(3^{n-\lvert\alpha\rvert})$.

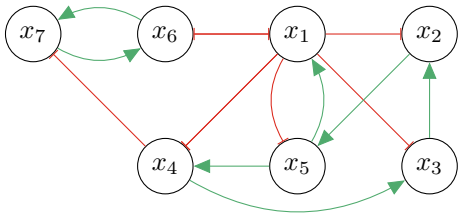


Figure 4.1: Network that is used as an example to show a constant attractor with a period greater than one.

Proof. Keep each node in the target set constant when generating the STG. This reduces the number of possible combinations to $3^{n-|\alpha|}$. □

Our first try at a more efficient solution, is to reduce the SRG network into a BN, which we do in the next section.

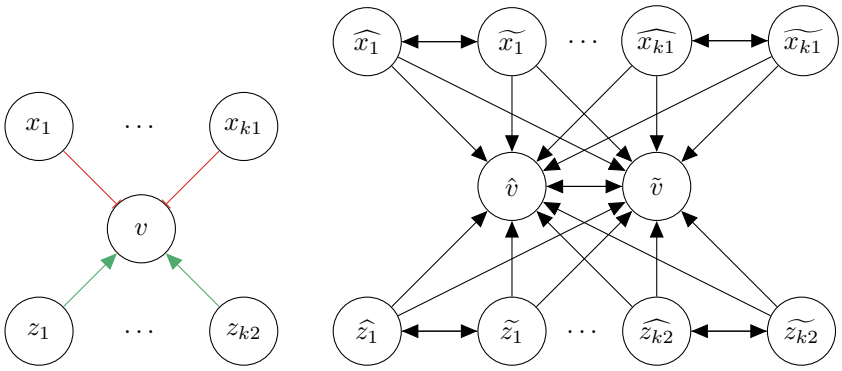
4.2 Boolean network transformation

In this section, we present an algorithm for transforming any network expressed in the SRG model into an equivalent Boolean one. Since we are interested in controllability, we will consider the two networks equivalent, if the attractors of the two networks have a one-to-one mapping between them. If the attractors were somehow different, then any results found by this transformation would not be useful, since we would not be reasoning about the same state space anymore. The usefulness of this transformation is mostly in the controllability sense; since there has been some work on controllability in BNs, it would be useful for us to be able to apply those results to the SRG model as easily as possible. Transforming a given SRG into a BN is one way to achieve this. Another benefit that we can mention, is that the transformation will provide us with an upper bound for many complexity problems. As we know, many attractor and controllability-related problems are $\mathcal{NP}$-hard in a BN context. This reduction would thus give an upper bound on **UnanimousAttractorDetection**, since it is a special case of **PeriodicAttractor** which, as mentioned earlier, is believed to be $\mathcal{PSPACE}$-complete.

To begin this section, we will first give a high-level intuitive explanation of how the transformation is done, and in what sense the newly created BN is equivalent to the SRG it was derived from. After that, we will state the transformation in more formal terms. We finish this section by presenting the complete transformation algorithm and show that the transformation can be done in polynomial time.

4.2.1 High-level description

The intuitive idea behind the algorithm is that we only need two bits to encode our three-value logic with binary states. With two bits, we can represent $2^2 = 4$ different states, thus our three states can be encoded. The next step comes from inspecting the update function in Definition 16. As can be seen, each update clause is essentially a logical statement with a binary outcome. The statements



(a) Arbitrary node v in an SRG before the transformation.

(b) The same node v, but now after the transformation. We can see v has now been split into two nodes $\hat{v}$ and $\tilde{v}$. Every other nodes has also been split into two new nodes.

Figure 4.2: On the left we see a node in the SRG before the transformation. The picture on the right demonstrates what the node v and its immediate adjacent nodes will look like in the BN after the transformation.

are also disjunct from each other. Taking advantage of this fact, we will set one node to be the "active" node $\hat{v}$, and another to be the "inactive" node $\tilde{v}$. The node $\hat{v}$ will have the update function that corresponds to the state "1" in the SRG model, similarly we assign to the node $\tilde{v}$ the update function that corresponds to "-1". The state "0", is modeled in the binary case implicitly, due to the fact that if both $\hat{v}$'s and $\tilde{v}$'s update functions evaluate to false, then that would correspond to the *otherwise* update clause in Definition 16. Thus, each time both of the functions evaluate to false, the original node v would be updated to "0" in the SRG model.

The notion of activating and inhibiting edges is translated implicitly in the Boolean formula for each new node. This new BN that we create will only use unlabeled edges that have no special meaning biologically.

In Figure 4.2, we show graphically the idea behind the transformation. We can see that the node v is split into two new nodes: $\hat{v}$ and $\tilde{v}$. Here, $\hat{v}$ is the "active" node and $\tilde{v}$ is the "inactive" node. We can also see that all other nodes have been split into two new nodes. At this point, we have not, yet, defined a Boolean function for each node, so we should not draw any edges between the nodes, if we are being strict. This is because in a BN, the edges are defined by the update functions, and not the other way around as in the SRG model. However, we jump slightly ahead here and show the edges for illustration purposes. We will see later that our transformation will create exactly such a function that gives rise to such edges.

4.2.2 Formalization

Consider a node v in the SRG. In the transformed BN, v will correspond to two new nodes, $\hat{v}$ and $\tilde{v}$. The following encoding scheme is used for the active,

inactive, and ambiguous states:

$$\begin{cases} (\hat{v}, \tilde{v}) = (1,0) \implies \text{Active} \\ (\hat{v}, \tilde{v}) = (0,1) \implies \text{Inactive} \\ (\hat{v}, \tilde{v}) = (0,0) \implies \text{Ambiguous} \end{cases} \tag{4.1}$$

In Equation (4.1) we can see the encoding. The idea is that the $\hat{v}$ node only has the value 1, when the node is active in the SRG. The same idea applies to $\tilde{v}$. Now, if neither of these is 1, then this should correspond to the ambiguous state as we explained earlier. This is encoded by $(0,0)$. We can see that $(1,1)$ is unused, however, it does not matter since we only have three values to encode. Assuming that the formulas we present later are correct, we can decode a state from the transformed BN back into the SRG model by using a decoding function. We formally define it as follows:

Definition 17. (Decoding Function) The *decoding function* is a function

$$\pi : \{1,0\}^{|2V|} \to \{1,-1,0\}^{|V|}$$

where $|V|$ is number of nodes in the given SRG under consideration. The decoding function is defined as:

$$\pi(\boldsymbol{v}) = (\tau(v_1, v_2), \ldots, \tau(v_{2|V|-1}, \tau(v_{2|V|})). \tag{4.2}$$

Here τ is a function $\tau : \{1,0\} \times \{1,0\} \to \{1,-1,0\}$ and is defined as:

$$\tau(v_1, v_2) = \begin{cases} 1, & (v_1, v_2) = (1,0) \\ -1, & (v_1, v_2) = (0,1) \\ 0 & (v_1, v_2) = (0,0) \end{cases} \tag{4.3}$$

Using the decoding function, we can easily decode states from the transformed BN back into our SRG model. We make a small note here that the transformed BN will have $3^{|2V|}$ states, since we are introducing two new nodes for each node in the SRG. This means that the state space of the BN will be larger than our original state space for the SRG, and thus include some states that have no meaning in the SRG framework. We will ignore these states, since they would have no meaning in the SRG.

We now introduce the definitions of the Boolean functions that will be used in the transformed BN. In the update rules in Definition 3.2, we can see that we need to check the following conditions, one from each column in the table:

$$\begin{cases} \rho^+(v) \subseteq \{-1\} & \text{“All adjacent activation nodes are inactive”} \\ \rho^-(v) \subseteq \{-1\} & \text{“All adjacent inhibition nodes are inactive”} \\ 1 \in \rho^+(v) & \text{“At least one adjacent activation node is active”} \\ 1 \in \rho^-(v) & \text{“At least one adjacent inhibition node is active”} \\ \rho^+(v) \subseteq \{-1,0\} & \text{“All adjacent activation nodes are inactive or ambiguous”} \\ \rho^-(v) \subseteq \{-1,0\} & \text{“All adjacent inhibition nodes are inactive or ambiguous”} \end{cases} \tag{4.4}$$

In Equation (4.4), we can see that we need to encode quantified statements with logical formulas. We first consider the simpler case when we only have one

node, and generalize the concept later. Now, instead of checking if all nodes are for instance active, we instead consider only one node. The states we want to formulate according to Equation (4.4) are: *a node is active*, *a node is inactive*, and *a node is inactive or ambiguous*. The first statement can be written as follows:

$$\hat{v} \wedge \neg\tilde{v}. \tag{4.5}$$

We can see that the formula in Equation (4.5), is only true if $(\hat{v}, \tilde{v}) = (1, 0)$, which is exactly how we encode the active node in Equation (4.1). We can now construct a similar formula for an inactive node:

$$\neg\hat{v} \wedge \tilde{v}. \tag{4.6}$$

Equation (4.6) will only be true if: $(\hat{v}, \tilde{v}) = (0, 1)$, which is how we encoded inactivity earlier. Now only the last statement remains. We want to create a logical formula for when a node is either inactive or ambiguous. We already know how to check if a node is inactive from Equation (4.6), so we only need to combine that with the formula for a node being ambiguous. We combine them using a disjunction since either of the states can be true:

$$(\neg\hat{v} \wedge \tilde{v}) \vee (\neg\hat{v} \wedge \neg\tilde{v}). \tag{4.7}$$

We see that the first part of the disjunction is Equation (4.6). The right part $\neg\hat{v} \wedge \neg\tilde{v}$, is only true if: $(\hat{v}, \tilde{v}) = (0, 0)$, which is how we encoded the ambiguous state. Equation (4.7) can be simplified to a more intuitive form:

$$\begin{aligned}
(\neg\hat{v} \wedge \tilde{v}) \vee (\neg\hat{v} \wedge \neg\tilde{v}) &\equiv (\neg\hat{v} \wedge (\tilde{v} \vee \neg\tilde{v})) && \text{(De Morgan)} \\
&\equiv (\neg\hat{v} \wedge T) && \text{(Law of Excluded Middle)} \\
&\equiv \neg\hat{v}.
\end{aligned}$$

The whole formula has collapsed into $\neg\hat{v}$. This is a natural result, since the formula now simply says: *v is not active*, which is equivalent to saying: *v is either inactive or ambiguous*, which is what we had before. Now we can state all the Boolean formulas together at once:

$$\begin{cases}
\hat{v} \wedge \neg\tilde{v} & v \text{ is active;} \\
\neg\hat{v} \wedge \tilde{v} & v \text{ is inactive;} \\
\neg\hat{v} & v \text{ is not active.}
\end{cases}$$

We now have the logical formulas we need in order to create Boolean functions that correspond to the statements in Equation (4.4). The equation has six statements, three for activation and inhibition each. We only cover one of each sort since they are symmetric to each other. If we first consider the statement "All adjacent activation nodes are inactive", that is, $\rho^+(v) \subseteq \{-1\}$. This statement is quantified over all adjacent nodes, so we might think that we would need quantifiers in the formula. This is not the case, since the set of adjacent nodes is finite for each node. Quantified statements are essentially infinite conjunctions, however since our domain is finite, we can stay within propositional logic for the formulas. The following formula checks that all adjacent nodes are inactive:

$$\bigwedge_{w \in \rho^+(v)} (\neg \widehat{w} \wedge \widetilde{w}).$$

What is important to note here, is that v is a node in the SRG, while $\widehat{w}$ and $\widetilde{w}$ are nodes in the transformed BN.

The next statement is: "At least one adjacent activation node is active". This means that as soon as we have an active adjacent node, this statement should be true. A natural choice for this is to have a disjunction over all adjacent nodes, together with the activation formula from Equation (4.5). We now have the following:

$$\bigvee_{w \in \rho^+(v)} (\widehat{w} \wedge \neg \widetilde{w}).$$

The last of the three statements is: "All active nodes should be either inactive or ambiguous", or as we derived earlier: "All active nodes should be non-active", which is more of a mouthful in written form, but simpler with logic. The formula follows the same reasoning as the two earlier cases:

$$\bigwedge_{w \in \rho^+} (\neg w).$$

We now have formulas for the BN that are equivalent to the statements in Equation (4.4). The only matter that is left is to combine them in such a way, so that they correspond to Table 3.2. Between each of the columns is a conjunction and between the two rows there is a disjunction. The rule for inactivity, which is the first two rows of the table, can be stated as a Boolean function that we assign to the "inactive" node $\tilde{v}$:

$$\begin{aligned} \tilde{v}(t) = &\left(\bigwedge_{w \in \rho^+(v)} (\neg \widehat{w} \wedge \widetilde{w}) \right) \wedge \left(\bigvee_{u \in \rho^-(v)} (\hat{u} \wedge \neg \bar{u}) \right) \vee \\ &\left(\bigwedge_{w \in \rho^+(v)} (\neg \widehat{w} \wedge \widetilde{w}) \right) \wedge \left(\bigvee_{u \in \rho^-(v)} \neg \hat{u} \right) \wedge (\neg \hat{v} \wedge \tilde{v}). \end{aligned} \tag{4.8}$$

We can see that Equation (4.8) encodes the rules for a node to become inactive. For completeness, we will also state the same rule for the active node $\hat{v}$.

$$\begin{aligned} \hat{v}(t) = &\left(\bigvee_{u \in \rho^+(v)} (\hat{u} \wedge \neg \tilde{u}) \right) \wedge \left(\bigwedge_{w \in \rho^-(v)} (\neg \widehat{w} \wedge \widetilde{w}) \right) \vee \\ &\left(\bigvee_{u \in \rho^+(v)} \neg \hat{u} \right) \wedge \left(\bigwedge_{w \in \rho^-(v)} (\widehat{w} \wedge \neg \widetilde{w}) \right) \wedge (\hat{v} \wedge \neg \tilde{v}). \end{aligned} \tag{4.9}$$

Using eqs. (4.8) and (4.9) we now have a function for each node. Using all this we can now describe the complete algorithm for doing the transformation to a BN, or Booleanization as we will call it. We describe it in the next section.

4.2.3 Algorithm

In this section, we present the complete algorithm for doing the Booleanization. We also discuss its complexity. See Algorithm 1 for a complete code listing in pseudocode.

Algorithm 1 Booleanization of an SRG

```
1: procedure BOOLEANIZE(G, G_B)   ▷ Construct a BN G_B from the SRG G
2:     V ← G(V)
3:     V_B ← G_B(V)
4:     for v ∈ V do
5:         V_B ← V_B ∪ (v̂, ṽ)
6:         v̂(t) ← Equation (4.9)
7:         ṽ(t) ← Equation (4.8)
8:     end for
9:     return G_B
10: end procedure
```

As can be seen from the code, the algorithm constructs a new BN called G_B, which has for every node $v \in G$ two new nodes in G_B. This implies that $|V_B| = 2|V|$, meaning that there is twice the number of nodes in G_B when compared to G. The edges are defined on line six and seven by eqs. (4.8) and (4.9). If we look closer on the formulas, we see that if $|\rho^+(v)| > 0$, then we need to add two edges to both nodes: $\hat{v}$ and $\tilde{v}$. The same applies when $|\rho^-(v)| > 0$. The final clause in each of the equations also adds a self-loop to each node and an edge from $\hat{v}$ to $\tilde{v}$ and vice versa. If we sum over the indegree of each node in G_B, then we will get the total number of edges. Denote the indegree of a node $u \in V_B$, by $\delta^+(u)$. This quantity can be calculated as:

$$\delta^+(u) = 2\left(|\rho^+(v)| + |\rho^-(v)|\right) + 2, \forall u \in V_B. \tag{4.10}$$

To get the total number of edges, we sum over the indegree of all the nodes.

$$|E_B| = \sum_{u \in V_B} \delta^+(u)$$

We can see that, in general, G_B will be a densely connected graph with a high indegree for each node. As can be seen from Equation (4.10), we see that $\delta^+(u) \geq 2$, since the node u will always have a self-loop and at least one more edge from its partner node. We will now discuss the complexity of Algorithm 1. We will prove that it runs in quadratic time with respect to the number of nodes in the SRG.

Proposition 4. Algorithm 1 has a worst-case running time of $\mathcal{O}(|V|^2)$.

Proof. We see that the running time is dominated by the loop starting on line four. Inside the loop, we construct each of the Boolean functions according to the given equations. To do this, we have to go through each node in $\rho^+(v)$ and $\rho^-(v)$. In the worst case, which is if the graph G is fully connected with respect to both type of edges, the size of these two sets will be $|V|$, including self-loops. What this means is that in the worst case we have to, for each node

$v \in V$, consider every other node in $V \setminus \{v\}$. The following calculation shows this formally.

$$\begin{aligned}\mathcal{O}(|V| \cdot (|\rho^+(v)| + |\rho^-(v)|) &= \mathcal{O}(|V| \cdot (|V| + |V|) \quad |\rho^+(v)| = |\rho^-(v)| = |V| \\ &= \mathcal{O}(2|V|^2) \\ &= \mathcal{O}(|V|^2)\end{aligned}$$

Thus, we have now shown that Algorithm 1 has a worst-case running time of $\mathcal{O}(|V|^2)$. □

This result shows that the transformation can be done in polynomial time and is thus efficient. Next, we show an example to illustrate the theory developed in this chapter thus far.

Example 4.2.1. *Consider the graph in Figure 3.1. If we give this graph to Algorithm 1, it returns the graph that can be seen in Figure 4.4. The functions constructed by the algorithm will define the edges of the BN. The following functions were constructed by the algorithm:*

$$\begin{aligned}
\widehat{x_1}(t) &= ((\neg\widehat{x_2} \wedge \widetilde{x_2}) \wedge (\neg\widehat{x_3} \wedge \widetilde{x_3})) \vee ((\widehat{x_2} \wedge \neg\widetilde{x_2}) \wedge (\widehat{x_3} \wedge \neg\widetilde{x_3}) \wedge (\widehat{x_1} \wedge \neg\widetilde{x_1})) \\
\widetilde{x_1}(t) &= ((\widehat{x_2} \wedge \neg\widetilde{x_2}) \vee (\widehat{x_3} \wedge \neg\widetilde{x_3})) \vee ((\neg\widehat{x_2}) \vee (\neg\widehat{x_3}) \wedge (\neg\widehat{x_1} \wedge \neg\widetilde{x_1})) \\
\widehat{x_2}(t) &= ((\widehat{x_1} \wedge \neg\widetilde{x_1}) \wedge (\neg\widehat{x_3} \wedge \widetilde{x_3})) \vee ((\neg\widehat{x_1}) \wedge (\widehat{x_3} \wedge \neg\widetilde{x_3}) \wedge (\widehat{x_2} \wedge \neg\widetilde{x_2})) \\
\widetilde{x_2}(t) &= ((\neg\widehat{x_1} \wedge \widetilde{x_1}) \wedge (\widehat{x_3} \wedge \neg\widetilde{x_3})) \vee ((\neg\widehat{x_1} \wedge \widetilde{x_1}) \wedge (\neg\widehat{x_3}) \wedge (\neg\widehat{x_2} \wedge \widetilde{x_2})) \\
\widehat{x_3}(t) &= ((\widehat{x_1} \wedge \neg\widetilde{x_1}) \vee (\widehat{x_2} \wedge \neg\widetilde{x_2})) \vee ((\neg\widehat{x_1} \vee \neg\widehat{x_2}) \wedge (\widehat{x_3} \wedge \neg\widetilde{x_3})) \\
\widetilde{x_3}(t) &= ((\neg\widehat{x_1} \wedge \widetilde{w}) \wedge (\neg\widehat{x_2} \wedge \widetilde{x_2})) \vee ((\neg\widehat{x_1} \wedge \widetilde{x_1}) \wedge (\neg\widehat{x_2} \wedge \widetilde{x_2}) \wedge (\neg\widehat{x_3} \wedge \widetilde{x_3}))
\end{aligned}$$

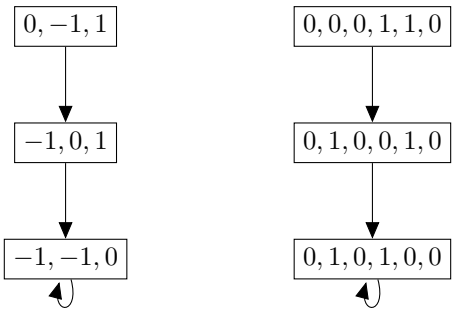
As we can see from the equation above, every node is connected to every other node.

As the last topic that we cover in this section, we will show an example that demonstrates how the equivalence between the attractors in the SRG model and the transformed BN model works in practice. Remember, that the equivalence is not only in the attractors, but also on the transitions between the states.

Example 4.2.2. *The SRG in Figure 3.1 has the attractor that can be seen in Figure 4.3 along with its basin. The BN constructed in Example 4.2.1, has the "same" attractor and basin, which can be seen on the right in Figure 4.3. The states from the BN's STG can be decoded by using the decoding function we defined earlier. The states would be decoded as follows:*

$$\begin{aligned}
\pi((0,0,0,1,1,0)) &= (\tau(0,0), \tau(0,1), \tau(1,0)) = (0,-1,1) \\
\pi((0,1,0,0,1,0)) &= (\tau(0,1), \tau(0,0), \tau(1,0)) = (-1,0,1) \\
\pi((0,1,0,1,0,0)) &= (\tau(0,1), \tau(0,1), \tau(0,0)) = (-1,-1,0)
\end{aligned}$$

We can see that the decoded attractor is the same as the one in the SRG.



(a) Attractor and its basin in the SRG.

(b) Attractor and its basin in the transformed BN.

Figure 4.3: Figure showing the two different paths in the STG from the SRG in Figure 3.1, and the BN created from the SRG in Example 4.2.1. On the left, is the SRG's STG, and on the right is the BN's STG.

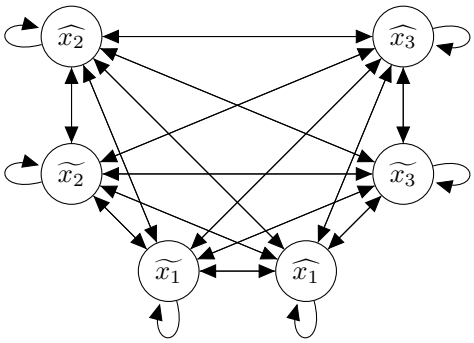


Figure 4.4: The new BN G_B constructed from the SRG G in Figure 3.1.

The transformation that we have presented in this section, while certainly useful, is of more theoretical interest than practical. The blowup of the number of edges in the transformed BN is likely an obstacle for many current algorithms that work with controllability problems on BNs. It can still be used for a modest number of nodes, and if any efficient algorithms for controllability problems for BNs are invented, they can be applied directly to our SRG model with the help of this Booleanization algorithm.

4.3 Conditions for existence of target attractors

The BN transformation from the previous section turned out to be a suboptimal solution for finding target attractors. In this section, we will go through a more efficient approach that is based entirely on the SRG model.

We will derive necessary and sufficient conditions for when a target attractor can exist, with specific target values, in the SRG. We will give a proof that states if the graph is built in a certain way, then there cannot exist a target attractor with some given target values $\boldsymbol{\alpha}$. We will show that if one such attractor

can exist, then there may be multiple such attractors. We will show that the conditions are necessary and sufficient for finding singleton target attractors. We also show that the conditions are necessary, but not in general sufficient for periodic target attractors to exist. When we say *target attractor* in this section, we mainly mean singleton target attractors, unless otherwise stated.

The proof will also give the necessary values for such an attractor, allowing us to construct one of them. We start by stating our goal for this section:

Theorem 1. UnanimousAttractorDetection $\in \mathcal{P}$.

We first consider a more constrained version of **UnanimousAttractorDetection**, where we only have one target node, and the target is that the node should be active in the attractor. We start with this concept and then move on to the inactive case. Finally, we show how to combine them and the end result is an algorithm that solves **UnanimousAttractorDetection**.

4.3.1 Attractor with active target value

As mentioned, we start with a more constrained version of our problem. Here, we only consider one node that should be active in the attractor. This problem can be formally formulated as:

Problem: UnanimousAttractorSingle

Input:	An SRG(V, E), a target value $\alpha = 1$, and a target node v.
Output:	"Yes" if there exists a target attractor A with $v = 1$. "No" otherwise.

We will show that this problem can be decided by inspecting the graph. We first note that, if a node is to remain active in a target attractor, then all nodes in $\rho^-(v)$ must be inactive, otherwise v would become inactive in the next state. This, in turn, gives a condition for the adjacent activation nodes for $\rho^-(v)$, they must also be inactive, and their activation nodes must also be inactive. We have now found a pattern, each node that we enumerate in this way must stay inactive. We can safely ignore all incoming activation edges, since they have no impact on whether v stays active or not.

We will now introduce some notation, which will help us formalize the idea above.

Definition 18. (Positive path) A positive path, is a directed path in the SRG that only follows edges labeled with a $+$.

Definition 19. (Negative path) A negative path, is a directed path in the SRG that only follows edges labeled with a $-$.

With these two definitions, we can now define the concept of *inactive ancestors*. Intuitively, an inactive ancestor is a node that must remain inactive for v to remain active. The set of inactive ancestors is then all nodes that must remain inactive for v to stay active. Formally, we can define the set with the help of positive paths and we will call the set: *Inactive ancestors for activity*.

Definition 20. (Inactive ancestors for activity) The set of inactive ancestors for activity for a node v is defined as:

$$\lambda^-(v) = \rho^-(v) \cup \{u : \text{there is a positive path from } u \text{ to } w \text{ where } w \in \rho^-(v)\}$$

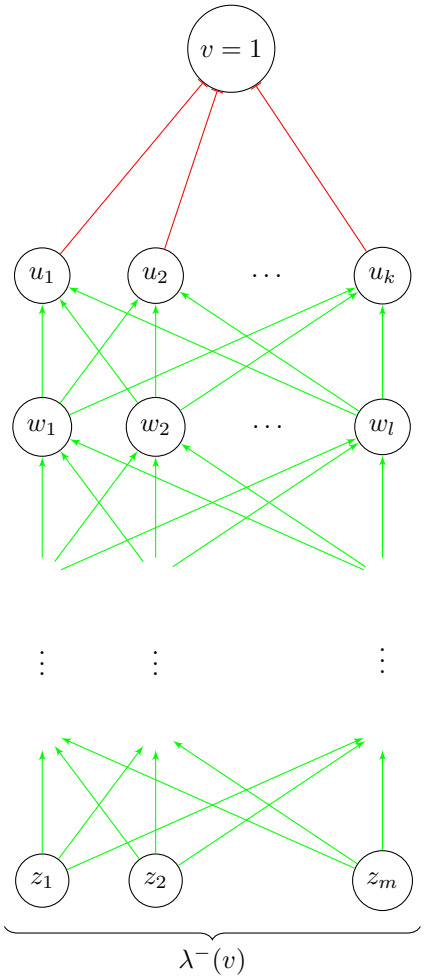


Figure 4.5: Illustration of a node with its inactive ancestors for activity, $\lambda^-(v)$. The first layer of the graph is just v's inhibitors. The second layer is all nodes that have a positive path of length one to any of the inhibitors. The k:th layer is all nodes that have a positive path of length k to any inhibition node of v. Here, we show the most general version of the figure, which is when every node is connected to every node that has a positive path of $k-1$. In general only one such path is needed.

The set of inactive ancestors can be seen graphically in Figure 4.5. In the figure, we have put the nodes in layers depending on the length of the positive path from each node. In general, there can be more than one such path and if that is the case, then we say that the positive path is simply the shortest path, taken from the set of all positive paths.

We can now state our first condition for v to remain active. In all the proposition and proofs we will assume that v is active, unless otherwise stated.

Proposition 5. Given an SRG(V, E) and a target value of $\alpha = 1$ for some node $v \in V$, all nodes belonging to $\lambda^-(v)$ must be inactive for there to be a target attractor with $v = 1$.

Proof. Assume that a node $u \in \lambda^-(v)$ is not inactive. If $u \in \rho^-(v)$, then v will become non-active in the next state and no target attractor with $v = 1$ can exist. If $u \notin \rho^-(v)$, then there must be a positive path from u leading to a node $w \in \rho^-(v)$. Assume the length of this path is k, then w will become either 0 or 1 in k timesteps and in turn make v non-active in the $k + 1$:th state. We have now shown that if any node in $\lambda^-(v)$ is active, then no target attractor with $v = 1$ can exist. □

We are now ready to consider the case when v has outgoing edges. We already know that the only nodes which are important for v to stay active is exactly the set $\lambda^-(v)$. Thus, only edges that go into that set will have any impact on v's dynamics. It can easily be seen that an inhibition edge into the set of inactive ancestors will make no difference since, by assumption, all those nodes must be inactive anyway. This leaves the activation edge. The relationship between the activation edge and the inactive ancestors is the following:

Proposition 6. If the target node v with $\alpha = 1$, has at least one activation edge to any node in $\lambda^-(v)$, then no target attractor is possible with $v = 1$.

Proof. Assume there is one such edge $e = (v, w)$, where $w \in \lambda^-(v)$, then in the next state w will become active. By Proposition 5 there can, therefore, exist no target attractor with $v = 1$, since not all nodes in $\lambda^-(v)$ are inactive. This completes the proof. □

What we have now proved is that for a target attractor to exist for a node v with a target value of $\alpha = 1$, all nodes in $\lambda^-(v)$ must be inactive and, furthermore, there can be no activation edge from v to any node in $\lambda^-(v)$. It turns out that this generalizes to k nodes. That is, if we have k nodes, and we ask: is there an attractor where all k nodes are active? It can be solved in the same way as for only one node, but with an extra special case. The problem we are now interested in solving is the following:

Problem: UnanimousAttractorMultiple

Input:	An SRG(V, E), a set of target nodes V and target values $\boldsymbol{\alpha} = \{1\}^{\lvert V \rvert}$.
Output:	"Yes" if there exists a target attractor A with $v = 1$, $\forall v \in V$. "No" otherwise.

Proposition 6 gives the condition for one node. We can extend it to k nodes, by realizing that as soon as any node v has an activation edge into any other

node's $u \neq v$ inactive ancestors, then there cannot exist any target attractors. The reason is that this edge would violate Proposition 6, since v would be active, and not inactive as required by the proposition. We can state this fact in many equivalent ways but one is the following:

Proposition 7. For there to exist a target attractor A, for a set of nodes V with $\boldsymbol{\alpha} = \{1\}^{|V|}$ no node can belong to any of the other node's inactive ancestors.

Proof. Assume that some node $v \in V$ belongs to some other node u's inactive ancestors. By Proposition 5, no target attractor with the requested target values can then exist. This completes the proof. □

We have now covered the case for how to find a target attractor, when the given target values are that all target nodes should be active. We now move on to the case when the target is instead that the node should be inactive.

4.3.2 Attractor with inactive target value

We begin this section by considering, as before, the easier case when the target set V only consists of one node. The difference now is that we want the node to be inactive in the attractor, that is $\alpha = -1$. We define a similar problem as we defined earlier when we considered active target values:

Problem: UnanimousAttractorSingleInactive

Input:	An SRG(V, E), a target value $\alpha = -1$, and a target node v.
Output:	"Yes" if there exists a target attractor A with $v = -1$. "No" otherwise.

The thing to note with inactivity in the SRG model is that as soon as a node becomes inactive, its edges no longer have any impact on the nodes it is connected to. Thus, we can ignore the outgoing edges from our target node.

We start by defining the inactive ancestors for inactivity. An inactive node can only become active if some other node activates it. For this reason, the set of inactive ancestors for inactivity is the set of nodes that have a positive path to the target node. The inhibiting nodes will have no impact on the target node. Formally, we define the set of *inactive ancestors for inactivity* as follows:

Definition 21. (Inactive ancestors for inactivity) The set of inactive ancestor for inactivity is defined as:

$$\lambda^{+}(v) = \{u : \text{such that there is a positive path from } u \text{ to } v\}$$

Depending on the target value for a node, we now have two different sets of inactive ancestors. For this reason, we introduce the notation: $\lambda(v)$, to mean the inactive ancestors of v. Which set this refers to should be clear from the context. Formally, we define $\lambda(v)$ as:

Definition 22. (Inactive ancestors) The set of inactive ancestors, for v, given some target value $\alpha \in \{1, -1\}$, is denoted by $\lambda(v)$ and is defined as:

$$\lambda(v) = \begin{cases} \lambda^{-}(v) & \text{if } \alpha = 1 \\ \lambda^{+}(v) & \text{if } \alpha = -1. \end{cases}$$

In the rest of this section, we will write $\lambda(v)$ to mean the set of inactive ancestors.

The following fact follows immediately by inspecting the graph and due to similar reasoning as for node activity in the previous section:

Proposition 8. For a target attractor A to exist with a target value $\alpha = -1$, for some given node v, it is enough that all nodes in the set $\lambda(v)$ are inactive.

In fact, such an attractor will always exist. Consider the configuration $\boldsymbol{v} = \{-1\}^{|V|}$, that is, setting every node in the network to be inactive. By definition of our update function, all nodes will remain inactive if they do not have a clear activation signal. Proposition 8 actually gives something more powerful than simply stating that all nodes must be inactive. It says that it is enough if only the nodes belonging to the inactive ancestors are inactive. This allows for more freedom, since we are still free to choose the values for the remaining nodes. We will now prove this.

Proof. Consider a node u that is not in the set of inactive ancestors for v. This node will either only have an inhibition edge to v, or it will reside in some other strongly connected component. Either way, it will not have any effect on v once v has become inactive. For this reason, we can ignore all such nodes u.

Consider now instead a node w, which is in the set of inactive ancestors. Assume further that this node is not inactive. This non-inactive state will propagate through the positive path and activate v in k timesteps, where k is the length of the shortest positive path from w to v. Thus, we see that if any such node w, is non-inactive, then v will become active. For this reason, all such nodes w must be inactive. □

We now move on to the case where we have multiple target nodes that we want to have as inactive. In fact, we can very easily extend Proposition 8 to multiple nodes in the following way: we simply require that, for each node v in the target set, all nodes belonging to each node's inactive ancestors must be inactive. We do not have to consider outgoing edges, since all the nodes are inactive and the outgoing edges are now, in some sense, unused. We have now proved the following fact:

Proposition 9. For a target attractor A, given a target set of nodes V, with target values $\boldsymbol{a} = \{-1\}^{|V|}$, it is enough to set all nodes in $\bigcup_{v \in V} \lambda^{+}(v)$ to inactive.

Now all that is left is to combine both of the cases for activity and inactivity and show that we can determine if a target attractor exists when we are given a combination of target values.

4.3.3 Combining the cases

In this section, we want to combine the theory we developed in the previous two sections, in order to find conditions that can solve **UnanimousAttractorDetection**. When combining the cases, we will be back to the original problem that we started with, namely **UnanimousAttractorDetection**.

It turns out there are some additional cases we need to consider when we have a combination of target values. The first observation is that Proposition 7 must hold for all nodes that have a target value of $\alpha = 1$.

The second observation is that no node with a target value of $\alpha = 1$ can be in the inactive ancestors for a node that has an inactive target value. The reason is that this would contradict Proposition 9, since it requires that all nodes are inactive. When we are looking for a target attractor that has a combination of active and inactive values, we will consider this as an extra case. The following theorem combines all that we have discussed thus far.

Theorem 2. For a target attractor to exist, where the target nodes V have the following target values $\boldsymbol{\alpha} = \{1, -1\}^{|V|}$, the following two conditions must hold:

1. For all nodes $v \in V$ with a target value of $\alpha = 1$, Proposition 6 and Proposition 7 must hold.
2. For all nodes $v \in V$ with a target value of $\alpha = -1$. Proposition 9 must hold and no node u with an active target value can be in the inactive ancestors of v.

From this theorem, we can see that this also gives a condition for the existence of target periodic attractors. The reason is, if there cannot be a singleton target attractor with the given target values, then there certainly cannot be a periodic attractor that has those values. Thus, we arrive at the following corollary:

Corollary 1. For a periodic target attractor to exist, where the target nodes V have the following target values $\boldsymbol{\alpha} = \{1, -1\}^{|V|}$, the conditions in Theorem 2 must hold.

In fact, we have now shown that the original problem we set out to solve can be solved by applying the conditions stated in Theorem 2. What we have still not yet shown, is that the solution is efficient. We will now give algorithms that put the theory into practice, and then give a short analysis stating that the algorithms run in polynomial time.

4.3.4 Algorithms

The most important set that we should be able to compute is the set of inactive ancestors, that is $\lambda^+(v)$ and $\lambda^-(v)$. One way to do it, is by starting from our target node v, then compute in a breadth-first search (BFS) [36] like manner, the inactive ancestors. Here, we consider the incoming edges, instead of outgoing edges. We have a set U, where we keep track of all the nodes we have seen so far. We also have a queue, where we put new unexplored nodes. When we encounter a node that we have not seen, we put it into U, as well as the queue. We continue exploring with nodes from the queue until it is empty. Algorithm 2, formalizes this idea.

Algorithm 2 is a special case of BFS, thus we can immediately conclude that the worst case running time is $\mathcal{O}(|V| + |E|)$, where V is the number of nodes in the network, and E is the number of edges. We have now established that:

Lemma 2. The set of inactive ancestors λ^+ and λ^- can be constructed in $\mathcal{O}(|V| + |E|)$ time.

Algorithm 2 Algorithm for constructing the set of inactive ancestors. The parameter v is the target node and α is the target value.

```
1: procedure InactiveAncestors(v, α)
2:     Let S be a queue
3:     if α = 1 then
4:         U ← ρ⁻(v)
5:     else
6:         U ← ∅
7:     end if
8:     S.enqueue(v)
9:     while S is not empty do
10:        u ← S.dequeue()
11:        for w ∈ ρ⁺(u) do
12:            if w ∉ U then
13:                U ← U ∪ {w}
14:                S.enqueue(w)
15:            end if
16:        end for
17:    end while
18:    return U
19: end procedure
```

To conclude, we will show an algorithm for solving **UnanimousAttractorDetection**. Algorithm 3 makes use of Theorem 2 to decide whether the requested target attractor can exist. The algorithm returns either $\mathcal{T}$ or $\mathcal{F}$, representing the truth values true and false respectively. True indicates that a target attractor exists, while false indicates that no such target attractor exists at all. We can see that each condition that we derived for target attractors is checked in the algorithm.

The running time of the algorithm is dominated by the time it takes to construct the set of inactive ancestors. The loop starting on line four, can in the worst case run in $\mathcal{O}(|T|(|V| + |E|))$ time. This is because the time it takes to construct the inactive ancestors is $\mathcal{O}(|V| + |E|)$, in the worst case. The other loops only work with smaller subsets of V and T, so for that reason they do not have any impact on the worst-case running time. We have now shown the following result:

Proposition 10. Algorithm 3 has a worst case running time of $\mathcal{O}(|T|(|V|+|E|))$ and solves **UnanimousAttractorDetection** in polynomial time.

Using Proposition 10, we can prove Theorem 1, which we set as our goal for this section.

Theorem 1. UnanimousAttractorDetection $\in \mathcal{P}$.

Proof. Proposition 10 shows that Algorithm 3 runs in polynomial time and solves **UnanimousAttractorDetection**. □

It is important to note that **UnanimousAttractorDetection** can be solved in polynomial time, only because the dynamics of the SRG model have certain

Algorithm 3 Algorithm that solves the **UnanimousAttractorDetection** problem. The parameters are the target set of nodes V and the target values $\boldsymbol{\alpha}$.

```
1: procedure UnanimousAttractor(T, α)
2:     T' ← {u_i : α_i = 1}
3:     T'' ← {u_i : α_i = −1}
4:     for 0 ≤ i ≤ |T| do
5:         λ(v_i) ← InactiveAncestors(v_i, α_i)
6:     end for
7:     for v ∈ T' do                                          ▷ Proposition 6
8:         if There is an activation edge (v, u), where u ∈ λ(v) then
9:             return F
10:        end if
11:    end for
12:    for v ∈ T' do                                          ▷ Proposition 7
13:        for u ∈ T' \ {v} do
14:            if v ∈ λ(u) then
15:                return F
16:            end if
17:        end for
18:    end for
19:    for v ∈ T'' do                  ▷ Inactivity requirement from Theorem 2
20:        for u ∈ T' do
21:            if u ∈ λ(v) then
22:                return F
23:            end if
24:        end for
25:    end for
26:    return T
27: end procedure
```

properties. If we would try and solve **UnanimousAttractorDetection** in a BN context, then the result would most likely not hold. This is because, in a BN, *any* Boolean function can be assigned to a given node, so it is not possible to reason about the network structure and inactive ancestors, in the way we have done here for the SRG model.

We now present an example illustrating the theory we have developed in this chapter.

Example 4.3.1. *Consider the SRG in Figure 3.5. Assume we are interested in finding out if there exists an attractor where the nodes* $V = \{E2F, Cdk2\}$ *have the following target values* $\boldsymbol{\alpha} = \{1, -1\}$*. We first compute the set of inactive ancestor for each of the nodes. The result is as follows:*

$$\lambda^{-}(E2F) = \{Myc, E2F, Cdc25A, Cdk2\}$$
$$\lambda^{+}(Cdk2) = \{E2F, Myc, Cdc25A\}$$

Now we are ready to use Algorithm 3 to check all the conditions. The output is $\mathcal{F}$*, since the node* $E2F$ *has an edge into its set of inactive ancestors. In fact, there are multiple conditions that are not fulfilled for this target attractor. One is that* $E2F$ *is in the set of inactive ancestors for* $Cdk2$*, which is not allowed.*

In this section, we have derived an algorithm for solving **UnanimousAttractorDetection** in polynomial time. It is important to note that the algorithm does not give as an output any attractor, it only decides if one can exist. In fact, there can be multiple such target attractors. In some sense, we do not really care about which of those target attractors we have found, as long as the target values hold. Thus, we would be interested in constructing at least one of the attractors. This is what we focus on in the next section.

4.3.5 Constructing a target attractor

In this section, we build on the theory of the previous section and show an algorithm that constructs one of the target attractors, given that one exists. The idea behind the algorithm is that the conditions for the target attractor give some nodes their necessary values. This makes generating one of the attractors easier, since we already have values for a subset of the nodes. The target nodes need to be given their target values. Apart from the target nodes, and their inactive ancestors, there will be a set of nodes that have no impact on the dynamics of the target nodes. One possible solution is to simply initialize those nodes to some random values, then we simply simulate starting from that configuration until we reach an attractor. The set of nodes that have no impact on the target nodes, called *non-impacting nodes* is defined as:

Definition 23. (Non-impacting nodes) The set of *non-impacting nodes*, denoted by $\zeta(T)$, are nodes that have no impact on the target nodes, meaning that there is no restriction on their values. They are all the nodes that are neither target nodes nor inactive ancestors. With this reasoning, we can construct the set of non-impacting nodes as follows:

$$\zeta(T) = V \setminus \left(T \cup \left[\bigcup_{u \in T} (\lambda(u)) \right] \right)$$

We are now ready to present the algorithm for constructing a target attractor.

Algorithm 4 Algorithm that constructs a target attractor. Assume the sets of inactive ancestors $\lambda(v)$ have already been constructed. The parameters are the SRG $G(V, F)$, the target nodes T, and the target vector $\boldsymbol{\alpha}$.

```
1: procedure ConstructUnanimousAttractor(G(V,F), T, α)
2:     t ← 0
3:     for v ∈ |T| do v(t) ← α_i
4:         for u ∈ λ(v) do
5:             u(t) ← −1
6:         end for
7:     end for
8:     U ← ζ(T)                          ▷ Nodes not impacting nodes in T.
9:     for w ∈ U do
10:        w(t) ← Random value from {1, −1, 0}
11:    end for
12:    G_STG(V′, E′) = ∅
13:    while G_STG is acyclic do
14:        x(t + 1) ← F(x(t))
15:        t ← t + 1
16:        E′ ← E′ ∪ (x(t), x(t + 1))
17:    end while
18:    return G_STG
19: end procedure
```

Algorithm 4 constructs one target attractor, which attractor is actually constructed is due to random chance. The running time is dominated by the time it takes to construct the STG. This running time will depend on the time it takes before the simulation reaches the attractor. This time is called the *transient period* of the system. At least for small networks, this period seems to be relatively short.

We finish this section, with an example of how to use Algorithm 4 to construct a target attractor.

Example 4.3.2. *Suppose that we are interested in finding a target attractor for the SRG in Figure 3.5, where our target set is* $T = \{pRb, p27\}$*, and the target values are:* $\boldsymbol{\alpha} = \{1, 1\}$*. We first need the set of inactive ancestors for both of the nodes:*

$$\lambda(pRb) = \{Cdk2, E2F, Myc, Cdc25A, Cdk4\}$$
$$\lambda(p27) = \{Cdk2, E2F, Myc, Cdc25A\}.$$

From this, we find the required values for all nodes but one. The nodes in $\lambda(v) \cup \lambda(v)$ *must all be inactive, while miR-17-92 can have any value. Assume we let it be active.*

Now we have a starting configuration $\boldsymbol{x_0} = (1, -1, 1, -1, 1, -1, -1, -1)$*. This configuration should be read in clockwise order from Figure 3.5, starting from p27. After simulating from the state* $\boldsymbol{x_0}$*, we have the following STG:*

$$(1,-1,1,-1,1,-1,-1,-1) \rightarrow (1,-1,-1,-1,1,-1,-1,-1) \circlearrowleft$$

The arrow at the ends means that we stay in the same state. Now we have found one target attractor that we were after. We can note here that letting miR-17-92 have a different initial value, would have resulted in the same attractor.

4.4 Controllability to target attractors

In this section, we will investigate a few controllability problems, in the context of the SRG model. We reason about the problems in the context of *strong controllability*. When we mention strong controllability we are proposing a special kind of controllability. The word *strong* comes from the fact that we want the values of the nodes in the SRG to be unanimous, meaning as little ambiguity among the nodes as possible, which was the original idea behind the definitions of our model. We state two problems, which we think are of importance, but leave their solutions as open research problems. We give some possible solutions, but do not claim that they are optimal. The first problem we want to consider is the following:

Problem: AnyStateAttractorControl

Input:	An SRG(V, E) and a target attractor A.
Output:	A minimal set of driver nodes that allows the network to be controlled into the attractor from any state $\boldsymbol{x}$ in one control.

The target attractor should be one that has the properties we presented in Section 4.1. Control is applied only at $t = 1$, this is perhaps the most reasonable assumption, since control at more time instances is biologically difficult to achieve [17]. Formally, we define the concept of control as:

Definition 24. (Control) Given a set of driver nodes $d_i \in D$ and an assignment $\mu : D \rightarrow \{1, -1\}$, a *control* of a state $\boldsymbol{x} = (x_1, \cdots, x_k)$ is defined as:

$$\mu \circ \boldsymbol{x} = (u_1, u_2, \cdots, u_k), \tag{4.11}$$

where u_i is the application of the control values, namely:

$$u_i = \begin{cases} x_i & \text{if } d_i \notin D \\ \mu_i & \text{if } d_i \in D. \end{cases} \tag{4.12}$$

Here, each μ_i can be any of the values $\{1, -1\}$ that can achieve the required control.

We will now discuss a possible solution to **AnyStateAttractorControl**. This problem can at least be partially solved, by using a concept from graph theory called the feedback vertex set (FVS). The FVS is defined as a set of nodes, whose removal from a graph, makes the graph acyclic. The problem of finding such a set can be stated as follows:

Problem: FVS-Problem

Input:	A directed graph $G(V, E)$.
Output:	A set of nodes $V' \subseteq V$, such that removal of the nodes V' from G makes G acyclic.

Acyclic means that the graph contains no cycle. In [29], they showed that the FVS can be used as driver nodes in a BN if the target states are restricted to attractors. In general, we are interested in finding the minimal number of driver nodes. For this reason, we modify the definition of the problem above as:

Problem: Minimal FVS

Input:	A directed graph $G(V, E)$.
Output:	A minimal set of nodes $V' \subseteq V$, such that removal of the nodes V' from G makes G acyclic.

The inclusion of minimality is important since otherwise, we could choose $V' = V$, which would work, but hardly be of any use. It is known that the problem of finding a minimal FVS is $\mathcal{NP}$-complete [22]. We now show an example of how we can use the FVS to locate a driver set that solves **AnyStateAttractorControl**. It is currently unknown if this approach works in general on any SRG.

Example 4.4.1. *Consider the SRG in Figure 4.1. Assume that our target attractor is the attractor we used in Example 4.1.1. A minimal FVS for the SRG is* $\{x_7, x_6, x_5, x_3\}$. *The following control will guide the network from any state into the attractor:*

$$\mu(x_7) = -1, \mu(x_6) = 1, \mu(x_5) = 1, \mu(x_3) = 1 \tag{4.13}$$

To see that this works, consider what other values the non-driver nodes can have. The node x_1 *will become active no matter what, since* x_5 *is activating it. Node* x_2 *will become inactive no matter what, since* x_1 *will be inhibiting it in the next state. The last node,* x_4*, will also become inactive, because of* x_1*. As we can see, the network is now in a configuration that belongs to the attractor, no matter what the initial state of the non-driver nodes is.*

The other problem we want to introduce is called **AnystateConstantAttractorControl**. The problem is almost the same: we have some target attractor, and want to find a control policy, which allows the SRG to be controlled from any other state into the attractor. What is different in this problem is that now control is not applied at $t = 1$, but instead the nodes are kept constant at some given value. What we would prefer to happen is that all attractors "collapse" into our target attractor A. We formally define the problem as:

Problem: AnyStateConstantAttractorControl

Input:	An SRG(V, E) and a target attractor A.
Output:	A minimal set of driver nodes that allows the network to be controlled into the attractor from any state $\boldsymbol{v}$, by keeping the driver nodes constant.

Now that the nodes are kept constant, some states will no longer be reachable. They will collapse into other states. Our target attractor will most likely also be different, however, the important issue to us will be that the target nodes in the target attractor have the correct values. This variant of controllability, has to the author's best knowledge, not been studied in any other kind of modeling framework before. We leave this as an open research problem.

Chapter 5

Description of the Simulation Software

To be able to study the dynamics of our model and see long-term behavior in non-trivial networks software assistance is required. In this chapter, we give a description of the requirements for the software, an overview of its architecture, and a breakdown of the different features available. The software is available to download from `https://bitbucket.org/patric_gustafsson/network-sim/src/master/`, and can be run on Windows, Mac and Linux-based systems. The software does not require any additional libraries or packages in order to be run. We use the cancer model that we presented in Section 3.4.3 as a running example to show the features of the software.

5.1 Requirements

The requirements of the software evolved from the problems we wanted to study. Our models become computationally difficult to compute by hand even when the number of nodes n is very small. Already $n = 4$ has $3^4 = 81$ states. Thus, we can see that already for small models it becomes a necessity to have access to software that can handle the simulation automatically.

The software was developed using methods similar to Agile methods and has for that reason no set requirements at the start. Rather, we developed iteratively different prototypes to see what would best fit our needs at the time. This is the essence of Agile, to be able to cope with changing requirements quickly and develop rapid prototypes in order to always have a working version of the software [28]. Some requirements were known from the start though:

1. *Simulation.* The software should possess the capability to run simulations on a predefined network. The results, which is the whole STG, should be available as a file in an appropriate format. The file should be compatible with other graph visualization software so as to ease the analysis of the STG. Since the STG can be very large, it needs to be opened in tools that have the capability to handle large graphs, such as Cytoscape [35].

2. *Attractors.* The software should be able to compute the attractors using

whatever algorithms we create, or simply with brute force. They should then be presented inside the software to the user.

3. *Controllability.* The user should be able to study controllability of a given network inside the software. From an initial state, the user should be able to see the dynamics evolve and be able to specify the state of some nodes or make them constant.

These requirements have all been implemented, and in the next section, we give an overview of the software's architecture and see how the requirements have shaped some of the design choices.

5.2 Architecture of the software

The software follows the Model-View-Controller (MVC) pattern which is a common pattern for developing desktop GUI applications and was first introduced in the 1980s in [25]. The idea with MVC is to separate the logic, data, and the viewing of the software data into different classes or structures so that they are independent from each other. Here is a short breakdown of the main responsibilities for each component in MVC:

Model The model's purpose is to hold the data that the application needs that is domain-specific. A good example from [25] is that the model of a text editor would be a string. For our software it would be the graph that is loaded in by the user. The model is also responsible for doing all the computations.

View The view is what the user sees when interacting with the program. The view should issue calls to the controller whenever some features that need the model are used. In our software, the view would be the GUI that is presented to the user when the program is started. Whenever the view is updated, it has to request the most recent data from the model.

Controller The controller is responsible for communication between the model and the view. In our software, we do not explicitly use a controller class and communication is mainly done between the model and the view. Since Java automatically handles button events for us, there is no need for a separate controller for doing that functionality. The controller is in some form still there in our software only that is handled by existing Java classes.

In Figure 5.1, the structure of our application is shown. The *MainApplication* class is the view of the application. It contains functions for updating the GUI and makes heavy use of the *NetworkSim* class which actually does all the computations. The *NetworkSim* class is in this sense our model, since it takes care of computations and keeps our model, the graph, as a class member. Whenever the view wants to update it has to request the graph from the *NetworkSim*. We then also have two additional helper classes that are used. One is the *DynamicsAnimator* which takes care of animating the updates done to the network when using the controllability feature of the software, which we will explain later. For reading the actual data from disk, a class called *MVGParser* is used.

For the actual simulation we use a library called BioLQM developed by the Colomoto (Consortium for Logical Models and Tools) Consortium [31]. It is a Java library for manipulation of logical models and includes features for doing various simulations. In our software, it is used by the class *NetworkSim* to simulate our model. BioLQM was chosen since it has ready-made support for multi-valued models. The function *exportToMnet* creates a file which can be read and interpreted by BioLQM.

One modification had to be made to BioLQM before we could use it in our simulator. BioLQM has multi-value logic support, however, not in the manner that we needed it. BioLQM uses the multi-valued notion that we discussed in Section 3.4, where values during the simulation are only updated monotonically. This is not how the SRG model works, rather updates can happen from any value to any other value. After we modified BioLQM to use the type of update rules that the SRG model uses, it was successfully included into the software.

5.3 Features of the software

In this section, we give an overview of the features that are available in the simulator. We also provide screenshots of the main functionality, which can function as a short manual for using the software. Before any simulations can be carried out, a network must be loaded in. We have created a file format called MVG (Multi-valued Graph) for this purpose. The format is simple and has a restricted grammar, which eases the parsing of MVG files.

5.3.1 The MVG format

The format can be seen as a subset of the simple interaction format (SIF), which is used by Cytoscape. The grammar is simple and is easiest described by an example.

Example 5.3.1.

$$
\begin{aligned}
&A \ <node\ \ interaction> \ B \\
&B \ <node\ \ interaction> \ C \\
&C \ <node\ \ interaction> \ D\ E\ F \\
&E \ <node\ \ interaction> \ F
\end{aligned}
$$

On the left-hand side of the <node interaction>, should be the name of the first node in the interaction. The right-hand side should be the name of the node that is being interacted with. Node names are unique, so a node can only have one name. It is also possible to specify multiple interactions at once, as is done on line three. Interactions, where a node is interacting with itself, are also allowed, this will give self-loops.

As mentioned, the MVG format is a subset of SIF. With SIF, it is possible to specify different types of interactions, however, we are only interested in the network topology. That is why we define the grammar of a node's interactions as: <node interaction> ::= {" + "|" − "}. This statement should be read in extended Backus-Naur form (EBNF) [34]. Here, a "+" means that there is a regulatory activation between the nodes. That is, if A is active and there is no contradiction, then B should become active and likewise for "-". Next, we show an example of how to use the MVG format.

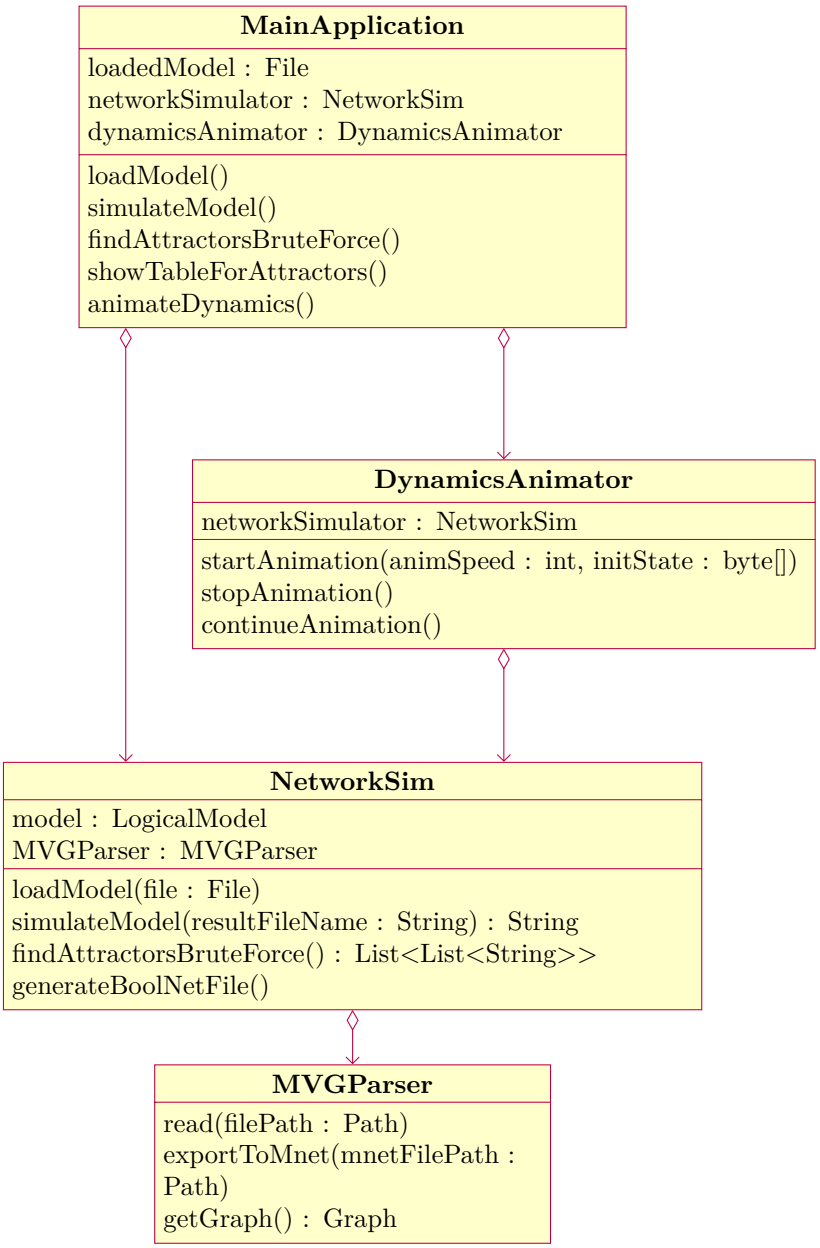


Figure 5.1: The overall class structure of our simulation software. Only the methods that are relevant to the discussion are included in the diagram. They represent the main functionality of the application and features that we discuss.

Example 5.3.2. *The cancer network from Figure 3.5 would be written in the following way in the MVG format:*

$$
\begin{aligned}
&Myc\ +\ Mir1792\\
&Myc\ +\ E2F\\
&Myc\ +\ Cdc25A\\
&Myc\ +\ Cdk2\\
&Myc\ -\ Myc\\
&Mir1792\ -\ E2F\\
&Mir1792\ -\ Mir1792\\
&E2F\ +\ E2F\\
&E2F\ +\ Myc\\
&E2F\ +\ Cdc25A\\
&E2F\ +\ Cdk2\\
&E2F\ +\ E2F\\
&E2F\ +\ Mir1792\\
&pRb\ -\ E2F\\
&Cdk2\ +\ Cdc25A\\
&Cdk2\ -\ p27\\
&Cdk2\ -\ pRb\\
&Cdc25A\ -\ Cdc25A\\
&Cdc25A\ +\ Cdk2\\
&Cdk4\ -\ Cdk4\\
&Cdk4\ -\ pRb\\
&p27\ -\ Cdk2
\end{aligned}
$$

5.3.2 Viewing a network

A network can be loaded in and viewed in the software by clicking on *File* and choosing *Load Network* from the menu. An MVG file should then be chosen, which should be written in the format described in the previous section. There is no upper limit on the number of nodes that the network can consist of. The software has been tested with a few hundred nodes. See Figure 5.2 for a screenshot of the software after we have loaded in the cancer network.

When the network has loaded, the nodes and the edges between the nodes should be visible. It is possible to move nodes around by clicking and dragging them. Edges can also be moved in the exact same way. A green edge means activation and a red edge means inhibition. If a new network is loaded, then the previous network will be deleted and the new one will be displayed instead. The software tries to layout all the nodes in a circle and this is the reason for the nodes being in a circle, whenever a network is loaded in.

Under the menu *Tools*, there are a few different tools that can be used with the network. In the following sections, we describe each of them.

5.3.3 Simulating a network

By clicking on *Tools* and then choosing *Simulate Network*, the network will be simulated. By simulate, we mean that the simulator generates each state that the network can be in and simulates to the next state, essentially constructing

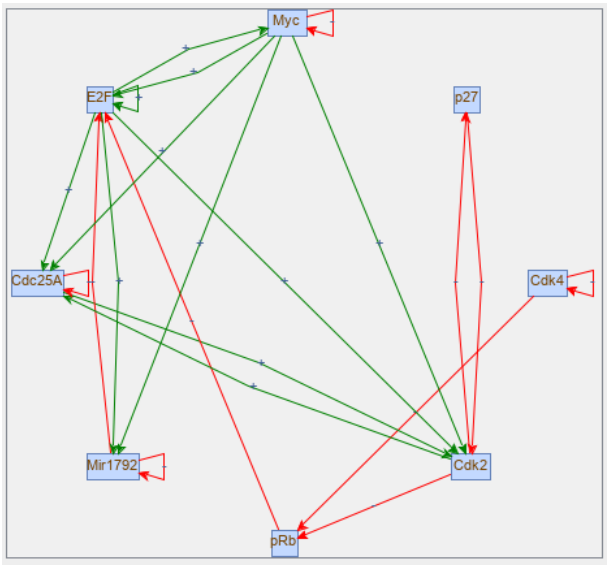
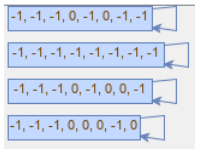


Figure 5.2: Screenshot of the software after having loaded in a model of the cancer network from Section 3.4.3.

the whole STG. After the simulation is done, the program will output a Comma-Separated Value (CSV) file, which contains two columns: one for the initial state and one for the successor state. The CSV file can be imported and viewed in other programs, such as Cytoscape, which is specifically made to handle large graphs. There is a limit for how large the network can be before the memory requirements become too large to handle. In testing, on a computer with 8GB of RAM, about $n = 25$ is the upper limit before the JVM and the computer run out of memory.

5.3.4 Computing attractors

The software also provides methods for computing attractors. Currently, only a brute-force method is available. It can be invoked by going into *Tools* and choosing *Compute Attractors*. The brute-force algorithm works by generating the whole STG, in which cycles are located. Every cycle in the STG is an attractor. This method suffers from the same limitations as the simulation feature, since it is doing the same computations and in addition also searches for attractors. After the computation is done, the attractors will be shown in the tool in a new window, see Figure 5.3. The software will also open a window where the attractors are ranked by the size of their basins. This is useful, since the STG quickly grows so large that it is infeasible to show it for larger networks. Thus, knowing the size of each basin of attraction can help by providing a bird's-eye view of the state space, without explicitly drawing it. An example of such a table can be seen in Figure 5.3.



(a) Top three attractors shown in the tool.

Basin Size	pRb	p27	Cdk4	Myc	Cdk2	E2F	Mir1792	Cdc25A
1152	0	-1	0	-1	-1	-1	-1	-1
736	0	0	-1	-1	1	-1	-1	-1
616	0	0	0	-1	-1	-1	-1	-1
576	-1	-1	-1	-1	-1	-1	-1	-1
576	-1	-1	0	-1	-1	-1	-1	-1
288	0	-1	-1	-1	-1	-1	-1	-1
272	0	0	-1	-1	-1	-1	-1	-1
84	0	0	0	0	1	0	0	-1
80	0	-1	0	0	-1	0	0	-1

(b) Tabular view of the attractors. Ranked by their basin size.

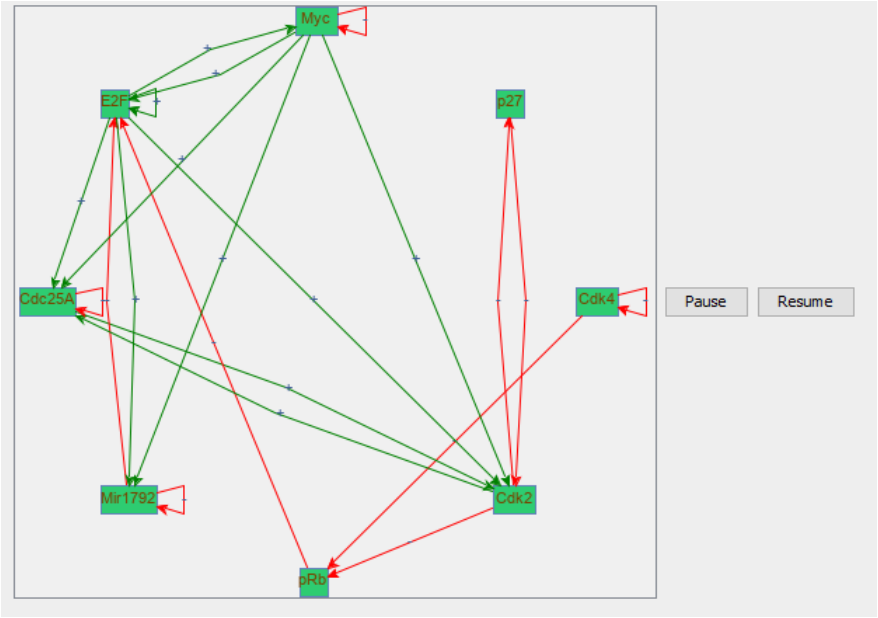
Figure 5.3: Two screenshots from the software. On the left we see the top four attractors. On the right we can see a tabular view of the attractors ranked by their basin size.

5.3.5 Booleanization of a network

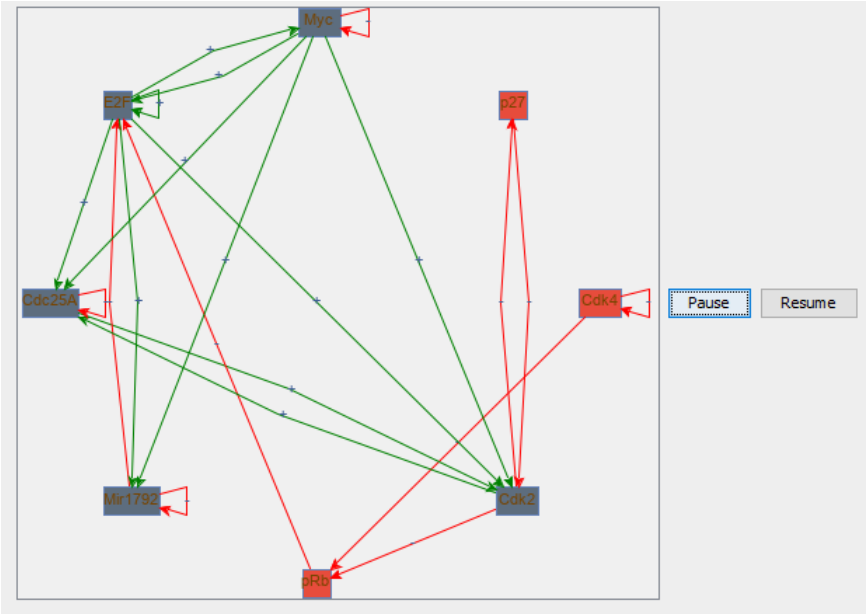
The algorithm presented in Section 4.2 is available in the software. It takes the network and generates an output file, which can then be imported into the R program BoolNet [30]. BoolNet is an R library that contains many tools and ready implemented algorithms for Boolean networks, so it would not make sense to reimplement them into the simulator. The Booleanization method can be run even on very large networks, since the Booleanization algorithm runs in quadratic time with respect to the number of nodes in the original SRG.

5.3.6 Controlling a network

Controllability can also be studied with the software. By selecting *Tools* and then *Simulate Dynamics*, a dialog opens where an initial state can be entered. An animation should now start on the network in the view. Nodes that are active will be displayed in green, inactive ones in red, and ambiguous ones will be in a dark gray color. The animation can be paused by clicking on the *Pause* button on the right-hand side in the GUI. It should now be possible to click on each node and change its state, thus controlling it and then resuming the simulation. See Figure 5.4 for a demonstration of this feature.



(a) The network shown in the controllability view. Here the initial state has been chosen as having all nodes active.



(b) The network shown after one timestep. At this point, it is possible to stop the simulation and set different values for some nodes if needed, and then resume the simulation.

Figure 5.4: The controllability view is shown here. In a) we see the network in the chosen initial state. In b) we see the network after one timestep from the initial state.

Chapter 6

Discussion

Introducing a new model for something is never a simple task. There are more angles for exploiting weaknesses in some new work than in comparison to tested and widely accepted models. One will have to convince not only oneself, but other people that one's model brings something novel both from a usability and theoretical standpoint. The opinion of the author is that this endeavor was successful in the context of this thesis, at least. When the project started we were even worried that we were dealing with the empty set or a model that would display completely degenerative behavior. One might ask why use these logical modeling frameworks when so many of the problems that we want to study in them are difficult, such as finding attractors?

The fact is that many of these logical models that are currently used in systems biology are not scalable to any real example. Already networks that have a few hundred nodes will make many state of the art algorithms unusable. One might wonder, if this an intrinsic limitation to all logical models used in systems biology? Perhaps the exponential growth of the state space is something that we will never be able to handle, since even with improved computer hardware the gains will only be marginal.

This is why it was surprising to see that in the SRG model we are able to decide if a target attractor exists, in polynomial time. The state space that the SRG model deals with is even larger than the Boolean one. For the target attractor we were helped by how the dynamics of our model was defined. This might be an indication that the controllability problems we introduced can also be solved efficiently. If that is the case, then the SRG model would have a very strong case against other kinds of models. Another benefit of our model if we compare it to others, is that it can be applied directly to any gene regulatory network, since our model only needs the topology of the network. This is not the case for most of the other models we have discussed in this thesis, only for majority voting, but there we also have to take into account the starting parameters and so on.

The author is of the opinion that the SRG model has properties that make it worthwhile to study it from a controllability perspective. The whole idea of the model was guided by the notion that we wanted *strong controllability* as a goal. If unanimity among nodes can be conserved over long paths, then this would hopefully present a stronger framework for controllability than the traditional models.

The two problems we presented in Chapter 4 should serve as a good guideline for future research of the SRG model. If it turns out that these problems can also be solved in polynomial time, as the target attractor problem could, then we might be on the verge of some major discovery. This would allow us to quickly generate control strategies for real biological networks consisting of hundreds of nodes and not only tens of nodes which today is considered state of the art.

Using software will be essential for any future research. For this purpose, the software that we developed in this thesis can be used. It can easily be extended and new features can be added to study controllability more in depth.

Svensk sammanfattning

Inom systembiologin studerar man biologiska strukturer med målet att få en förståelse för hur systemet fungerar som en helhet. Detta är i kontrast till den traditionella biologin där man ofta har studerat en viss del av ett system väldigt noggrant, i hoppet om att genom att förstå de små delarna så skulle en förståelse för systemet som en helhet uppenbara sig. Inom systembiolgin samarbetar biologer och datavetare för att lösa problem som direkt kan användas i verkligheten. Så kallade *regulatoriska gennätverk* är en vanlig from av nätverk som används. Ett sådant nätverk kan ses som en graf där varje gen är en nod. Varje nod reglerar andra noder genom att antingen aktivera eller inaktivera andra noder. En nod som reglerar en annan nod innebär att det uppkommer en båge mellan noderna i grafen. Systembiologin tillämpas t.ex. inom medicinen, där man b.la. kunnat förutspå vilka celler som är cancerbildande samt för utveckling av nya mediciner [14, 7].

Ett ofta använt sätt för att studera gennätverk är med hjälp av logiska modeller eller differentialekvationer. Differentialekvationer har nackdelen att de kräver väldigt specifik data för att få en användbar modell. I de flesta fall finns det inte tillräckligt med data för att göra en modell [39]. De logiska nätverken jobbar på en abstraktionsnivå högre än differentialekvationerna och tar endast i beaktande nätverkets (grafens) struktur. Den kändaste modellen för detta är så kallade Booleska nätverk som introducerades år 1969 av Kauffman [23]. I de Booleska nätverk kan en nod endast ha värdet sant eller falskt, oftast skrivet som 1 eller 0. Från ett biologiskt perspektiv skulle sant betyda att noden är aktiv medan falskt skulle betyda att den är inaktiv. Även om de Booleska nätverken kan ge intrycket av att de har en allt för enkel bild av verkligheten, så har de använts inom den medicinska industrin. Till exempel så har de använts för att förutspå vilka celler som utgör en risk för tumörutveckling [14].

Vi introducerar en ny modell som är en sorts logisk modell. Men istället för endast två värden, som de Booleska nätverken, använder vår modell tre. Två av värdena har samma betydelse som i de Booleska nätverken, d.v.s. aktivt eller inaktivt, medan det tredje värdet har en annan innebörd. Idén bakom det tredje värdet är följande: anta att en nod påverkas av två andra noder där den ena noden ger en signal för att aktivera medan den andra ger en signal för att inaktivera. Hur ska noden i detta fall veta om den ska vara inaktiv eller aktiv i följande tillstånd? I vår modell ser vi detta som en tvetydlig situation och skulle tilldela noden värdet 0. Vår modell har på engelska namnet *strong regulatory graphs* vilket på svenska kan översättas till ungefär starka regulatoriska grafer (SRG). Med starka menar vi de noder som är aktiva eller inaktiva verkligen ska ha ett av de tillstånden. För att om de inte skulle vara aktiva eller inaktiva skulle de vara i det tredje tillståndet. Detta är tanken bakom vår modell, minimera

tvetydighet.

Attraktorer och attraktionsbassänger

Varje nod i SRG modellen har en funktion som anger vilket värde noden får beroende på de regulerande nodernas tillstånd. I sammanfattningen har vi inte utrymme att beskriva hur tillstånden uppdateras i detalj men vad kan sägas är att varje tillstånd alltid leder till antingen samma tillstånd eller ett nytt tillstånd. Eftersom de nätverk vi behandlar är ändliga finns det endast ett ändligt antal tillstånd som nätverken kan befinna sig i. Det är möjligt att rita upp ett tillståndsdiagram där varje tillstånd finns med. Cykler i denna graf kallas för attraktorer. De kan ses som låsta tillstånd från vilket det inte är möjligt att komma ut. Längden på attraktoren kallas för attraktorens period.

De tillstånd som leder till attraktorerna kallas för attraktionsbassänger. Så fort ett nätverk har hamnat i en attraktionsbassäng för en viss attraktor kommer nätverket nödvändigtvis att gå mot attraktorn. Attraktorer är intressanta hur en biologisk synpunkt eftersom de ofta kan ses som olika tillstånd i vilken en cell kan vara i [19]. Ofta är vi intresserade av att *leda* nätverket från ett ohälsosamt tillstånd till ett hälsosammare tillstånd. Detta koncept är känt som *nätverksstyrning*, d.v.s. vi vill styra nätverket från ett tillstånd till ett annat. Detta kan ske på olika sätt, men ett sådant är att ta styra över ett visst antal noder genom att tilldela ett passande värde till dem så att nätverket styrs till det tillstånd man är ute efter.

I SRG modellen kan varje nod ha ett värde från $\{1, -1, 0\}$. Eftersom varje nod kan ha ett utav tre värden är det totala antalet olika tillstånd som nätverket kan vara i 3^n, där n är antalet noder.

SRG modellen och nätverksstyrning

Nätverksstyrning är ofta definierat som att givet ett tillstånd, styr nätverket till vilket annat tillstånd som helst. Detta är för vårt ändamål en för bred definition. Istället fokuserar vi på att styra nätverket till vissa attraktorer som har den egenskapen att en nods värde hålls konstant som 1 (aktivt) eller -1 (inaktivt) under hela attraktorn. Detta kallar vi för stark nätverksstyrning. Idén är att noderna ska vara överens om varför nätverket är i ett visst tillstånd. Att hitta attraktorer med den egenskapen kan ses som en svår uppgift eftersom vi har ett exponentiellt antal tillstånd att söka igenom. Som ett första försök tar vi hjälp utav de Booleska nätverken.

Vi introducerar en algoritm som förvandlar ett SRG nätverk till ett Booleskt nätverk. Nätverken är ekvivalenta med varandra med anseende på attraktorerna och tillstånden. Förvandlingen gör det möjligt för oss att ta i bruk de algoritmer och resultat som redan har visats för Booleska nätverk. En nackdel med förvandlingen är att den skapar mycket täta nätverk. Med täta menas att det finns många bågar mellan noderna. Detta är ett problem för en del algoritmer eftersom de oftast blir långsamma om graferna de jobbar med är täta. Denna förvandling ger en övre gräns för svårighetsgraden på vårt problem men är inte effektiv. Från teorin vet vi att nätverksstyrningsproblemet är ett $\mathcal{NP}$-komplett problem i Booleska nätverk, d.v.s. svårt att lösa. Eftersom vårt problem att hitta konstanta attraktorer är ett specialfall av nätverksstyrningsproblemet så är detta inte en effektiv lösning.

En effektivare lösning existerar. Vi visar att det är möjligt att avgöra om en konstant attraktor existerar i polynomiell tid, vilket anses vara en effektiv körtid för algoritmer. Genom att kolla om SRG-nätverket uppfyller vissa krav kan vi ge ett svar om en viss attraktor existerar eller inte, helt utan att ta i beaktande tillståndsdiagrammet. De krav som vi hittar ger även värden på en del noder i grafen vilket gör det möjligt för oss att inte bara visa att en konstant attraktor existerar utan även konstruera en sådan.

Till slut introducerar vi även två problem som har med nätverksstyrning att göra. I korthet vill vi styra nätverket från vilket tillstånd som helst till en utav de konstanta attraktorerna. Vi lämnar de som öppna problem för framtida forskning. I avhandlingen nämner vi ändå några möjliga lösningar men ger inga garantier på att de skulle vara de mest optimala som finns.

Programvara för simulering

Under projektet har vi även utvecklat ett program för att studera SRG-nätverk. Programmet kan köras på vilken som helst modern datorn och operativsystem. Programmet har funktioner för att visa, simulera, analysera, förvandla och styra nätverk. Programmet använder i bakgrunden programvaran BioLQM som är utvecklat av Colomoto consortium [31]. BioLQM är ett bibliotek av funktioner för att analysera och simulera olika modeller av logiska nätverk.

Att simulera nätverken har vissa begränsningar. Eftersom antalet tillstånd växer exponentiellt är antalet noder i ett nätverk i praktiken begränsat till omkring 25 stycken. De andra funktionerna som styrning eller förvandling till ett Booleskt nätverk påverkas inte av dessa begränsningar. Programvaran finns att ladda ner på följande sida: `https://bitbucket.org/patric_gustafsson/network-sim/src/master/`.

Avslutning

Att skapa en ny modell för något är inte ett enkelt uppdrag. Man måste kunna visa att den nya modellen är användbar inte bara sett ur ett teoretiskt perspektiv utan även praktiskt. Sedan ska modellen också skilja sig på ett märkbart sätt från de existerande modellerna. Enligt författarens åsikt har detta lyckats i detta avhandlingsprojekt. En tid fanns det en tanke att det som vi hade att göra med var endast tomma mängden eller att hela nätverket skulle ledas till endast en attraktor. Detta visade sig inte stämma utan istället har vi skapat en modell som kan tillämpas på verkliga data.

Att hitta attraktorer i polynomiell tid är en bedrift eftersom vi behandlar strukturerar som innehåller ett exponentiellt antal tillstånd. Inom andra logiska modeller skulle samma algoritm knappast fungera eftersom den är baserad på hur vi definierat vår modell. En annan fördel med SRG modellen är att den kan använda på vilka regulatoriska-gennätverk som helst.

Vi har även öppnat upp olika vägar för vidare forskning. Bland annat direkt genom att ge två öppna problem inom nätverksstyrning. All vidare forskning kommer att ha hjälp av mjukvaran som utvecklades för att studera SRG-nätverken. Nya upptäckter genom forskning kan enkelt studeras med hjälp av programmet och nya funktioner kan läggas till efter behov.

Bibliography

[1] Tatsuya Akutsu. *Algorithms for Analysis, Inference, and Control of Boolean Networks*. World Scientific, 2018.

[2] Tatsuya Akutsu, Morihiro Hayashida, Wai-Ki Ching, and Michael K Ng. Control of Boolean networks: Hardness results and algorithms for tree structured networks. *Journal of Theoretical Biology*, 244(4):670–679, 2007.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] Gilles Bernot, Jean-Paul Comet, Adrien Richard, and Janine Guespin. Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339–347, 2004.

[5] Cosetta Bertoli, Jan M Skotheim, and Robertus AM De Bruin. Control of cell cycle transcription during G1 and S phases. *Nature Reviews Molecular Cell Biology*, 14(8):518, 2013.

[6] Peter Bloomingdale, Van Anh Nguyen, Jin Niu, and Donald E. Mager. Boolean network modeling in systems pharmacology. *Journal of Pharmacokinetics and Pharmacodynamics*, 45(1):159–180, Feb 2018.

[7] Eugene C Butcher, Ellen L Berg, and Eric J Kunkel. Systems biology in drug discovery. *Nature Biotechnology*, 22(10):1253, 2004.

[8] Jean-Paul Comet, Hanna Klaudel, and Stéphane Liauzu. Modeling multivalued genetic regulatory networks using high-level Petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 208–227. Springer, 2005.

[9] Elena Dubrova and Maxim 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.

[10] Susan Elmore. Apoptosis: a review of programmed cell death. *Toxicologic Pathology*, 35(4):495–516, 2007.

[11] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335, 2000.

[12] Carlos Espinosa-Soto, Pablo Padilla-Longoria, and Elena R Alvarez-Buylla. A gene regulatory network model for cell-fate determination during arabidopsis thaliana flower development that is robust and recovers experimental gene expression profiles. *The Plant Cell*, 16(11):2923–2939, 2004.

[13] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009.

[14] Herman F Fumia and Marcelo L Martins. Boolean network model for cancer pathways: predicting carcinogenesis and targeted therapy outcomes. *PLOS ONE*, 8(7):e69008, 2013.

[15] Joseph A Gallian. A dynamic survey of graph labeling. *The Electronic Journal of Combinatorics*, 16(6):1–219, 2009.

[16] Alexander J Gates and Luis M Rocha. Control of complex networks requires both structure and dynamics. *Scientific reports*, 6:24456, 2016.

[17] Wenpin Hou, Peiying Ruan, Wai-Ki Ching, and Tatsuya Akutsu. On the number of driver nodes for controlling a boolean network when the targets are restricted to attractors. *Journal of Theoretical Biology*, 463:1–11, 2019.

[18] Wenpin Hou, Takeyuki Tamura, Wai-Ki Ching, and Tatsuya Akutsu. Finding and analyzing the minimum set of driver nodes in control of boolean networks. *Advances in Complex Systems*, 19(03):1650006, 2016.

[19] 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.

[20] Ivan Ivanov and Edward R Dougherty. Modeling genetic regulatory networks: continuous or discrete? *Journal of Biological Systems*, 14(02):219–229, 2006.

[21] Minoru Kanehisa and Susumu Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

[22] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

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

[24] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, April 1976.

[25] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[26] Nicolas Le Novere. Quantitative and logic modelling of molecular and gene networks. *Nature Reviews Genetics*, 16(3):146, 2015.

[27] H Lodish, A Berk, SL Zipursky, et al. *Molecular Cell Biology. 4th edition.* New York: W. H. Freeman, 4th edition, 2000.

[28] Agile Manifesto. Manifesto for agile software development. 2001.

[29] Atsushi Mochizuki, Bernold Fiedler, Gen Kurosawa, and Daisuke Saito. Dynamics and control at feedback vertex sets. ii: A faithful monitor to determine the diversity of molecular activities in regulatory networks. *Journal of Theoretical Biology*, 335:130–146, 2013.

[30] Christoph Müssel, Martin Hopfensitz, and Hans A Kestler. Boolnet—an r package for generation, reconstruction and analysis of boolean networks. *Bioinformatics*, 26(10):1378–1380, 2010.

[31] Aurélien Naldi. Biolqm: A Java toolkit for the Manipulation and Conversion of Logical Qualitative Models of Biological Networks. In *Frontiers in Physiology*, 2018.

[32] Aurélien Naldi, Denis Thieffry, and Claudine Chaouiya. Decision diagrams for the representation and analysis of logical models of genetic networks. In Muffy Calder and Stephen Gilmore, editors, *Computational Methods in Systems Biology*, pages 233–247, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[33] Dynamik Boolescher Netzwerke. Dynamics of Boolean networks. 2009.

[34] Richard E Pattis. Ebnf: A notation to describe syntax, 2013.

[35] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.

[36] Steven S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[37] René Thomas. Regulatory networks seen as asynchronous automata: a logical description. *Journal of Theoretical Biology*, 153(1):1–23, 1991.

[38] René Thomas, Denis Thieffry, and Marcelle Kaufman. Dynamical behaviour of biological regulatory networks—i. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of Mathematical Biology*, 57(2):247–276, 1995.

[39] Mikhail Tikhonov and William Bialek. Complexity in genetic networks: topology vs. strength of interactions. *arXiv preprint arXiv:1308.0317*, 2013.

[40] Rui-Sheng Wang, Assieh Saadatpour, and Reka Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Physical Biology*, 9(5):055001, 2012.

[41] Lijian Yang, Yan Meng, Chun Bao, Wangheng Liu, Chengzhang Ma, Anbang Li, Zhan Xuan, Ge Shan, and Ya Jia. Robustness and backbone motif of a cancer network regulated by mir-17-92 cluster during the G1/S transition. *PLOS ONE*, 8(3):e57009, 2013.

[42] Jorge GT Zanudo, Maximino Aldana, and Gustavo Martínez-Mekler. Boolean threshold networks: Virtues and limitations for biological modeling. In *Information Processing and Biological Systems*, pages 113–151. Springer, 2011.

[43] Jorge GT Zañudo, Steven N Steinway, and Réka Albert. Discrete dynamic network modeling of oncogenic signaling: Mechanistic insights for personalized treatment of cancer. *Current Opinion in Systems Biology*, 2018.