Phenotype Control of Partially Specified Boolean Networks[⋆]

Nikola Beneš[1], Luboš Brim[1], Samuel Pastva[2], David Šafránek[1], and Eva Šmijáková[1] ✉

[1] Systems Biology Laboratory, Masaryk University, Brno, Czech Republic
[2] Institute of Science and Technology Austria, Klosterneuburg, Austria
`xsmijak1@fi.muni.cz`

Abstract. Partially specified Boolean networks (PSBNs) represent a promising framework for the qualitative modelling of biological systems in which the logic of interactions is not completely known. Phenotype control aims to stabilise the network in states exhibiting specific traits. In this paper, we define the phenotype control problem in the context of asynchronous PSBNs and propose a novel semi-symbolic algorithm for solving this problem with permanent variable perturbations.

1 Introduction

Boolean networks (BNs) are a widely used model to study dynamics of complex biological systems [4,2]. Recently, a new interesting variant of BN control problem called *phenotype* control was proposed [16,18]. The goal of the phenotype control is to stabilize the network in states exhibiting specific traits regardless of the source state. This approach does not limit the control target to a single state or a specific attractor but rather considers arbitrary combinations of traits (subspaces of BN states). To control the network, we use *variable perturbations* that fix variable values to specific constants.

The behavior of a BN is given by the Boolean update functions and the considered updating scheme. In this work, we specifically focus on *asynchronous* updates, where one update rule is triggered at a time, because they suitably capture the behaviour of biological systems [3]. However, it is important to note that in many cases, the exact Boolean functions of the model may not be precisely known due to various uncertainties such as insufficient experimental knowledge [22], inconsistent observations [21], genetic mutations [30], or any other ambiguities. To address this issue, we employ a framework of *partially specified* Boolean networks (PSBN), in which update functions can be defined using uninterpreted function symbols [8,12,9].

The most well-studied variant of BN control problem is *source-target* control in which both source and target states are specified. Generalized variants of this

⋆ This work was supported by the Czech Foundation grant No. GA22-10845S, Grant Agency of Masaryk University grant No. MUNI/G/1771/2020, and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 101034413.

problem include target control [25], which aims to reach the target attractor regardless of the initial state, and full-control [20], which seeks a control strategy between all attractor pairs.

The control problems also differ significantly in the perturbations which they employ. The simplest perturbation variant is solved in the context of Boolean control networks (where only inputs are allowed to be set) [27]. In contrast, another line of research allows influence of any BN variable. In biology, such perturbations can be implemented in terms of gene knock-outs or over-expressions. Commonly used are *permanent* perturbations which consider variables being stabilized ad infinitum [40,34,33]. Other types of perturbations include one-step [5], temporary [28,36,38], and sequential dynamics [1,29,31,37]. It is worth noting that the source-target control was also studied in the context of PSBNs [11,12].

A common drawback of source-target, target and full control problems is that the target attractor must be fully known in advance in order to be provided as an input to the source-target control algorithm. However, the unknown parts might cause a PSBN to exhibit very similar attractors differing only in some negligible parts. Therefore, multiple attractors might be in fact a target of interest. A similar scenario can occur even for fully known networks, as the modeler's goal might be to stabilize just some subset of traits (phenotype) which are actually exhibited in several different attractors.

The phenotype control of BNs was previously solved using a method based on a reduction of the network into a layered network and converging trees [15]. However, the method is assuming synchronous update semantics. The phenotype control was later addressed also for asynchronous update semantics and solved using model checking methods [18]. A very similar problem, *marker control*, was also solved for the most permissive updating scheme [32].

Our contribution. In this work, we lift the problem of phenotype control from standard asynchronous BNs to partially specified BNs. We use permanent variable perturbations to achieve the control objective. We develop semi-symbolic method based on Binary Decision Diagrams (BDDs). This method allows us to find solutions for all PSBN interpretations in a single run, unlike brute-force methods, which need to compute solutions for each fully specified BN instance separately. Our method is based on a search of trap-sets, which makes its main idea transferable to other BN updating schemes as well. The method is conceptually related to [18], but aside from our novel incorporation of PSBNs, we also lift the perturbations into the symbolic domain (as opposed to the brute-force enumeration in [18]). We also demonstrate how this method can be applied to obtain interesting observations about PSBN models on real-world case studies.

2 Theory

This section presents a formal introduction to the topic of partially specified Boolean networks, including the notion of permanent perturbation, phenotype control and perturbation robustness.

Notation. Let us first note that we consider 0 and 1 as interchangeable with *false* and *true*, respectively. We then define $\mathbb{B} = \{0, 1\}$ to be the set of Boolean values and $\mathbb{B}_* = \{0, 1, *\}$ to be an extension of $\mathbb{B}$ which also admits a *free* value $*$ (i.e. neither *true* nor *false*). We write $\mathbb{B}^n$ to denote the set of all n-element vectors over $\mathbb{B}$. In the following, such n refers to the *size* of a Boolean network, while x represents its state (a configuration). For each $x \in \mathbb{B}^n$, x_i then denotes the i-th element of x. Finally, for $x \in \mathbb{B}^n$, index $i \in [1, n]$, and a Boolean value $b \in \mathbb{B}$, expression $x[i \mapsto b]$ denotes a substitution of the i-th element in x for the value b. Formally, the result is $x' = x[i \mapsto b]$ s.t. $x'_j = b$ for $j = i$, and $x'_j = x_j$ otherwise.

2.1 Boolean networks

Before we define partially specified Boolean networks, we first introduce the classical (i.e. fully specified) Boolean network (BN) and other related concepts:

Definition 1. *Let n be the number of system variables. A* Boolean network *is a collection $\mathbb{BN} = \{f_1, \ldots, f_n\}$ with each $f_i : \mathbb{B}^n \to \mathbb{B}$ being the Boolean* update function *(sometimes called* local function*) of the network's i-th variable.*

In the context of a specific Boolean network, the set $\mathbb{B}^n$ is the network's *state space*, and the vectors $x \in \mathbb{B}^n$ are its *states*. Note that although the input of each f_i is a full state $x \in \mathbb{B}^n$, the output of f_i does not typically depend on all network variables, but rather on a smaller subset of variables which we say *regulate* the i-th variable. If a variable has no regulators (i.e. its update function is a constant), we also call it an *input* of $\mathbb{BN}$. Symmetrically, variable that does not regulate any other variable is called an *output*.

Additionally, we call the members of $\mathbb{B}_*^n$ the *subspaces* of $\mathbb{BN}$. Intuitively, each subspace $S \in \mathbb{B}_*^n$ describes a hypercube in the state space $\mathbb{B}^n$. This hyper-cube consists of states $x \in \mathbb{B}^n$ such that $x_i = S_i$ for all i where $S_i \in \mathbb{B}$. We can thus treat each subspace S as a set of states. Furthermore, to denote a specific subspace, we will often simply use a string of values from $\mathbb{B}_*$ instead of the full vector notation (e.g. $S = 11{*}0$ instead of $S = (1, 1, *, 0)$).

To formally reason about the evolution of the network's state, we consider its asynchronous state-transition graph:

Definition 2. *For $\mathbb{BN} = \{f_1, \ldots, f_n\}$, the* state-transition graph $\mathrm{STG}(\mathbb{BN}) = (V, E)$ *is a directed graph with $V = \mathbb{B}^n$ and $E \subseteq V \times V$ given as follows:*

$$(u, v) \in E \Leftrightarrow (u \neq v \wedge \exists i \in [1, n].\ v = u[i \mapsto f_i(u)])$$

We can write $u \to v$ whenever $(u, v) \in E$. Observe that each transition within $\mathrm{STG}(\mathbb{BN})$ always updates exactly one network variable. To then study the long-term behaviour of a network, we focus on the terms *trap set* and *attractor* [17,32]:

Definition 3. *Let $\mathbb{BN} = \{f_1, \ldots, f_n\}$ be a Boolean network and $X \subseteq \mathbb{B}^n$ a set of network states. We say that X is a* trap set *when for all $x \in X$ and $y \in \mathbb{B}^n$ we have that $x \to y$ implies $y \in X$ (i.e. X cannot be escaped). We write that X is an* attractor *when X is strongly connected within* $\mathrm{STG}(\mathbb{BN})$.

Equivalently, we can also define attractors as exactly the inclusion-minimal trap sets of $\mathbb{BN}$. We write $\mathcal{A}(\mathrm{STG}(\mathbb{BN}))$ to denote the set of all attractors of $\mathbb{BN}$.

It has been shown that attractors are closely tied to the notion of biological phenotypes [26,18]. Each attractor represents a possible stable *outcome* achievable within a particular $\mathbb{BN}$. The combination of traits observable as part of this outcome then forms the actual phenotype. However, in many cases, there can be multiple attractors that exhibit the same set of traits, and thus the same biological phenotype. We formalise this concept as follows:

Definition 4. *A* character *is a set of BN variables $\mathcal{U} \subseteq [1, n]$ which cover the observable real-world properties of the system. A* trait *T is then a valuation of these character variables: $T : \mathcal{U} \to \mathbb{B}$.*

Such character variables $\mathcal{U}$ typically correspond to the network outputs, but this is not required. Each trait defines a subspace $S^T \in \mathbb{B}^n_*$ s.t. $S^T_i = T(i)$ for $i \in \mathcal{U}$, and $S^T_i = *$ otherwise. With a slight abuse of notation, we simply use T to also mean the subspace S^T when clear from context.

Definition 5. *A* phenotype *is a set of states $P \subseteq \mathbb{B}^n$ described by an arbitrary combination of the BN traits. We say that phenotype P* exists *in $\mathbb{BN}$ when $A \subseteq P$ for some $A \in \mathcal{A}(\mathrm{STG}(\mathbb{BN}))$. We say that $\mathbb{BN}$* exhibits *P when $A \subseteq P$ for all $A \in \mathcal{A}(\mathrm{STG}(\mathbb{BN}))$.*

While a *trait* is always a subspace, a phenotype is an arbitrary combination of traits. For example, assuming $n = 4$ and $\mathcal{U} = \{1, 2\}$, $11{*}{*}$ and $00{*}{*}$ are two of the four admissible traits (or rather trait subspaces). Each of these traits can represent a single phenotype, but they can also represent a combined phenotype $00{*}{*} \cup 11{*}{*}$. We typically assume that if a network admits more than one phenotype, these are mutually disjoint. Finally, note that not all traits have to belong to some phenotype (e.g. if a trait is not biologically viable).

2.2 Partially specified Boolean networks

A shortcoming of classical BNs as defined above is that to study network dynamics, all update functions must be fully known. However, this is often not realistic for large-scale systems. To address this problem, we consider the notion of *Partially Specified Boolean Networks* (PSBNs) [8] (also termed *Coloured Boolean Networks*).

In a PSBN, we can use *uninterpreted functions* as stand-ins for unknown (fixed but arbitrary) parts of the network's dynamics. Each uninterpreted function is then denoted by its *symbol* (a name) and input arguments.

Definition 6. *Let n be the number of system variables, and $\mathfrak{F}$ a set of* uninterpreted function symbols. *A* partially specified Boolean network *$\mathbb{PSBN} = \{P_1, \ldots, P_n\}$ consists of expressions P_i given by the following grammar:*

$$E ::= 0 \mid 1 \mid x \mid \neg E \mid E \wedge E \mid E \vee E \mid \mathcal{F}^{(a)}(E, \ldots, E)$$

Here, x ranges over the network variables and $\mathcal{F}$ over the uninterpreted functions of $\mathfrak{F}$ (superscript $a \in \mathbb{N}_0$ denotes the arity of $\mathcal{F}$). Other Boolean operators (e.g. $\Rightarrow$ or $\Leftrightarrow$) can be implemented as syntactic abbreviations using $\vee$, $\wedge$, and $\neg$.

In other words, a PSBN is defined using standard Boolean constants (0 and 1), variable state propositions (x) and Boolean connectives ($\neg$, $\wedge$, $\vee$), but it can also use uninterpreted functions from $\mathfrak{F}$ as a way of incorporating unknown behaviour. This makes it possible to idiomatically describe systems whose dynamics are not fully known.

Note that this definition also allows zero-arity uninterpreted functions (e.g. $\mathcal{F}^{(0)} \in \mathfrak{F}$). These are effectively unknown Boolean constants. As such, they are functionally equivalent to network inputs with an unknown value. To distinguish them, we sometimes call these uninterpreted functions *logical parameters*.

To assign meaning to a particular $\mathbb{PSBN}$, we rely on the term *interpretation*. An interpretation $\mathcal{I}$ is a function which assigns each symbol from $\mathfrak{F}$ a Boolean function of the corresponding arity. By substituting each $\mathcal{F} \in \mathfrak{F}$ in expressions $P_1, \ldots, P_n$ for its corresponding $\mathcal{I}(\mathcal{F})$ (written $P_i(\mathcal{I})$), we obtain a classical fully specified Boolean network which we denote $\mathbb{PSBN}(\mathcal{I}) = \{P_1(\mathcal{I}), \ldots, P_n(\mathcal{I})\}$.

Definitions of trap set and attractor for fully specified BNs then extend to PSBNs naturally per individual interpretations. For example, we write that a set $A \subseteq \mathbb{B}^n$ is an attractor of $\mathbb{PSBN}$ *for interpretation* $\mathcal{I}$ when it is an attractor of $\mathbb{PSBN}(\mathcal{I})$ (i.e. there are no attractors of $\mathbb{PSBN}$, only attractors of its interpretations). The definitions of character, trait and phenotype do not depend on the actual dynamics of the network (only on its variables). As such, these are identical for both BNs and PSBNs.

Note that for any given $\mathbb{PSBN}$ based on uninterpreted functions $\mathfrak{F}$, there is a logically equivalent *normalised* $\widehat{\mathbb{PSBN}}$ based on $\widehat{\mathfrak{F}}$, such that $\widehat{\mathfrak{F}}$ only admits zero-arity uninterpreted functions. The details of this conversion can be found in [12]. However, in general, the size of $\widehat{\mathfrak{F}}$ is exponential w.r.t. the arity of functions in $\mathfrak{F}$. Considering that $\widehat{\mathfrak{F}}$ only admits zero-arity logical parameters, the possible valuations of these parameters (which we also call *colours*) can be encoded as vectors $c \in \mathbb{B}^m$ where $m = |\widehat{\mathfrak{F}}|$. We then write $\mathcal{I}_c$ to denote an interpretation of $\mathbb{PSBN}$ that is encoded by a particular colour $c \in \mathbb{B}^m$.

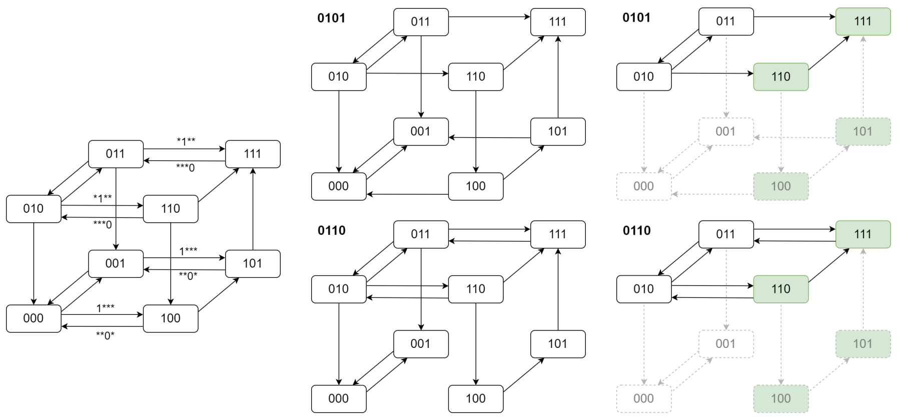
Finally, to algorithmically study the dynamics of possible PSBN interpretations, we rely on the term *coloured state-transition graph*.

Definition 7. *Let $\mathbb{PSBN} = \{P_1, \ldots, P_n\}$ be a partially specified Boolean network such that $\widehat{\mathbb{PSBN}}$ admits m logical parameters. The* coloured state-transition graph $\mathrm{STG}(\mathbb{PSBN}) = (V, C, E)$ *is a directed graph where $V = \mathbb{B}^n$, $C = \mathbb{B}^m$ and $E \subseteq V \times C \times V$ is given as $(u, c, v) \in E \Leftrightarrow (u, v) \in E(\mathrm{STG}(\mathbb{PSBN}(\mathcal{I}_c)))$.*

In other words, the coloured state-transition graph is a unifying structure which incorporates the STGs of all interpretations of $\mathbb{PSBN}$: A transition from a state u to state v is enabled for colour c if the same transition appears in the STG of $\mathbb{PSBN}(\mathcal{I}_c)$. An example of such an STG is depicted in Fig. 1a. We further explore the utility of coloured STGs in the Methods section, where we show how these can be efficiently manipulated *symbolically*.

Example 1. Consider a PSBN with $\mathfrak{F} = \{\mathcal{F}^{(2)}\}$ and $P_1 \equiv \mathcal{F}(x_1, x_2)$; $P_2 \equiv x_1 \wedge x_3$; $P_3 \equiv x_1 \vee \neg x_3$. After normalisation, we have $\widehat{\mathfrak{F}} = \{\mathcal{F}_{00}, \mathcal{F}_{01}, \mathcal{F}_{10}, \mathcal{F}_{11}\}$ and $\widehat{P}_1 \equiv ((\neg x_1 \wedge \neg x_2) \Rightarrow \mathcal{F}_{00}) \wedge ((\neg x_1 \wedge x_2) \Rightarrow \mathcal{F}_{01}) \wedge ((x_1 \wedge \neg x_2) \Rightarrow \mathcal{F}_{10}) \wedge ((x_1 \wedge x_2) \Rightarrow \mathcal{F}_{11})$.

The coloured STG of this PSBN is shown in Fig. 1a. Each edge label describes a subspace of the colour set $\mathbb{B}^m$ which enables said edge (edges without labels are enabled for all colours). Note that the presence of each edge always depends on a single logical parameter. Fig. 1b then shows STGs of two specific PSBN interpretations: $\mathcal{I}_{0101}$ and $\mathcal{I}_{0110}$ (i.e. $\mathcal{F}(x_1, x_2) = x_2$ and $\mathcal{F}(x_1, x_2) = x_1 \nLeftrightarrow x_2$).



(a) STG($\mathbb{PSBN}$) (b) STG($\mathbb{PSBN}(\mathcal{I}_c)$) (c) STG$[Q]$($\mathbb{PSBN}(\mathcal{I}_c)$)

Fig. 1: STGs of the PSBN from Example 1: (a) The full coloured STG; (b) STGs for interpretations $\mathcal{I}_{0101}$ and $\mathcal{I}_{0110}$; (c) Perturbed STGs for $Q = {*}1{*}$. Green states represent phenotype $P = 1{*}{*}$.

2.3 Phenotype control of partially specified Boolean networks

Finally, we can discuss the topic of phenotype control for PSBNs: we assume a set of phenotype states $P \subseteq \mathbb{B}^n$ and the goal is to perturb the network such that it *exhibits* this phenotype. In other words, after the perturbation is applied to the network, the network stabilizes in some attractor A such that $A \subseteq P$. However, in the presence of partially unknown dynamics, this may not be achievable using a sufficiently small perturbation. In such cases, we rely on control *robustness* to assess viability of possible perturbations.

As before, for simplicity, we first introduce some of the concepts for classical BNs and then expand these to PSBNs. First, to control the behaviour of a BN, we use the notion of variable perturbation:

Definition 8. *A* variable perturbation *is a vector $Q \in \mathbb{B}_*^n$. For every $Q_i = *$, we say that the i-th variable is* unperturbed*, whereas for $Q_i = 0$ or $Q_i = 1$, we say that the variable is* perturbed *to either* 0 *or* 1*.*

We write *fixed*(Q) and *free*(Q) to denote the subset of perturbed and unperturbed variables, respectively. The *size* of Q is the size of the *fixed*(Q) set. Furthermore, a perturbation can again be seen as a *subspace* of $\mathbb{B}^n$. We then write that the states in this subspace are *compliant* with Q.

For a particular $\mathbb{BN}$ and Q, we consider a *perturbed* state-transition graph $\text{STG}[Q](\mathbb{BN})$, which is a sub-graph of $\text{STG}(\mathbb{BN})$ *induced* by the states compliant with Q. Intuitively, the *fixed*(Q) variables are restricted to their perturbed values, while *free*(Q) variables are left to evolve without modification.

We say that a permanent variable perturbation $Q \in \mathbb{B}_*^n$ *controls* Boolean network $\mathbb{BN}$ towards phenotype P if and only if $\mathcal{A}(\text{STG}[Q](\mathbb{BN})) \subseteq P$. Intuitively, a perturbation represents an effective control strategy if it causes the network to only exhibit attractors from the desired phenotype P. However, note that $\mathcal{A}(\text{STG}[Q](\mathbb{BN}))$ is not necessarily a subset of $\mathcal{A}(\text{STG}(\mathbb{BN}))$. A permanent perturbation can disturb existing attractors or even introduce new ones.

This concept naturally applies to interpretations of PSBNs as well: given a $\mathbb{PSBN}$, an interpretation $\mathcal{I}$ and a perturbation Q, the interpretation has a perturbed $\text{STG}[Q](\mathbb{PSBN}(\mathcal{I}))$. As such, a perturbation Q controls $\mathbb{PSBN}$ under the interpretation $\mathcal{I}$ if it ensures $\mathcal{A}(\text{STG}[Q](\mathbb{PSBN}(\mathcal{I}))) \subseteq P$.

Example 2. Consider the network from Example 1, a phenotype $P = 1{*}{*}$ and a perturbation $Q = {*}1{*}$. Fig. 1c depicts the perturbed STGs of interpretations $\mathcal{I}_{0101}$ and $\mathcal{I}_{0110}$, with phenotype states highlighted in green. Perturbation $Q = {*}1{*}$ achieves control for interpretation $\mathcal{I}_{0101}$, but not for $\mathcal{I}_{0110}$.

Definition 9. *Assume a partially specified network $\mathbb{PSBN}$, a phenotype $P \subseteq \mathbb{B}^n$ and a set of admissible perturbations $\mathcal{Q} \subseteq \mathbb{B}_*^n$. The goal of the* complete *phenotype control is to compute all pairs $(c, Q) \in \mathbb{B}^m \times \mathbb{B}_*^n$ such that $Q \in \mathcal{Q}$ and Q controls $\mathbb{PSBN}(\mathcal{I}_c)$ towards phenotype P.*

Intuitively, the result of PSBN phenotype control are all combinations of colours and perturbations for which the perturbation necessarily stabilizes the associated network interpretation in the given phenotype. The set $\mathcal{Q}$ is in practice used to restrict the problem setting to perturbations of a certain size or to otherwise biologically feasible perturbations.

Perturbation size and robustness In practice, it is often unnecessary to compute the complete set of colour-perturbation pairs. The goal is instead to find the smallest functioning perturbation [39,37]. However, in partially specified networks, such minimal perturbation typically only works for a small subset of interpretations, making it potentially unreliable in practice. To address this, we consider robust phenotype control:

Definition 10. *The* robustness *ρ of a perturbation $Q \in \mathbb{B}_*^n$ is defined as:*

$$\rho(Q) = \frac{|\{c \in \mathbb{B}^m \mid Q \text{ controls } \mathbb{PSBN}(\mathcal{I}_c)\}|}{|\mathbb{B}^m|}$$

Given a robustness threshold $r \in (0, 1]$, the goal of robust *phenotype control is to compute the smallest perturbation (or perturbations) Q s.t. $\rho(Q) \geq r$.*

Ideally, a suitable perturbation Q with $\rho(Q) = 1$ achieves control for all interpretations of $\mathbb{PSBN}$. However, if no such perturbation exists, the parameter r can be tuned to achieve a trade off between perturbation size and robustness.

3 Methods

Partially specified Boolean networks suffer from both state and parameter space explosion, which in practice makes them hard to analyse exhaustively. In the context of control, this problem is further complicated by perturbation space explosion. To mitigate this, we employ symbolic state space exploration which allows us to analyse the full set of possible perturbed STGs within a single algorithm pass. This technique exploits similarities between the STGs of various network interpretations (and perturbations) to significantly speed up the exploration process compared to the naive enumeration [6,11].

3.1 Symbolic computation model

A binary decision diagram (BDD) [13] is a directed acyclic graph which represents a Boolean function. A BDD-encoded function $f : \mathbb{B}^k \to \mathbb{B}$ can be used to encode a set (or relation) of Boolean vectors $X \subseteq \mathbb{B}^k$, such that $f(x) = 1 \Leftrightarrow x \in X$. Since both network states ($\mathbb{B}^n$) and interpretations ($\mathbb{B}^m$ after normalisation) correspond to Boolean vectors, such sets have a direct BDD representation.

For sets of perturbations, a naive approach requires two Boolean variables per component, since their domain is $\mathbb{B}_*$ instead of $\mathbb{B}$. However, in [12], we introduce an alternative encoding suitable for representing perturbed state-transition systems that only requires one variable. This significantly improves the efficiency of the encoding, since fewer symbolic variables typically result in smaller BDDs.

Perturbed STG encoding Observe that in the context of a state $x \in \mathbb{B}^n$ which is *compliant* with a perturbation Q, it is sufficient to know the set *fixed*(Q) to fully reconstruct Q: we have $Q_i = x_i$ for every $i \in$ *fixed*(Q) and $Q_i = *$ otherwise.

Consequently, a relation over states x, interpretations $\mathcal{I}_c$ and perturbations Q where all states are compliant with their respective perturbations can be encoded as a relation over $\mathbb{B}^n \times \mathbb{B}^m \times \mathbb{B}^n$. Here, the last component encodes the possible sets *fixed*(Q). Since each perturbed STG$[Q](\mathbb{BN})$ only admits states compliant with Q, such representation is suitable for collectively encoding dynamics of such perturbed STGs for different perturbations.

To avoid confusion, we write $\mathcal{V} \equiv \mathbb{B}^n$ to mean the set of network states, $\mathcal{C} \equiv \mathbb{B}^m$ to mean the set of network interpretations (colours), and $\mathcal{L} \equiv \mathbb{B}^n$ to mean the set of sets of perturbed variables *fixed*(Q), collectively $\mathcal{S} = \mathcal{V} \times \mathcal{C} \times \mathcal{L}$. For $(x, l) \in \mathcal{V} \times \mathcal{L}$, we also write that $Q \equiv (x, l)$ when $Q_i = x_i$ for all $l_i = 1$ and $Q_i = *$ when $l_i = 0$ (i.e. Q can be reconstructed based on the state x and the set of perturbed variables l).

When the goal is to symbolically represent a subset of perturbations $\mathcal{Q} \subseteq \mathbb{B}^n_*$, we do this through a relation $S_{\mathcal{Q}} \subseteq \mathcal{V} \times \mathcal{L}$ where $S_{\mathcal{Q}}$ contains exactly *all* states compliant with every $Q \in S$. Finally, in some cases, we may need to directly reference the Boolean BDD variables used in our encoding. We then use the notation $\mathcal{V}_i$, $\mathcal{C}_i$, resp. $\mathcal{L}_i$ to denote the i-th BDD variable that encodes the state, colour or perturbation component of $\mathcal{S}$.

Symbolic operations Set operations ($\cap, \cup, \setminus$, etc.) on symbolic sets (or relations) are implemented through Boolean logical operators ($\wedge, \vee, \neg$, etc.) as is customary for BDDs. Furthermore, the following standard BDD operations are used:

$$\begin{aligned}
\textsc{Project}_{\mathcal{X}}(X \subseteq \mathbb{B}^k) &= \{x \in \mathbb{B}^k \mid \exists b \in \mathbb{B}.\ x[\mathcal{X} \mapsto b] \in X\} \\
\textsc{Select}_{\mathcal{X}=b}(X \subseteq \mathbb{B}^k) &= \{x \in \mathbb{B}^k \mid x \in X \wedge x[\mathcal{X}] = b\} \\
\textsc{Restrict}_{\mathcal{X}=b}(X \subseteq \mathbb{B}^k) &= \textsc{Project}_{\mathcal{X}}(\textsc{Select}_{\mathcal{X}=b}(X))
\end{aligned}$$

Here, $\mathcal{X}$ denotes a variable of the BDD encoding (e.g. $\mathcal{V}_3$). We can also use a list of conditions in the subscript of the method as a shorthand for multiple nested calls to the same method. Intuitively, Project is equivalent to existential quantification, Select is implemented through conjunction, and Restrict is a combination of both. To explore the perturbed STGs, we use:

$$\begin{aligned}
\textsc{Pre}(X \subseteq \mathcal{S}) &= \{\ (s, c, l) \mid \exists (t, c, l) \in X.\ s \rightarrow_{c,Q} t \text{ for } Q \equiv (s, l)\ \} \\
\textsc{Post}(X \subseteq \mathcal{S}) &= \{\ (t, c, l) \mid \exists (s, c, l) \in X.\ s \rightarrow_{c,Q} t \text{ for } Q \equiv (s, l)\ \}
\end{aligned}$$

Here, $s \rightarrow_{c,Q} t$ denotes that $s \rightarrow t$ within the graph $\mathrm{STG}[Q](\mathbb{PSBN}(\mathcal{I}_c))$. Intuitively, Pre and Post compute the sets of predecessors and successors of the states in X within $\mathbb{PSBN}$ under their respective interpretations $\mathcal{I}_c$ and perturbations Q. The implementation details of these operations can be found in [12].

We then rely on a method $\textsc{Trap}(X \subseteq \mathcal{S})$ which computes the *maximal trap set* within the given set X. Naively, this method can be implemented as the greatest fixed point of $\textsc{Trap}(X) = X \cap \textsc{Trap}(X \setminus \textsc{Pre}(\textsc{Post}(X)))$. However, we use a more efficient implementation based on the technique called *saturation* [8].

Finally, we use $\mathbb{VAL}^n_k$ to denote a BDD which encodes a function of n inputs that is *true* iff exactly k inputs are *true*. To construct such BDD efficiently, we observe that $\mathbb{VAL}^n_0 = \bigwedge_{i \in [1,n]} \neg x_i$, and that $\mathbb{VAL}^n_k = \bigvee_{i \in [1,n]} (\mathbb{VAL}^n_{k-1} \vee x_i)$.

3.2 Control algorithm

Our control approach is based on Algorithm 1 where we describe:

- CompletePhenotypeControl: Core algorithm that iteratively computes the *complete* control map for an admissible set $\mathcal{Q}$.
- RobustPhenotypeControl: A wrapper for the core algorithm that facilitates minimal control under the desired *robustness*.
- Enumerate: An auxiliary method to enumerate all *working* perturbations and find the maximum robustness.

Algorithm 1: Symbolic permanent phenotype control of PSBNs.

```
Fn CompletePhenotypeControl($P \subseteq \mathcal{V}, \mathcal{Q} \subseteq \mathcal{V} \times \mathcal{L}$ (encodes $\mathbb{B}^n_*$))
    universe ← $\{ (x, c, l) \in \mathcal{S} \mid (x, l) \in \mathcal{Q} \}$;
    phenotype ← universe $\cap (P \times \mathcal{C} \times \mathcal{L})$;
    phenotype_trap ← Trap(phenotype);
    non_phenotype ← universe \ phenotype_trap;
    cannot_control ← Trap(non_phenotype);
    for $i \in [1, n]$ do
        not_perturbed ← Project$_{\mathcal{V}_i}$(Select$_{\mathcal{L}_i=0}$(cannot_control));
        cannot_control ← not_perturbed ∪ Select$_{\mathcal{L}_i=1}$(cannot_control);
    control_map ← universe \ cannot_control;
    return control_map;
Fn RobustPhenotypeControl($r, P \subseteq \mathcal{V}, \mathcal{Q} \subseteq \mathcal{V} \times \mathcal{L}$)
    for $k \in [0, n]$ do
        $\mathcal{Q}_k \leftarrow \mathcal{Q} \cap (\mathcal{V} \times \mathbb{VAL}^n_k)$;
        control_map ← CompletePhenotypeControl($P, \mathcal{Q}_k$);
        $\rho_{\texttt{best}}$ ← Enumerate(1, control_map);
        if $\rho_{best} \geq r$ then return;
Fn Enumerate($i \in \mathbb{N}$, control_map $\subseteq \mathcal{S}$ (encodes $\mathbb{B}^m \times \mathbb{B}^n_*$))
    if control_map = $\emptyset$ then return 0;
    if $i > n$ then return $\frac{|\{c \in \mathcal{C} \mid \exists (x,c,l) \in \mathcal{S}.(x,c,l) \in \texttt{control\_map}\}|}{|\mathcal{C}|}$;
    not_controlled ← Restrict$_{\mathcal{L}_i=0}$(control_map);
    controlled_true ← Restrict$_{\mathcal{L}_i=1,\mathcal{V}_i=1}$(control_map);
    controlled_false ← Restrict$_{\mathcal{L}_i=1,\mathcal{V}_i=0}$(control_map);
    best ← Enumerate($i + 1$, not_controlled);
    best ← $max$(best, Enumerate($i + 1$, controlled_true));
    best ← $max$(best, Enumerate($i + 1$, controlled_false));
    return best;
```

Complete control The main idea behind Algorithm 1 is the following: A perturbation achieves phenotype control if and only if all attractors in the perturbed STG are within the phenotype subset P. Consequently, we want to *discard* any perturbation that provably retains some attractor not contained in P.

Therefore, we compute a non-phenotype trap set that is guaranteed to contain all non-phenotype attractor states in it. First, we compute a similar trap subset of the phenotype P which is guaranteed to contain all phenotype attractors. Then, we invert this set and repeat the operation to obtain a trap which is a superset of all non-phenotype attractors. Note that simply computing the largest trap set within $\mathcal{V} \setminus P$ would only cover attractors that are completely within $\mathcal{V} \setminus P$. The above-described process is necessary to also cover attractors that intersect P but are not subsets of P.

The algorithm then iterates over all network variables and performs projection in cases where the variable is *not perturbed*. Initially, the `cannot_control` set contains at least one state of the perturbed STG of each $Q \in \mathcal{Q}$ that admits a non-phenotype attractor. After this operation, `cannot_control` contains *all* states of such perturbed STG. In other words: the resulting set can depend on variable $\mathcal{V}_i$ only if $\mathcal{L}_i = 1$ (i.e. the variable is perturbed). In such cases, the role of $\mathcal{V}_i$ is to encode the actual perturbed value of the i-th network variable.

Finally, we invert the `cannot_control` set to only retain perturbations where no non-phenotype attractor state exists.

Robust control To extend this algorithm to robust control, we test the perturbations of increasing size (using the $\mathbb{VAL}_k^n$ BDD) and use a recursive ENUMERATE method to find perturbations with maximal robustness. Notice that the necessity of this step makes our algorithm *semi-symbolic*.

Instead of iterating through all possible control strategies of size k, procedure ENUMERATE recursively branches into three cases for each variable i: i is not perturbed, i is perturbed to *true*, and i is perturbed to *false*. Note that each call completely eliminates both $\mathcal{L}_i$ and $\mathcal{V}_i$ from `control_map` (for the case of $\mathcal{L}_i = 0$, $\mathcal{V}_i$ was already eliminated by the core algorithm). As such, once $i > n$, the resulting `control_map` only depends on variables $\mathcal{C}_i$ and we can use it to compute the robustness.

Note that for simplicity, we do not store the actual perturbations with maximal robustness explicitly in the algorithm. However, these can be easily reconstructed from the recursion path in the ENUMERATE algorithm. Furthermore, note that the recursion in the ENUMERATE algorithm can be replaced using *projected iteration*, where we first project the `control_map` to the admissible $l \in \mathcal{L}$, and then project only to the state variables perturbed within each such l. This approach can be faster as it uses fewer symbolic steps. However, it is also highly specific to each BDD library. As such, we chose to present the more widely applicable algorithm.

4 Evaluation

In this section, we evaluate our proposed semi-symbolic method. For the implementation, we use the Rust language and libraries of the AEON tool [7]. This prototype implementation is available as an open-source GitHub repository[3]. The repository also includes auxiliary scripts we used for the measurements and

[3] https://github.com/sybila/biodivine-pbn-control/

Table 1: Comprehensive overview of the tested real-world Boolean networks. The first four columns contain the model name and the counts of network inputs, all variables, and perturbable variables, respectively. Column $\mathcal{Q}_{\leq 3}$ represents the number of all admissible perturbations of size up to three. Sixth column gives a reference to the literature that describes each model. Lastly, the seventh column contains variables that we do not allow to be perturbed.

Model	**Ins**	**Vars**	**Per.**	$\mathcal{Q}_{\leq 3}$	**Ref.**	**Uncontrollable vars**
Cardiac	2	13	11	6,252	[23]	`Tbx1`, `Tbx5`
Red. MAPK	4	14	11	25,008	[22]	`Apoptosis`, `Growth_Arrest`, `Proliferation`
ERBB	1	19	18	14,354	[24]	`pRB1`
Tumour	2	30	24	69,380	[19]	`Apoptosis`, `Metastasis`, `Invasion`, `Migration`, `EMT`, `CellCycleArrest`
Cell Fate	2	31	26	88,612	[14]	`Apoptosis`, `Survival`, `Death`, `Division`, `NonACD`
Full MAPK	2	49	46	2,010,768	[22]	`Apoptosis`, `Growth_Arrest`, `Proliferation`

the raw results. All experiments were performed using a computer with AMD Ryzen Threadripper 2990WX 32-Core Processor and 64GB of memory.

4.1 Performance

Benchmark model set We use real-world Boolean networks to evaluate our method. All tested networks contain input nodes which can be viewed as functionally equivalent to zero-arity uninterpreted functions in PSBNs. We thus treat these inputs as unknown external signal beyond our control.

Table 1 lists all models and their relevant characteristics, including number of inputs, variables, perturbable variables, and reference to the original publication. We also state the number of admissible perturbations of size up to three. Previous work shows that such relatively small size is both realistic to implement in practice [10,18,38] and robust enough in the presence of partially unknown dynamics [12]. This number therefore represents how many models would need to be explored in a brute-force based methods for the models of the given size. The table also lists variables which we explicitly do not allow to be perturbed. These variables are mostly outputs and they represent traits which induce individual phenotypes. Therefore, perturbing these variables would lead to trivial control which is neither viable nor interesting.

The first model illustrates cardiac progenitor cells differentiation into the first heart field (FHF) or second heart field (SHF) [23]. The second and sixth models represent Mitogen-Activated Protein Kinase (MAPK) network representing signalling pathways involved in diverse cellular processes including cancer deregulations. We use both the full and reduced versions of this model as stated

Table 2: Performance of PSBN control. The first two columns contain model and phenotype names. Then, for perturbations of the size up to three, we list the computation time and maximal robustness found in perturbations of this size. Last column gives the number of minimal perturbations with $\rho = 1.0$.

Model	Phenotype	Size 1		Size 2		Size 3		# Min. per.
		Time	ρ	Time	ρ	Time	ρ	($\rho = 1.0$)
Cardiac	FHF	<1s	0.5	<1s	1.0	<1s	1.0	4
	SHF	<1s	0.5	<1s	1.0	<1s	1.0	2
	No mesoderm	<1s	0.5	<1s	1.0	<1s	1.0	3
Red. MAPK	Apoptosis	<1s	1.0	<1s	1.0	<1s	1.0	1
	Growth arrest	<1s	0.75	<1s	1.0	<1s	1.0	1
	No decision	<1s	1.0	<1s	1.0	<1s	1.0	1
	Proliferation	<1s	0.25	<1s	1.0	<1s	1.0	3
ERBB	Phosphor.	<1s	1.0	<1s	1.0	<1s	1.0	7
	Non-phospor.	<1s	1.0	<1s	1.0	<1s	1.0	8
Tumour	Apoptosis	2s	1.0	8s	1.0	23s	1.0	2
	EMT	<1s	0.5	3s	1.0	11s	1.0	15
	Hybrid	<1s	0.25	5s	1.0	24s	1.0	3
	Metastasis	<1s	0.5	<1s	1.0	1s	1.0	6
Cell Fate	Apoptosis	<1s	0	4s	1.0	58s	1.0	24
	Naive	<1s	0	2s	1.0	29s	1.0	8
	Necrosis	<1s	1.0	<1s	1.0	11s	1.0	1
	Survival	<1s	1.0	2s	1.0	21s	1.0	1
Full MAPK	Apoptosis	<1s	1.0	7s	1.0	14min	1.0	6
	Growth arrest	4s	0.75	4s	1.0	22min	1.0	47
	No decision	3s	0.81	107s	1.0	22min	1.0	45
	Proliferation	3s	0.25	3s	1.0	20min	1.0	8

in [22]. The third model depicts the key event preceding breast cancer cells proliferation which is the hyper-phosphorylation and subsequent lack of pRB. This process is regulated by ERBB kinase, the lack of which is considered a breast cancer marker [35]. The fourth model focuses on specific conditions which lead to a metastatic tumour [19]. Finally, the cell fate model provides a high-level understanding of the interplays between pro-survival, necrosis, and apoptosis pathways in response to death receptor-mediated signals [14].

Performance evaluation on real-world models In Table 2, we show results of computing phenotype control on all our benchmark models from Table 1. We use real-world phenotypes as described in the source literature. These are typically subspaces obtained by fixing network outputs to the desired values. The details of exact phenotypes can be found in the git repository.

For each model and its phenotype, we computed all working perturbations of size up to three. We then show the times needed for individual computations (including enumeration and robustness calculation), the highest robustness achieved for each case, and the number of minimal perturbations with 100% robustness.

Table 3: Minimal perturbations for reduced MAPK model. The first column is target phenotype while other columns contain minimal perturbations for unperturbed model, model with over-expressed `EGFR`, and model with `FGFR3` gain-of-function. Variables divided by / stand for perturbing either of them. ∅ is used when no perturbation is necessary.

Phenot.	**Unperturbed**	**Perturbed** EGFR=1	**Perturbed** FGFR3=1
Apoptosis	DNA_dmg=1, TGFBR_st=1, FRS2=1	DNA_dmg=1, TGFBR_st=1, ERK=0, p53=1	DNA_dmg=1, TGFBR_st=1, ERK=0, p53=1
¬Apoptosis	∅	AKT=1, ERK=1, MSK=0, PTEN=0, p14=0, p53=0	AKT=1, ERK=1, MSK=0, PTEN=0, p14=0, p53=0
Prolif.	ERK=1	p14=0, p53=0	{p14/p53=0, FRS2=1}, {p14/p53=0, PI3K=1}, {p14/p53=0, EGFR=1}
¬Prolif.	∅	DNA_dmg=1, TGFBR_st=1, AKT=0, ERK=0, MSK=0, PI3K=0, PTEN=1, p53=1	DNA_dmg=1, TGFBR_st=1, AKT=0, ERK=0, MSK=0, PI3K=0, PTEN=1, p53=1
No decis.	∅	MSK=0	MSK=0
¬No decis.	DNA_dmg=1, TGFBR_st=1 EGFR=1, ERK=1, FRS2=1, p53=1	∅	DNA_dmg=1, TGFBR_st=1 ERK=1, FRS2=1, p53=1, PI3K=1

The performance of our method is almost instant for small-sized models which is very convenient for practical use. As for the bigger models, the method also performs sufficiently, nonetheless, due to complex unpredictable nature of PSBN dynamics, its performance may vary significantly from case to case. For example, even though Cell Fate model has only one more perturbable variable than the Tumour model, we can notice that perturbations of size three take on average twice longer to compute. The differences in PSBN dynamics can also have big impact within the same model for different phenotypes. Even though the phenotypes within the same model are all of the same size, we can notice that in any of the bigger models and perturbations of size three, the control takes much longer to compute. There are many factors which can contribute to these big performance differences such as fixed-point nature of the trap set algorithm (which might require numerous iterations in case of long paths with few neighbors) or heuristic BDD representation.

4.2 MAPK case study

Now we demonstrate how phenotype control can be used to replicate observations conducted in [22] and even strengthen these results with a more robust assessment of the model. In [22], various perturbations of a reduced MAPK model are simulated and their effect on phenotypes is observed. Specifically, networks with all inputs set to *false* and with `EGFR` or `FGFR3` over-expressed (gain-of-function) are exposed to a set of further perturbations. The model

Table 4: Full MAPK model with uninterpreted functions controlled to the apoptosis phenotype. Each of the three sections covers a different set of uninterpreted functions (increasing in size). For each case, we state: maximal robustness, count of perturbations having such robustness (- for no perturbation) and the computation time. N/A denotes a 24h timeout.

Uninter. functions	`EGFR`, `FGFR3`, `p53`, `p14`			`EGFR`, `FGFR3`, `p53`, `p14`, `PTEN`			`EGFR`, `FGFR3`, `p53`, `p14`, `PTEN`, `PI3K`, `AKT`		
Colours	806,400			2,419,200			1,161,216,000		
Metrics	ρ	#	Time	ρ	#	Time	ρ	#	Time
$\emptyset$	0.58	-	4s	0.39	-	5s	0.24	-	22s
Size 1	0.87	1	74s	0.58	4	2min	0.59	1	37min
Size 2	1.0	17	27min	0.87	4	38min	0.87	1	20.2hrs
Size 3	1.0	1165	5.6hrs	1.0	68	8.2hrs	N/A	N/A	N/A

has three outputs (`Apoptosis`, `Growth_arrest`, `Proliferation`) and three attractors are observed: apoptosis (`Apoptosis=Growth_arrest=1`), proliferation (`Proliferation=1`) and no decision (all outputs set to false).

We first compute phenotype control on the fully specified but reduced model. We list discovered minimal perturbations for network variants of interest in Table 3. Here, the perturbations which were also discovered in [22] are shown as green. In the enumeration, where appropriate, we only considered perturbations with over-expressed `EGFR` or `FGFR3`, as in the original paper. We also list the minimal perturbations working for the unperturbed network. This way we can compare such perturbations with the `EGFR` and `FGFR3` over-expressed variants. We can see that with our method we were able to not only replicate all solutions from [22], but also conveniently obtain more perturbation options, including the truly minimal controls (if we consider over-expressions of `EGFR` or `FGFR3` as perturbations, the further perturbations to these networks lead to a non-minimal perturbations in most of the cases).

The interest of the original MAPK study [22] is also to observe the impact on phenotypes caused by various gain- or loss-of-function mutations. Here, authors replace such functions with constants to simulate these effects. Nonetheless, such an approach could be too restrictive: a mutation could alter the function in unpredictable ways instead of knocking-out (resp. over-expressing) the variable permanently. To model such mutations, we employ the uninterpreted functions of the PSBN framework. This application is demonstrated in Table 4. Here, we selected apoptosis phenotype as the phenotype of interest (the "healthy" phenotype preserving non-cancerous cell behaviour). We then replace the dynamics of variables studied in [22] with uninterpreted functions in the *full* MAPK model.

Our method performs well in spite of the significant amount of colours introduced by the model uncertainty. Moreover, we see that perturbations with relatively small size are still capable of successful control. The obtained observations can be for example used to refute hypotheses about model's update functions. If a candidate perturbation is shown as non-viable, this indicates that the colours where such perturbation works do not represent the true dynamics of the system. This can guide further refinements of the partially specified model.

5 Conclusion

In this work, we presented a novel problem of phenotype control for partially specified Boolean networks. We proposed a semi-symbolic method using permanent variable perturbations to solve this problem. Our approach offers a practical and flexible solution for stabilizing a network at specific collection of traits, regardless of its initial state. We have also demonstrated the applicability of our method to real-world networks. In future work, we would like to further optimize our method, investigate other types of perturbations for phenotype control and further explore applicability of these methods.

References

1. Abou-Jaoudé, W., Traynard, P., Monteiro, P.T., Saez-Rodriguez, J., Helikar, T., Thieffry, D., Chaouiya, C.: Logical modeling and dynamical analysis of cellular networks. Frontiers in Genetics **7**, 94 (2016)
2. Albert, R.: Boolean modeling of genetic regulatory networks. In: Complex Networks, pp. 459–481. Springer (2004)
3. Aracena, J., Goles, E., Moreira, A., Salinas, L.: On the robustness of update schedules in Boolean networks. Biosystems **97**(1), 1–8 (2009)
4. Barbuti, R., Gori, R., Milazzo, P., Nasti, L.: A survey of gene regulatory networks modelling methods: From differential equations, to Boolean and qualitative bioinspired models. Journal of Membrane Computing **2**(3), 207–226 (2020)
5. Baudin, A., Paul, S., Su, C., Pang, J.: Controlling large Boolean networks with single-step perturbations. Bioinformatics **35**(14), i558–i567 (2019)
6. Beneš, N., Brim, L., Huvar, O., Pastva, S., Šafránek, D., Šmijáková, E.: Aeon. py: Python library for attractor analysis in asynchronous boolean networks. Bioinformatics **38**(21), 4978–4980 (2022)
7. Beneš, N., Brim, L., Pastva, S., Šafránek, D.: AEON: Attractor bifurcation analysis of parametrised Boolean networks. In: International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 12224, pp. 569–581. Springer (2020)
8. Beneš, N., Brim, L., Pastva, S., Šafránek, D.: Bdd-based algorithm for scc decomposition of edge-coloured graphs. Logical Methods in Computer Science **18** (2022)
9. Beneš, N., Brim, L., Huvar, O., Pastva, S., Šafránek, D.: Boolean Network Sketches: A Unifying Framework for Logical Model Inference. Bioinformatics (2023)
10. Borriello, E., Daniels, B.C.: The basis of easy controllability in Boolean networks. Nature Communications **12**(1), 1–15 (2021)
11. Brim, L., Pastva, S., Šafránek, D., Šmijáková, E.: Parallel one-step control of parametrised Boolean networks. Mathematics **9**(5), 560 (2021)
12. Brim, L., Pastva, S., Šafránek, D., Šmijáková, E.: Temporary and permanent control of partially specified boolean networks. Biosystems **223**, 104795 (2023)
13. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **35**(8), 677–691 (1986)

14. Calzone, L., Tournier, L., Fourquet, S., Thieffry, D., Zhivotovsky, B., Barillot, E., Zinovyev, A.: Mathematical modelling of cell-fate decision in response to death receptor engagement. PLOS Computational Biology **6**(3), 1–15 (2010)
15. Choo, S.M., Ban, B., Joo, J.I., Cho, K.H.: The phenotype control kernel of a biomolecular regulatory network. BMC systems biology **12**(1), 1–15 (2018)
16. Choo, S.M., Cho, K.H.: An efficient algorithm for identifying primary phenotype attractors of a large-scale boolean network. BMC systems biology **10**(1), 1–14 (2016)
17. Cifuentes Fontanals, L., Tonello, E., Siebert, H.: Control strategy identification via trap spaces in Boolean networks. In: Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 12314, pp. 159–175. Springer (2020)
18. Cifuentes Fontanals, L., Tonello, E., Siebert, H.: Control in Boolean networks with model checking. Frontiers in Applied Mathematics and Statistics **8**, 838546 (04 2022)
19. Cohen, D.P., Martignetti, L., Robine, S., Barillot, E., Zinovyev, A., Calzone, L.: Mathematical modelling of molecular pathways enabling tumour cell invasion and migration. PLOS Computational Biology **11**(11), 1–29 (2015)
20. Fiedler, B., Mochizuki, A., Kurosawa, G., Saito, D.: Dynamics and control at feedback vertex sets. I: Informative and determining nodes in regulatory networks. Journal of Dynamics and Differential Equations **25**(3), 563–604 (2013)
21. Geris, L., Gomez-Cabrero, D.: An introduction to uncertainty in the development of computational models of biological processes. In: Uncertainty in Biology, pp. 3–11. Springer (2016)
22. Grieco, L., Calzone, L., Bernard-Pierrot, I., Radvanyi, F., Kahn-Perles, B., Thieffry, D.: Integrative modelling of the influence of MAPK network on cancer cell fate decision. PLOS Computational Biology **9**(10), e1003286 (2013)
23. Herrmann, F., Groß, A., Zhou, D., Kestler, H.A., Kühl, M.: A boolean model of the cardiac gene regulatory network determining first and second heart field identity. PloS one **7**(10), e46798 (2012)
24. Ito, N., Kuwahara, G., Sukehiro, Y., Teratani, H.: Segmental arterial mediolysis accompanied by renal infarction and pancreatic enlargement: a case report. Journal of Medical Case Reports **6**(1), 1–5 (2012)
25. Kim, J., Park, S.M., Cho, K.H.: Discovery of a kernel for controlling biomolecular regulatory networks. Scientific reports **3**(1), 1–9 (2013)
26. Klarner, H., Heinitz, F., Nee, S., Siebert, H.: Basins of attraction, commitment sets, and phenotypes of boolean networks. IEEE/ACM transactions on computational biology and bioinformatics **17**(4), 1115–1124 (2018)
27. Kobayashi, K., Hiraishi, K.: Optimal control of asynchronous Boolean networks modeled by Petri nets. In: Biological Process & Petri Nets. pp. 7–20. CEUR-WS (2011)
28. Mandon, H., Haar, S., Paulevé, L.: Temporal reprogramming of Boolean networks. In: Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 10545, pp. 179–195. Springer, Springer (2017)
29. Mandon, H., Su, C., Haar, S., Pang, J., Paulevé, L.: Sequential reprogramming of Boolean networks made practical. In: Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 11773, pp. 3–19. Springer (2019)
30. Martin, A.J., Dominguez, C., Contreras-Riquelme, S., Holmes, D.S., Perez-Acle, T.: Graphlet based metrics for the comparison of gene regulatory networks. PLOS ONE **11**(10) (2016)

31. Pardo, J., Ivanov, S., Delaplace, F.: Sequential reprogramming of biological network fate. In: Bortolussi, L., Sanguinetti, G. (eds.) Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 11773, pp. 20–41. Springer (2019)
32. Paulevé, L.: Marker and source-marker reprogramming of most permissive boolean networks and ensembles with BoNesis. Peer Community Journal (2023)
33. Rozum, J.C., Deritei, D., Park, K.H., Gómez Tejeda Zañudo, J., Albert, R.: pystablemotifs: Python library for attractor identification and control in Boolean networks. Bioinformatics **38**(5), 1465–1466 (2022)
34. Rozum, J.C., Gómez Tejeda Zañudo, J., Gan, X., Deritei, D., Albert, R.: Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks. Science Advances **7**(29), eabf8124 (2021)
35. Sahin, Ö., Fröhlich, H., Löbke, C., Korf, U., Burmester, S., Majety, M., Mattern, J., Schupp, I., Chaouiya, C., Thieffry, D., et al.: Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance. BMC Systems Biology **3**(1), 1–20 (2009)
36. Su, C., Pang, J.: A dynamics-based approach for the target control of Boolean networks. In: ACM International Conference on Bioinformatics, Computational Biology and Health Informatics. pp. 1–8. Association for Computing Machinery (2020)
37. Su, C., Pang, J.: Sequential temporary and permanent control of Boolean networks. In: Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 12314, pp. 234–251. Springer (2020)
38. Su, C., Paul, S., Pang, J.: Controlling large Boolean networks with temporary and permanent perturbations. In: International Symposium on Formal Methods. Lecture Notes in Computer Science, vol. 11800, pp. 707–724. Springer (2019)
39. Su, C., Paul, S., Pang, J.: Scalable control of asynchronous Boolean networks. In: Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 11773, pp. 364–367. Springer (2019)
40. Zañudo, J.G.T., Albert, R.: Cell fate reprogramming by control of intracellular network dynamics. PLOS Computational Biology **11**(4), 1–24 (2015)