Sequential Reprogramming of Boolean Networks Made Practical

Hugues Mandon*[1,2], Cui Su*[3], Stefan Haar[1], Jun Pang[3,4], and Loïc Paulevé[5]

[1] LSV, ENS Cachan, INRIA, CNRS, Université Paris-Saclay, France
[2] LRI UMR 8623, Univ. Paris-Sud – CNRS, Université Paris-Saclay, France
[3] SnT, University of Luxembourg, Luxembourg, Luxembourg
[4] FSTC, University of Luxembourg, Esch-sur-Alzette, Luxembourg
[5] Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Abstract. We address the sequential reprogramming of gene regulatory networks modelled as Boolean networks. We develop an attractor-based sequential reprogramming method to compute all sequential reprogramming paths from a source attractor to a target attractor, where only attractors of the network are used as intermediates. Our method is more practical than existing reprogramming methods as it incorporates several practical constraints: (1) only biologically observable states, viz. attractors, can act as intermediates; (2) certain attractors, such as apoptosis, can be avoided as intermediates; (3) certain nodes can be avoided to perturb as they may be essential for cell survival or difficult to perturb with biomolecular techniques; and (4) given a threshold k, all sequential reprogramming paths with no more than k perturbations are computed. We compare our method with the minimal one-step reprogramming and the minimal sequential reprogramming on a variety of biological networks. The results show that our method can greatly reduces the number of perturbations compared to the one-step reprogramming, while having comparable results with the minimal sequential reprogramming. Moreover, our implementation is scalable for networks of more than 60 nodes.

Keywords: Cell reprogramming · Boolean networks · Attractors

1 Introduction

Cell reprogramming is one of the big discoveries of regenerative medicine. Takahashi and Yamanaka in [23] demonstrated that cell fate decisions could be reversed: a mature cell can be reprogrammed into an induced pluripotent stem cell. Even though different cocktails of transcription factors have been found to switch cell phenotypes [8,22], the identification of specific transcription factors for a particular task remains a big obstacle. Blindly testing combinations of transcription factors is unfeasible due to the high cost of biological experiments.

Computational models of cell dynamics enable the *in silico* prediction of reprogramming targets. Qualitative models, notably Boolean networks, allow

* Co-first authors.

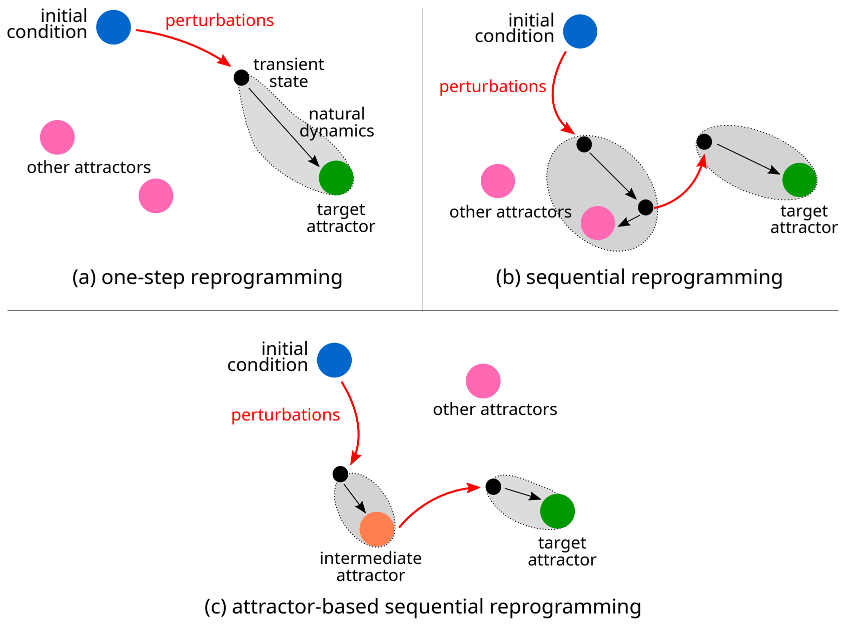


Fig. 1: Different flavors of Boolean networks reprogramming.

accounting the influences between numerous genes by requiring few modelling parameters. Thus, they turn out to be well suited for modelling cellular differentiation processes and thereby predict perturbations for their control [1,5,6,7,18,24]. In Boolean networks, each gene or protein is modelled as a binary variable, which can only take 0 or 1 as its value: a value of 0 means that the gene or protein is inactive, whereas a value of 1 means that the gene or protein is active. Each variable is assigned with a Boolean function, which determines the next value of the variable given the current values of other variables of the network. The computation of the next states depends on the chosen update mode for the variables. In this paper, we focus on the asynchronous updating mode where a single variable is updated at a time, selected non-deterministically. The long term dynamics of a Boolean network is described as attractors, which can be either single-state attractors (fixed points), or cyclic attractors.

Cell reprogramming consists of triggering a change of cellular phenotype. In the context of Boolean networks, phenotypes are modelled by the attractors. Cellular reprogramming becomes then a control problem: driving the dynamics of the network from a source attractor to a target attractor. In order to control a network, the system is perturbed out of its actual state. These perturbations can be applied instantaneously (for an instant), temporarily (for limited time), or permanently (mutations). In this paper, we focus on instantaneous perturbations. Moreover, the perturbations can take place at different "times", and as such, multiple kinds of reprogramming strategies can be found in the literature.

Existing works focus on one-step reprogramming [5,7,10,16,18], or in rare instances, on sequential reprogramming, e.g., [12]. One-step reprogramming allows applying perturbations only once as shown in Fig. 1(a). On the other hand, sequential reprogramming identifies a sequence of perturbations to be applied at different intermediate states. The intermediate states can be either a transient state or a state in an attractor. As illustrated in Fig. 1(b), a set of perturbations are applied to the initial state, which stirs the network to a transient state. After one-step spontaneous evolution, we apply another set of perturbations to the new transient state. This leads the network dynamics to a state in the strong basin of the target attractor, from which the network always eventually reaches the target attractor. By taking advantage of the natural dynamics of the network, sequential reprogramming can provide alternative predictions to one-step reprogramming, notably requiring considerably less perturbations [12]. However, in order to apply the perturbations at the correct time, sequential reprogramming requires *complete observability* of the network (i.e., the state of the network is known at any discrete time), which is rarely feasible in practice. This motivates us to develop an attractor-based sequential reprogramming as illustrated by Fig. 1(c), where perturbations should be applied only at attractors. Since the attractors can be observed experimentally, the attractor-based sequential reprogramming only requires *partial observability* of the network. Moreover, in experiments, perturbations need to take time before effectively changing the values of the variables. Attractor-based sequential reprogramming captures this requirement well, as the network dynamics remains in the attractor when perturbations are applied.

In this paper, we describe in detail our attractor-based sequential reprogramming to compute sequential reprogramming paths through other attractors of the network. We compare the performance of this new method with the minimal one-step reprogramming and the minimal sequential reprogramming. The results show that all the three methods are efficient in terms of computation time. Both sequential reprogramming methods can greatly reduce the number of perturbations compared to the minimal one-step reprogramming. Even though our attractor-based sequential reprogramming may need a few more perturbations than the minimal sequential reprogramming for some cases, the paths identified by our method are more easily transferable to biological experiment protocols.

Outline. Section 2 gives preliminary notions on Boolean networks. Section 3 addresses the attractor-based sequential reprogramming, with definitions and an algorithm to compute the solutions. Section 4 evaluates it by comparing its performance with the minimal one-step reprogramming and the minimal sequential reprogramming on several biological networks. Lastly, section 5 discusses the results and reviews further the state of the art.

2 Background

2.1 Boolean networks

A Boolean network (BN) describes elements of a dynamical system with binary-valued nodes and interactions between elements with Boolean functions. It is formally defined as follows.

Definition 1 (Boolean networks). *A Boolean network is a tuple* $\mathsf{BN} = (\mathbf{x}, \mathbf{f})$ *where* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ *such that each* $x_i, 1 \leq i \leq n$ *is a Boolean variable and* $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ *is a tuple of Boolean functions over* $\mathbf{x}$. $|\mathbf{x}| = n$ *denotes the number of variables.*

In what follows, i will always range between 1 and n, unless stated otherwise. A Boolean network $\mathsf{BN} = (\mathbf{x}, \mathbf{f})$ may be viewed as a directed graph $\mathcal{G}_{\mathsf{BN}} = (V, E)$ where $V = \{v_1, v_2 \ldots, v_n\}$ is the set of vertices or nodes and for every $1 \leq i, j \leq n$, there is a directed edge from v_j to v_i if and only if f_i depends on x_j. An edge from v_j to v_i will be often denoted as $v_j \to v_i$. A path from a vertex v to a vertex v' is a (possibly empty) sequence of edges from v to v' in $\mathcal{G}_{\mathsf{BN}}$. For the rest of the exposition, we assume that an arbitrary but fixed network BN of n variables is given to us and $\mathcal{G}_{\mathsf{BN}} = (V, E)$ is its associated directed graph.

A state s of BN is an element in $\{0, 1\}^n$. Let $\mathbf{S}$ be the set of states of BN. For any state $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, and for every i, the value of s_i, often denoted as $\mathsf{s}[i]$, represents the value that the variable x_i takes when the BN 'is in state $\mathbf{s}$'. For some i, suppose f_i depends on $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$. Then $f_i(\mathbf{s})$ will denote the value $f_i(\mathsf{s}[i_1], \mathsf{s}[i_2], \ldots, \mathsf{s}[i_k])$. For two states $\mathsf{s}, \mathsf{s}' \in \mathbf{S}$, the Hamming distance between $\mathbf{s}$ and $\mathbf{s}'$ will be denoted as $\mathsf{hd}(\mathbf{s}, \mathbf{s}')$ and $\arg(\mathsf{hd}(\mathbf{s}, \mathbf{s}')) \subseteq \{1, 2, \ldots, n\}$ will denote the set of indices in which $\mathbf{s}$ and $\mathbf{s}'$ differ. For a state $\mathbf{s}$ and a subset $\mathbf{S}' \subseteq \mathbf{S}$, the Hamming distance between $\mathbf{s}$ and $\mathbf{S}'$ is defined as the minimum of the Hamming distances between $\mathbf{s}$ and all the states in $\mathbf{S}'$. That is, $\mathsf{hd}(\mathbf{s}, \mathbf{S}') = \min_{\mathbf{s}' \in \mathbf{S}'} \mathsf{hd}(\mathbf{s}, \mathbf{s}')$. We let $\arg(\mathsf{hd}(\mathbf{s}, \mathbf{S}'))$ denote the set of subsets of $\{1, 2, \ldots, n\}$ such that $I = \arg(\mathsf{hd}(\mathbf{s}, \mathbf{S}'))$ if and only if I is a set of indices of the variables that realise this Hamming distance.

2.2 Dynamics of Boolean networks

We assume that the Boolean network evolves in discrete time steps. It starts initially in a state $\mathbf{s}_0$ and its state changes in every time step according to the update functions $\mathbf{f}$. The updating may happen in various ways. Every such way of updating gives rise to a different dynamics for the network. In this article, we focus on the fully asynchronous update mode, but the method is actually generic to any update mode, as it computes on the resulting global transition system.

Definition 2 (Asynchronous dynamics of Boolean networks). *Suppose* $\mathbf{s}_0 \in \mathbf{S}$ *is an initial state of* BN. *The asynchronous evolution of* BN *is a function* $\xi : \mathbb{N} \to \wp(\mathbf{S})$ *such that* $\xi(0) = \mathbf{s}_0$ *and for every* $j \geq 0$, *if* $\mathbf{s} \in \xi(j)$ *then* $\mathbf{s}' \in \xi(j{+}1)$ *is a possible* next state *if and only if either* $\mathsf{hd}(\mathbf{s}, \mathbf{s}') = 1$ *and* $\mathbf{s}'[i] = f_i(\mathbf{s})$ *where* $\{i\} = \arg(\mathsf{hd}(\mathbf{s}, \mathbf{s}'))$ *or* $\mathsf{hd}(\mathbf{s}, \mathbf{s}') = 0$ *and there exists* i *such that* $\mathbf{s}'[i] = f_i(\mathbf{s})$.

Note that the asynchronous dynamics is non-deterministic – the value of exactly one variable is updated in a single time-step. The index of the variable that is updated is not known in advance. Henceforth, when we talk about the dynamics of BN, we shall mean the asynchronous dynamics as defined above.

The dynamics of a Boolean network can be represented as a *state transition graph* or a *transition system (TS)*.

Definition 3 (Transition system of BN). *The transition system of* BN*, denoted by the generic notation* TS *is a tuple* $(\mathbf{S}, \rightarrow)$ *where the vertices are the set of states* $\mathbf{S}$ *and for any two states* $\mathbf{s}$ *and* $\mathbf{s}'$ *there is a directed edge from* $\mathbf{s}$ *to* $\mathbf{s}'$*, denoted* $\mathbf{s} \rightarrow \mathbf{s}'$ *iff* $\mathbf{s}'$ *is a possible next state according to the asynchronous evolution function* ξ *of* BN*.*

2.3 Attractors and basins of attraction

A path from a state $\mathbf{s}$ to a state $\mathbf{s}'$ is a (possibly empty) sequence of transitions from $\mathbf{s}$ to $\mathbf{s}'$ in TS. A path from a state $\mathbf{s}$ to a subset $\mathbf{S}'$ of $\mathbf{S}$ is a path from $\mathbf{s}$ to any state $\mathbf{s}' \in \mathbf{S}'$. For any state $\mathbf{s} \in \mathbf{S}$, let $\mathsf{pre}_{\mathsf{TS}}(\mathbf{s}) = \{\mathbf{s}' \in \mathbf{S} \mid \mathbf{s}' \rightarrow \mathbf{s}\}$ and let $\mathsf{post}_{\mathsf{TS}}(\mathbf{s}) = \{\mathbf{s}' \in \mathbf{S} \mid \mathbf{s} \rightarrow \mathbf{s}'\}$. $\mathsf{pre}_{\mathsf{TS}}(\mathbf{s})$ contains all the states that can reach $\mathbf{s}$ by performing a single transition in TS and $\mathsf{post}_{\mathsf{TS}}(s)$ contains all the states that can be reached from $\mathbf{s}$ by a single transition in TS. $\mathsf{pre}_{\mathsf{TS}}(\mathbf{s})$ and $\mathsf{post}_{\mathsf{TS}}(\mathbf{s})$ are often called the set of *predecessors* and *successors* of $\mathbf{s}$. Note that, by definition, $\mathsf{hd}(\mathbf{s}, \mathsf{pre}_{\mathsf{TS}}(\mathbf{s})) \leq 1$ and $\mathsf{hd}(\mathbf{s}, \mathsf{post}_{\mathsf{TS}}(\mathbf{s})) \leq 1$. $\mathsf{pre}_{\mathsf{TS}}$ and $\mathsf{post}_{\mathsf{TS}}$ can be lifted to a subset $\mathbf{S}'$ of $\mathbf{S}$ as: $\mathsf{pre}_{\mathsf{TS}}(\mathbf{S}') = \bigcup_{\mathbf{s} \in \mathbf{S}'} \mathsf{pre}_{\mathsf{TS}}(\mathbf{s})$ and $\mathsf{post}_{\mathsf{TS}}(\mathbf{S}') = \bigcup_{\mathbf{s} \in \mathbf{S}'} \mathsf{post}_{\mathsf{TS}}(\mathbf{s})$. We define $\mathsf{pre}^{i+1}_{\mathsf{TS}}(\mathbf{S}') = \mathsf{pre}_{\mathsf{TS}}(\mathsf{pre}^{i}_{\mathsf{TS}}(\mathbf{S}'))$ and $\mathsf{post}^{i+1}_{\mathsf{TS}}(\mathbf{S}') = \mathsf{post}_{\mathsf{TS}}(\mathsf{post}^{i}_{\mathsf{TS}}(\mathbf{S}'))$ where $\mathsf{pre}^{0}_{\mathsf{TS}}(\mathbf{S}') = \mathsf{post}^{0}_{\mathsf{TS}}(\mathbf{S}') = \mathbf{S}'$. For a state $\mathbf{s} \in \mathbf{S}$, $\mathsf{reach}_{\mathsf{TS}}(\mathbf{s})$ denotes the set of states $\mathbf{s}'$ such that there is a path from $\mathbf{s}$ to $\mathbf{s}'$ in TS and can be defined as the fixpoint of the successor operation which is often denoted as $\mathsf{post}^{*}_{\mathsf{TS}}$. Thus, $\mathsf{reach}_{\mathsf{TS}}(\mathbf{s}) = \mathsf{post}^{*}_{\mathsf{TS}}(\mathbf{s})$.

Definition 4 (Attractor). *An attractor* A *of* TS *(or of* BN*) is a minimal subset of states of* $\mathbf{S}$ *such that for every* $\mathsf{s} \in A, \mathsf{reach}_{\mathsf{TS}}(\mathsf{s}) = A$*.*

Remark that attractors are the bottom strongly connected component of TS.

Any state which is not part of an attractor is a transient state. An attractor A of TS is said to be reachable from a state $\mathbf{s}$ if $\mathsf{reach}_{\mathsf{TS}}(\mathbf{s}) \cap A \neq \emptyset$. Attractors represent the stable behaviour of the BN according to the dynamics. Assuming strong fairness, the network starting at any initial state $\mathbf{s}_0 \in \mathbf{S}$ will eventually end up in one of the attractors of TS and remain there forever unless perturbed.

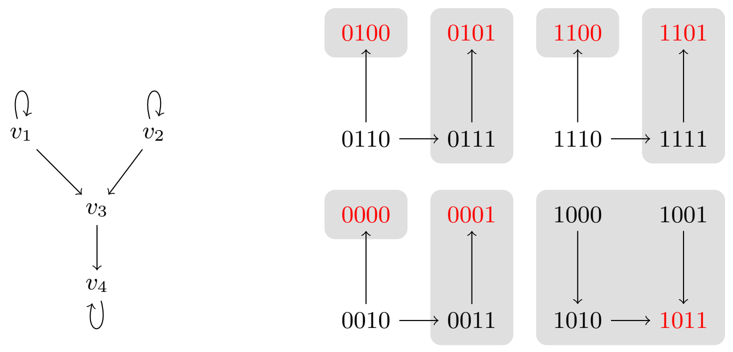
For an attractor A of TS, we define a subset of states of $\mathbf{S}$ called the strong basins of A, denoted as $\mathsf{bas}^{S}_{\mathsf{TS}}(A)$, as follows.

Definition 5 (Strong basin). *Let* A *be an attractor of* TS*. The* **strong basin** *of attraction of* A *with respect to* TS*, is defined as* $\mathsf{bas}_{\mathsf{TS}}(A) = \{\mathsf{s} \in \mathbf{S} | \mathsf{reach}_{\mathsf{TS}}(\mathsf{s}) \cap \bigcup A' = \emptyset\}$ *where the union is over all attractors* A' *of* TS *such that* $A' \neq A$*.*

The definition of strong basin guarantees that any state $\mathbf{s}$ in $\mathsf{bas}_{\mathsf{TS}}(A)$ can only reach the attractor A and cannot reach any other attractor $A', A' \neq A$ of BN.[6]

Example 1. Consider the following four-node network $\mathsf{BN} = (\mathbf{x}, \mathbf{f})$ where $\mathbf{x} = (x_1, x_2, x_3, x_4)$, and $\mathbf{f} = (f_1, f_2, f_3, f_4)$ where $f_1 = x_1, f_2 = x_2$, $f_3 = x_1 \wedge \neg x_2$ and $f_4 = x_3 \vee x_4$. The graph of the network $\mathcal{G}_{\mathsf{BN}}$ and its associated transition system TS is given in Fig. 2. TS has seven attractors marked in red. Their corresponding strong basins of attractions are shown by enclosing grey regions of a lighter shade.

[6] Henceforth, we drop the subscript TS for the sake of simplicity.



(a) $\mathcal{G}_{\mathsf{BN}}$ (b) $\mathsf{TS}_{\mathsf{BN}}$

Fig. 2: The graph of BN and its transition system, with the attractors in red.

3 Attractor-based Sequential Reprogramming

3.1 Motivation

In most methods on cellular reprogramming using Boolean networks [18,7,16], all perturbations are done at once, and the system is left to stabilize itself towards the desired target attractor. However, allowing perturbations to be performed at different points in time opens alternative reprogramming paths, possibly less costly. In general, sequential reprogramming allows the network to be perturbed in any state (transient states or states in an attractor) [19,12]. This requires complete observability of the system, which is very hard to obtain experimentally.

To make the sequential reprogramming practical, we design an attractor-based sequential reprogramming, which only requires partial observability of the network. The principle of this method is to use other attractors as intermediate states for the reprogramming. At each step, we apply a set of perturbations to stir the dynamics towards a state in the strong basin of an intermediate attractor (or a target attractor). We then let the network evolve spontaneously to the intermediate attractor (or the target attractor). We repeat the above procedure until the network reaches the target attractor. In this paper, we focus on instantaneous perturbations, while applying the perturbations longer will not affect the reachability of the target attractor. In practice, based on empirical experience, biologists may be able to determine how long it takes for the network to stabilize in an intermediate attractor, i.e., the timing to apply the next perturbations. In that case, if the intermediate attractors are single-state attractors, partial observability is not required. However, if the intermediate attractors are cyclic attractors, an observation of the state might still be required.

A feasible reprogramming method has to encode practical considerations. In most cases, some variables cannot be perturbed, either because they represent an external cause the experimenter cannot change, or a set of multiple genes and proteins that would require a lot more work to perturb, or a transcription factor impacting only the gene or protein hasn't been found. Moreover, some attractors might not be suitable as intermediate states, because they lead to the death or disease of the cell. Thus, the algorithm we will describe in Section 3.3 provides options to avoid perturbing user-specified variables and/or avoid passing user-specified attractors.

The general principle of this method can be applied to other means to compute the required perturbations for the system to reach a target attractor, given an initial state in an attractor.

3.2 The reprogramming problem

In this work, we are interested in instantaneous perturbations, thus we define reprogramming of a BN as follows.

Definition 6 (Reprogramming). *A reprogramming set* C *of a* BN *is a (possibly empty) subset of* $\{1, 2, \ldots, n\}$. *For a state* $\mathbf{s} \in \mathbf{S}$, *the application of* C *to* $\mathbf{s}$ *reprograms the state of* BN *from* $\mathbf{s}$ *to* $\mathbf{s}' \in \mathbf{S}$, *such that* $\mathbf{s}'[i] = 1 - \mathbf{s}[i]$ *if* $i \in \mathsf{C}$ *and* $\mathbf{s}'[i] = \mathbf{s}[i]$ *otherwise.*

Since the perturbations are applied instantaneously, only the state of BN is changed while the Boolean functions remain the same. Based on the above definition, we define one-step reprogramming of a BN as follows.

Definition 7 (One-step reprogramming). *Given a source attractor* A_s *and a target attractor* A_t *of* BN, *find a state* $\mathbf{s} \in A_s$ *and a reprogramming set* C, *such that the dynamics of* BN *always eventually reaches* A_t *after the application of* C *to* $\mathbf{s}$.

According to [16], we can easily obtain the following proposition.

Proposition 1. *A one-step reprogramming* $\mathsf{C}^{A_s \to A_t}(\mathbf{s})$ *(*$\mathbf{s} \in A_s$*) from* A_s *to* A_t *is minimal if and only if*

1. $\mathsf{C}(\mathbf{s}) \in \mathsf{bas}(A_t)$ *and* $\mathsf{C} = \arg(\mathsf{hd}(\mathbf{s}, \mathsf{bas}(A_t)))$.
2. $\forall \mathsf{s}' \in A_s$, $\mathsf{hd}(\mathsf{s}', \mathsf{bas}(A_t)) \geq \mathsf{hd}(\mathsf{s}, \mathsf{bas}(A_t))$.

We denote a minimal one-step reprogramming from A_s to A_t as $\mathsf{C}^{A_s \to A_t}_{min}(\mathbf{s})$. A minimal one-step reprogramming drives the dynamics of BN from A_s to a state in the strong basin of A_t, from which spontaneous evolution will eventually guide the network to A_t.

As explained in Section 3.1, attractor-based sequential reprogramming can provide new solutions apart from the one-step reprogramming paths. Let $|\mathcal{A}_{\mathsf{BN}}|$ denote the total number of attractors of BN. We define attractor-based sequential reprogramming as follows.

Definition 8 (Attractor-based sequential reprogramming). *Given* A_s *(a source attractor) and* A_t *(a target attractor) of* BN, *find a sequence of attractors* $\{A_1, A_2, \ldots, A_m\}$ *of* BN, *where* $A_1 = A_s, A_m = A_t$, $A_i \neq A_j$ *for any* $i, j \in [1, m]$ *and* $2 \leq m \leq |\mathcal{A}_{\mathsf{BN}}|$, *such that a sequence of minimal one-step reprogramming*

$\{\mathsf{C}_{min}^{A_1 \to A_2}, \mathsf{C}_{min}^{A_2 \to A_3}, \ldots, \mathsf{C}_{min}^{A_{m-1} \to A_m}\}$ *always eventually reaches* A_t (A_m). *We call it a attractor-based sequential path, denoted as*

$$\rho : A_1 \xrightarrow{\mathsf{C}_{min}^{A_1 \to A_2}} A_2 \xrightarrow{\mathsf{C}_{min}^{A_2 \to A_3}} A_3 \xrightarrow{\cdots} \ldots \xrightarrow{\mathsf{C}_{min}^{A_{m-1} \to A_m}} A_m$$

$(|\mathsf{C}_{min}^{A_1 \to A_2}| + |\mathsf{C}_{min}^{A_2 \to A_3}| + \ldots + |\mathsf{C}_{min}^{A_{m-1} \to A_m}|)$ *is the total number of perturbations.*

Due to the diversity of biological networks, there does not exist one universal reprogramming strategy that suits all different networks. Hence, we develop an algorithm to compute all attractor-based sequential reprogramming paths satisfying the following constraints:

1. the total number of perturbations is less than a threshold;
2. certain attractors can be avoided as intermediates;
3. certain nodes of the network can be avoided to be perturbed.

These constraints encode practical considerations described in Section 3.1 and thus lead to biologically feasible reprogramming paths. We describe such an algorithm in the next section.

3.3 Algorithm

Let $\mathsf{BN} = (\mathbf{x}, \mathbf{f})$ be a Boolean Network of size $n = |\mathbf{x}|$. Let U be the set of variables that cannot be perturbed, $\mathsf{A_s}$ be an attractor of the network, which is the initial state of the system, and $\mathsf{A_t}$ be another attractor of the network, which is the target to reprogram to.

Algorithm 1 describes the algorithm to compute sequential paths from $\mathsf{A_s}$ to $\mathsf{A_t}$, using other attractors as intermediate steps. The inputs are: the Boolean Network BN, the initial attractor $\mathsf{A_s}$, the target attractor $\mathsf{A_t}$, the set of attractors $\mathcal{A}$ that can act as intermediate states, and the set of variables U that can not be perturbed. The set $\mathcal{A}$ excludes the attractors that cannot act as intermediates, such as the source attractor and the undesired attractors.[7]

The algorithm uses a modified Hamming distance hd_m between the states of the transition system. Between a state s and a state t, this modified Hamming distance $\mathsf{hd}_m(s,t)$ is defined as:

$$\mathsf{hd}_m(s,t) = \begin{cases} \infty & \text{if } \exists v \in \mathsf{U}, s[v] \neq t[v] \\ \mathsf{hd}(s,t) & \text{otherwise} \end{cases}$$

The modified Hamming distance between two sets of states S and T is defined as: $\mathsf{hd}_m(S,T) = min_{s \in S, t \in T}(\mathsf{hd}_m(s,t))$.

To compute the sequential paths from $\mathsf{A_s}$ to $\mathsf{A_t}$ using other attractors as intermediate states, we have to compute the strong basin of $\mathsf{A_t}$, which is $\mathsf{bas}(\mathsf{A_t})$. Since we only use the distance between a state and a basin, let HB_m, the distance between a set of states S and the basin of a set of states T, be defined as

[7] We refer details on attractor detection to [13].

Algorithm 1 Inevitable reprogramming of BNs from A_s to A_t

```
1: procedure COMPUTATION_OF_INEVITABLE_PATHS($\mathsf{BN}, \mathsf{A_s}, \mathsf{A_t}, \mathcal{A}, \mathsf{U}$)
2:     $\mathsf{max\_dist} = \mathsf{HB}_m(\mathsf{BN}, \mathsf{U}, \mathsf{A_s}, \mathsf{A_t}))$
3:     $\mathcal{L}_{\mathsf{A_s}}$ = new empty dictionary
4:     if $\mathsf{max\_dist} < \infty$ then
5:         $a = \mathsf{arg\_HB}(\mathsf{BN}, \mathsf{U}, \mathsf{A_s}, \mathsf{A_t})$
6:         ▷ Associate (distance, [perturbations list]) to the [path]
7:         $\mathcal{L}_{\mathsf{A_s}}$.add([$\mathsf{A_t}$] : ($\mathsf{max\_dist}$, [$a$]))
8:         ▷ Associate the minimal length of all paths from $\mathsf{A_s}$ to $\mathsf{A_t}$
9:         $\mathcal{L}_{\mathsf{A_s}}$.add("min" : $\mathsf{max\_dist}$)
10:    $\mathsf{list} := \emptyset$
11:    for $\mathsf{A} \in \mathcal{A}$ do
12:        $d = \mathsf{HB}_m(\mathsf{BN}, \mathsf{U}, \mathsf{A}, \mathsf{A_t})$
13:        if $d < \mathsf{max\_dist}$ then
14:            $\mathsf{list}$.add($\mathsf{A}$)
15:            $\mathcal{L}_{\mathsf{A}}$ = map()
16:            $a = \mathsf{arg\_HB}(\mathsf{BN}, \mathsf{U}, \mathsf{A}, \mathsf{A_t})$
17:            ▷ Associate (distance, [perturbations list]) to the [path]
18:            $\mathcal{L}_{\mathsf{A}}$.add([$\mathsf{A_t}$] : ($d$, [$a$]))
19:            ▷ Associate minimal length of all paths from $\mathsf{A}$ to $\mathsf{A_t}$
20:            $\mathcal{L}_{\mathsf{A}}$.add("min" : $d$)
21:    ▷ Recursively computes the paths with attractors as intermediate steps
22:    while $\mathsf{list} \neq \emptyset$ do
23:        $l := \emptyset$
24:        for $\mathsf{A_1} \in \mathcal{A}$ do
25:            for $\mathsf{A_2} \in \mathsf{list}$ do
26:                $d = \mathsf{HB}_m(\mathsf{BN}, \mathsf{U}, \mathsf{A_1}, \mathsf{A_2})$
27:                if $d \neq \infty$ and $d + \mathcal{L}_{\mathsf{A_2}}[\text{"min"}] \leq \mathsf{max\_dist}$ then
28:                    $l$.add($\mathsf{A_1}$)
29:                    for $\mathsf{path} \in \mathcal{L}_{\mathsf{A_2}} \setminus \{\text{"min"}\}$ do
30:                        $td = d + \mathcal{L}_{\mathsf{A_2}}[\mathsf{path}][0]$    ▷ total length of the new path to $\mathsf{A_t}$
31:                        if $td \leq \mathsf{max\_dist}$ and $\mathsf{A_2} \notin \mathsf{path}$ then
32:                            if $\mathcal{L}_{\mathsf{A_1}}$ does not exists then
33:                                $\mathcal{L}_{\mathsf{A_1}}$ = map()
34:                                ▷ Associate minimal length of all paths from $\mathsf{A}$ to $\mathsf{A_t}$
35:                                $\mathcal{L}_{\mathsf{A_1}}$.add("min" : $td$)
36:                            $a = \mathsf{arg\_HB}(\mathsf{BN}, \mathsf{U}, \mathsf{A_1}, \mathsf{A_2})$
37:                            ▷ Associate (distance, [perturbations]) to the [path]
38:                            $\mathcal{L}_{\mathsf{A_1}}$.add([$\mathsf{A_2}$] + $\mathsf{path}$ : ($td$, [$a$] + $\mathcal{L}_{\mathsf{A_2}}[\mathsf{path}][1]$))
39:                            if $td < \mathcal{L}_{\mathsf{A_1}}[\text{"min"}]$ then
40:                                $\mathcal{L}_{\mathsf{A_1}}[\text{"min"}] = td$
41:        $\mathsf{list} = l$
42:    return $\mathcal{L}_{\mathsf{A_s}}$
```

Algorithm 2 Distance functions

```
1: function $\mathsf{HB}_m(\mathsf{BN}, \mathsf{U}, S, T)$
2:     $B = \mathsf{bas}(T)$
3:     ▷ Details on the computation of basinS can be found in [16,17]
4:     return $min_{s \in S, t \in B}(\mathsf{hd}_m(\mathsf{U}, s, t))$
5: function $\mathsf{hd}_m(\mathsf{BN}, \mathsf{U}, s, t)$
6:     $sum = 0$
7:     for $i = 1, i \leq n, i++$ do
8:         if $s[i] \neq t[i]$ then
9:             if $i \in \mathsf{U}$ then
10:                return $\infty$
11:            $sum = sum + |s[i] - t[i]|$
12:    return $sum$
13: function $\mathsf{arg\_HB}(\mathsf{BN}, \mathsf{U}, S, T)$
14:    $min = \mathsf{HB}_m(\mathsf{BN}, \mathsf{U}, S, T)$
15:    if $min = \infty$ then
16:        Fail("infinite distance")
17:    $D = \text{map}()$
18:    for $s \in S$ do
19:        for $t \in T$ do
20:            if $\mathsf{hd}_m(\mathsf{U}, s, t) = min$ then
21:                for $i = 1, i \leq n, i++$ do
22:                    if $s[i] \neq t[i]$ then
23:                        ▷ Associate the desired value of the variable $i$ to $t_i$
24:                        $D.\text{add}(i : t_i)$
25:    return $D$
```

$\mathsf{HB}_m(S, T) = \mathsf{hd}_m(S, \mathsf{bas}(T))$. Algorithm 2 describes how to compute both of these distances, as well as how to compute the argument of HB_m, including the set of variables that realize the minimum Hamming distance and the desired value of these variables. The distance between $\mathsf{A_s}$ and the basin of $\mathsf{A_t}$, $\mathsf{max\_dist} = \mathsf{HB}_m(\mathsf{A_s}, \mathsf{A_t})$, will be used as a benchmark for the next computations: this is the maximum number of perturbations allowed to reach $\mathsf{A_t}$.

An empty dictionary $\mathcal{L}_{\mathsf{A_s}}$ is created, to store the possible paths. If $\mathsf{max\_dist} < \infty$, the perturbed variables, $a = \mathsf{arg\_HB}(\mathsf{BN}, \mathsf{U}, \mathsf{A_s}, \mathsf{A_t})$ are computed. The path, represented by a list of targets to reach in order to reach the next one, $[\mathsf{A_t}]$ is added as an entry of the dictionary, with the value $(\mathsf{max\_dist}, [a])$. This dictionary regroups all paths from $\mathsf{A_s}$ to $\mathsf{A_t}$, the first value is the length of the path, and the second is how to get from one attractor to the next one in the list. A special value is added to the dictionary, "min", which is the minimal length of all the paths from $\mathsf{A_s}$ to $\mathsf{A_t}$, and it is given the value $\mathsf{max\_dist}$.

Then, for all attractor A in $\mathcal{A}$, the distance $d = \mathsf{HB}_m(\mathsf{A}, \mathsf{A_t})$ is computed. If this distance d is strictly lower than $\mathsf{max\_dist}$[8], then A is added to a list of attractors list and a dictionary $\mathcal{L}_{\mathsf{A}}$ is created. We add to $\mathcal{L}_{\mathsf{A}}$ the entry $[\mathsf{A_t}]$ to

[8] In this case, if $\mathsf{max\_dist} = \infty$, any non infinite distance is considered strictly lower.

which we associate the length of the path, d, and the perturbations made in a list, $[\mathsf{arg\_HB}(\mathsf{BN}, \mathsf{U}, \mathsf{A}, \mathsf{A_t})]$. The path is a list of the attractors to reach in the right order. The perturbations made are a set, a dictionary in our case, containing the variables to perturb and the desired values. This set is put in a list: each set of the list is a set of perturbations to go from the current attractor to the next one in the path defined above. A special value "min" is added to the dictionary, in the same way as for $\mathcal{L}_{\mathsf{A_s}}$, to store the minimal length of paths from A to $\mathsf{A_t}$.

The list list is used to recursively compute the shortest paths. As long as list is not empty, the following steps are done:

1. First, create an empty list l.
2. Then, from all attractor $\mathsf{A_1}$ in $\mathcal{A}$, for all attractor $\mathsf{A_2}$ in list, the distance $d = \mathsf{HB}_m(\mathsf{A_1}, \mathsf{A_2})$ is computed. If this distance plus $\mathcal{L}_{\mathsf{A_2}}[\text{"min"}]$[9] is lower than $\mathsf{max\_dist}$, then for every path path in $\mathcal{L}_{\mathsf{A_2}}$, the total distance $d + \mathcal{L}_{\mathsf{A_2}}[\mathsf{path}][0]$[10] is computed. If this distance is lower or equal to $\mathsf{max\_dist}$ and if $\mathsf{A_1} \notin \mathsf{path}$, a new entry $[\mathsf{A_2}] + \mathsf{path}$[11] is added to $\mathcal{L}_{\mathsf{A_1}}$, with the value $(d + \mathcal{L}_{\mathsf{A_2}}[\mathsf{path}][0], [\mathsf{arg\_HB}(\mathsf{A_1}, \mathsf{A_2})] + \mathcal{L}_{\mathsf{A_2}}[\mathsf{path}][1])$. The first value, the distance, is the one to go from $\mathsf{A_1}$ to $\mathsf{A_t}$ using $\mathsf{A_2}$ as an intermediate step, and the paths from $\mathsf{A_2}$ to $\mathsf{A_t}$ already computed. The second value is the set of variables to perturb, using the same principle. If the dictionary does not exist, it is created, and "min" is updated or created. Moreover, $\mathsf{A_1}$ is added to l.
3. Lastly, the value of list is changed to match l, $\mathsf{list} = l$.

When this loop is over, all paths are in $\mathcal{L}_{\mathsf{A_s}}$, with the associated length and steps of variables to perturb, and $\mathcal{L}_{\mathsf{A_s}}$ is returned.

4 Evaluation

To demonstrate the efficiency and the efficacy of our attractor-based sequential reprogramming described in Algorithm 1, we compare its performance with the minimal one-step reprogramming [16] and the minimal sequential reprogramming [12] on a variety of biological networks.

4.1 Reprogramming strategies

To drive a network from a source state to a target attractor, the minimal one-step reprogramming [16] computes a minimal set of perturbations to be conducted simultaneously, and the minimal sequential reprogramming [12] computes shortest sequential paths, where any state may act as an intermediate state. Different from [12], the attractor-based sequential reprogramming (this work) identifies all the sequential paths with at most k perturbations, where only attractors (biologically observable states) can play the role of intermediate states. We compute

[9] This value is the minimal length path from $\mathsf{A_1}$ to $\mathsf{A_t}$.
[10] As $\mathsf{A_2}$ is in list, $\mathcal{L}_{\mathsf{A_2}}$ exists.
[11] Here, the + is the usual concatenation for lists.

the reprogramming paths for all combinations of source and target attractors of the studied networks with the three methods. For the attractor-based sequential reprogramming, the maximal number of perturbations allowed is set as the number of perturbations required by the minimal one-step reprogramming; and we assume all the nodes can be perturbed, thus $\mathsf{U} = \emptyset$ due to the lack of relevant biological knowledge. The three methods are implemented as part of the software tool ASSA-PBN [14]. All the experiments are performed on a computer with a CPU of Intel Core i7 @3.1 GHz and 8 GB of DDR3 RAM[12].

4.2 Benchmark biological networks

We give a short description of the biological networks on which we test the three different reprogramming methods of Boolean networks. Table 1 gives an overview of the sizes and number of attractors for these networks. All the attractors of the networks are single-state attractors.

– The myeloid differentiation network is designed to model myeloid differentiation from common myeloid progenitors to megakaryocytes, erythrocytes, granulocytes and monocytes [11].
– The cardiac gene regulatory network is constructed for the early cardiac gene regulatory network of the mouse, including the core genes required for the cardiac development and the FHF/SHF determination [9].
– The ERBB receptor regulated G1/S transition network enables us to identify potential targets in the treatment of trastuzumab resistant breast cancer [20].
– The tumour network is constructed to study the role of individual mutations or their combinations in the metastatic process [5].
– The PC12 cell network models the temporal sequence of protein signalling, transcriptional response and subsequent autocrine feedback [15].
– The model of hematopoietic cell specification recaps cytokine induced differentiation, several reported gene knockdowns and the reprogramming of pre-B cells [6].
– The model of bortezomib responses can predict responses to the lower bortezomib exposure and the dose-response curve for bortezomib [4].

4.3 Results on the myeloid differentiation network

Let us analyse in more depth the predictions obtained on the the myeloid differentiation network. Figure 3 depicts its influence graph, and Table 2 lists its six attractors, all being fixed points, four of which correspond to megakaryocyte (A_2), erythrocyte (A_3), granulocyte (A_5) and monocyte (A_6) [11]. Table 3 describes the number of perturbations required by the three methods ($|\mathsf{C}_O|$, $|\mathsf{C}_A|$, and $|\mathsf{C}_S|$) for this network. The first column and the first row stand for the

[12] Executable and data are available at the following link: `https://github.com/cuisu/attractor_based_sequential_reprogramming`.

network	# nodes	# edges	#A	range of $\lvert\mathsf{C}\rvert$			time (seconds)		
				$\lvert\mathsf{C}_O\rvert$	$\lvert\mathsf{C}_A^{\mathsf{min}}\rvert$	$\lvert\mathsf{C}_S\rvert$	T_O	T_A	T_S
myeloid	11	30	6	1-5	1-4	1-4	0.02	0.04	0.21
cardiac	15	39	6	1-9	1-8	1-4	0.23	0.63	2.28
ERBB	20	52	3	1-9	1-8	1-5	0.05	0.07	0.49
tumour	32	158	9	1-10	1-9	1-6	1.54	5.99	387.04
PC12	33	62	7	1-11	1-10	1-10	0.39	3.21	95.10
hematopoietic	33	88	5	1-13	1-12	1-12	1.89	4.87	8067.73
bortezomib	67	135	5	1-21	1-15	$*$	50.24	106.91	$*$

Table 1: An overview of the networks and the evaluation results. O, A and S stand for the minimal one-step reprogramming, the attractor-based sequential reprogramming and the minimal sequential reprogramming, respectively.

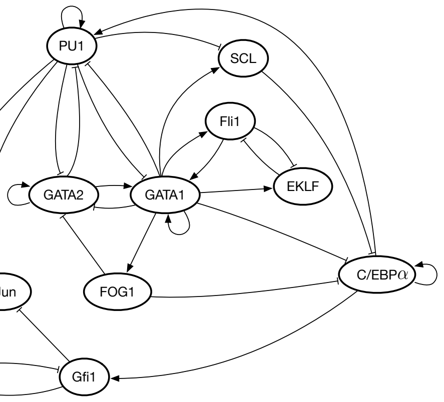


Fig. 3: Structure of the myeloid network. Rightarrow and bar arrow represent activation and inhibition, respectively.

source and the target attractors, respectively. The minimal one-step reprogramming needs more perturbations since it only allows to apply perturbations once. By choosing appropriate states as intermediates, the sequential reprogramming can reduce the number of perturbations for a few cases (e.g. from A_2 (A_3, A_4 or A_5) to A_6). The minimal number of perturbations required by the two sequential reprogramming methods are identical for this network. Besides the shortest paths, the attractor-based sequential reprogramming also identifies paths with at most $\lvert\mathsf{C}_O\rvert$ perturbations. For instance, there are in total three attractor-based sequential paths from A_2 to A_6:

– $\rho_1 : A_2 \xrightarrow{\text{GATA1,EgrNab,PU1,cJun, C/EBP}\alpha} A_6$,;
– $\rho_2 : A_2 \xrightarrow{\text{GATA1, Fli1}} A_4 \xrightarrow{\text{PU1}} A_1 \xrightarrow{\text{C/EBP}\alpha} A_6$;
– $\rho_3 : A_2 \xrightarrow{\text{GATA1,PU1}} A_1 \xrightarrow{\text{C/EBP}\alpha} A_6$.

Path ρ_1 is also a shortest one-step path, which requires 5 perturbations. Paths ρ_2 and ρ_3 only require 4 and 3 perturbations, respectively. An interesting observation is that the sequential paths may require the perturbation of the same gene

	GATA2	GATA1	FOG1	EKLF	Fli1	SCL	C/EBPα	PU1	cJun	EgrNab	Gfi1
A_1	0	0	0	0	0	0	0	1	1	1	0
A_2	0	1	1	0	1	1	0	0	0	0	0
A_3	0	1	1	1	0	1	0	0	0	0	0
A_4	0	0	0	0	0	0	0	0	0	0	0
A_5	0	0	0	0	0	0	1	1	0	0	1
A_6	0	0	0	0	0	0	1	1	1	1	0

Table 2: Attractors of the myeloid network.

	A_1			A_2			A_3			A_4			A_5			A_6		
	$\lvert C_O\rvert$	$\lvert C_A\rvert$	$\lvert C_S\rvert$	$\lvert C_O\rvert$	$\lvert C_A\rvert$	$\lvert C_S\rvert$	$\lvert C_O\rvert$	$\lvert C_A\rvert$	$\lvert C_S\rvert$	$\lvert C_O\rvert$	$\lvert C_A\rvert$	$\lvert C_S\rvert$	$\lvert C_O\rvert$	$\lvert C_A\rvert$	$\lvert C_S\rvert$	$\lvert C_O\rvert$	$\lvert C_A\rvert$	$\lvert C_S\rvert$
A_1	0	0	0	3	3	3	3	3	3	1	1	1	3	3	3	1	1	1
A_2	2	2	2	0	0	0	2	2	2	2	2	2	4	4	4	5	3, 4, 5	3
A_3	2	2	2	2	2	2	0	0	0	1	1	1	3	3	3	5	3, 5	3
A_4	1	1	1	2	2	2	2	2	2	0	0	0	2	2	2	4	2, 4	2
A_5	1	1	1	3	3	3	3	3	3	2	2	2	0	0	0	3	2, 3	2
A_6	1	1	1	3	3	3	3	3	3	2	2	2	2	2	2	0	0	0

Table 3: The number of perturbations computed by the three reprogramming methods for the myeloid differentiation network. The first column and the first row stand for the source and the target attractors, respectively.

multiple times, which will not happen for the one-step reprogramming method. For instance, a sequential path from A_5 to A_6 is $\rho : A_5 \xrightarrow{\text{C/EBP}\alpha} A_1 \xrightarrow{\text{C/EBP}\alpha} A_6$. By perturbing 'C/EBP$\alpha$' twice, we can achieve the sequential reprogramming from A_5 to A_6.

4.4 Results on the benchmark biological networks

An overview of the evaluation results on the seven networks is given in Table 1. It is worth noting that Table 3 gives the number of perturbations for every pair of source and target attractors of the myeloid differentiation network, while in Table 1, columns $|C_O|$, $|C_A^{\mathsf{min}}|$, and $|C_S|$ summarise the minimal number of perturbations required by the three methods for all pairs of source and target attractors of the seven biological networks. [13] In Table 1, $|C_A^{\mathsf{min}}|$ only considers the shortest attractor-based sequential paths, instead of all the identified paths (see $|C_A|$ in Table 3) with less than $|C_O|$ perturbations.

In general, the sequential strategy results in less perturbations. Even though the attractor-based sequential method requires a few more perturbations than the minimal sequential reprogramming, it uses biological observable states as intermediates and thus is considered more realistic and applicable. In particular, the attractor-based sequential control can reduce up to 9 perturbations compared to the minimal one-step control. Columns T_O, T_A and T_S of Table 1 give the total computation time. We can see that all three methods are efficient and scale well for large networks. Even though the attractor-based sequential reprogramming takes a bit longer than the one-step reprogramming, it identifies a number of potential applicable solutions.

[13] '*' means the algorithm fails to return any result within five hours. We excluded the 'apoptosis' attractor for the tumour network for the evaluation.

5 Discussion

Combining the available techniques from computer science with the constraints of experimental protocols in biology, in this paper, we have designed attractor-based sequential reprogramming of Boolean networks. Compared to one-step reprogramming [16], where all perturbations are applied only once, our method identifies a sequence of perturbations to be applied sequentially. Taking full advantage of spontaneous evolutions, our method requires less perturbations and thus results in lower experimental costs. Different from the sequential reprogramming [12], our method only uses other attractors as intermediates. Therefore, it does not require complete observability, except within cyclic attractors, which makes its application more feasible in biological experiments.

Moreover, our method allows avoiding some variables to be perturbed and some attractors to be used as intermediate steps, which differs from a previously developed sequential reprogramming method [12]. These constrains key observations in practice, as some biological networks have genes that cannot yet be influenced by transcription factors (or they can be influenced at a very high cost), and some attractors such as apoptosis of the cell shouldn't be viable intermediate steps. Our method sits in a middle ground between one-step reprogramming [16] and sequential reprogramming [12].

Existing works mainly focus on one-step reprogramming [2,3,5,6,7,21,24], considering various kinds of perturbations and targeted dynamical properties. Predictions are obtained following different techniques, with probabilistic modelling in [3,5,6], or qualitative modelling in [2,7,21,24]. Sequential reprogramming is also studied in the literature [1,19,12] using quite different approaches: Abou-Jaoudé et al. [1] applied model checking to verify that a set of perturbations can reprogram the cell correctly, using other attractors as intermediate steps if needed, Ronquist et al. [19] used a quantitative model that returns a specific time for the perturbations to be made; lastly in the work of Mandon et al. [12], the perturbations can be done at any time, but require precise knowledge of the state of the system (i.e., complete observability).

In future work, besides relaxing the observability within cyclic attractors, we plan to address attractor-based sequential reprogramming with temporary perturbations (i.e., sustained for a limited time). This corresponds to another classical experimental setting in cellular reprogramming, and should provide alternative and potentially shorter sequences of perturbations.

Acknowledgement. This research was supported by the ANR-FNR project AlgoReCell (ANR-16-CE12-0034; FNR INTER/ANR/15/11191283); Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02); and by the project SEC-PBN funded by University of Luxembourg. Cui Su was also partially supported by the COST Action IC1405.

References

1. Abou-Jaoudé, W., Monteiro, P.T., Naldi, A., Grandclaudon, M., Soumelis, V., Chaouiya, C., Thieffry, D.: Model checking to assess T-helper cell plasticity. Frontiers in Bioengineering and Biotechnology **2** (2015)
2. Biane, C., Delaplace, F.: Abduction based drug target discovery using boolean control network. In: Feret, J., Koeppl, H. (eds.) Computational Methods in Systems Biology. pp. 57–73. Springer International Publishing, Cham (2017)
3. Chang, R., Shoemaker, R., Wang, W.: Systematic search for recipes to generate induced pluripotent stem cells. PLOS Computational Biology **7**(12), e1002300 (2011)
4. Chudasama, V., Ovacik, M., Abernethy, D., Mager, D.: Logic-based and cellular pharmacodynamic modeling of bortezomib responses in U266 human myeloma cells. Journal of Pharmacology and Experimental Therapeutics **354**(3), 448–458 (2015)
5. Cohen, D.P.A., 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), e1004571 (2015)
6. Collombet, S., van Oevelen, C., Ortega, J.L.S.and Abou-Jaoudé, W., Di Stefano, B., Thomas-Chollier, M., Graf, T., Thieffry, D.: Logical modeling of lymphoid and myeloid cell specification and transdifferentiation. Proceedings of the National Academy of Sciences **114**(23), 5792–5799 (2017)
7. Crespo, I., Perumal, T.M., Jurkowski, W., del Sol, A.: Detecting cellular reprogramming determinants by differential stability analysis of gene regulatory networks. BMC Systems Biology **7**(1), 140 (2013)
8. Graf, T., Enver, T.: Forcing cells to change lineages. Nature **462**(7273), 587–594 (2009)
9. Herrmann, F., Gro, 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**, 1–10 (10 2012)
10. Jo, J., Hwang, S., Kim, H.J., Hong, S., Lee, J.E., Lee, S.G., Baek, A., Han, H., Lee, J.I., Lee, I., et al.: An integrated systems biology approach identifies positive cofactor 4 as a factor that increases reprogramming efficiency. Nucleic Acids Research **44**(3), 12031215 (2016)
11. Krumsiek, J., Marr, C., Schroeder, T., Theis, F.J.: Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network. PLOS ONE **6**(8), e22649 (2011)
12. Mandon, H., Haar, S., Paulevé, L.: Temporal Reprogramming of Boolean Networks. In: Proc. 15th Conference on Computational Methods for Systems Biology. Lecture Notes in Computer Science, vol. 10545, pp. 179–195. Springer (2017)
13. Mizera, A., Pang, J., Qu, H., Yuan, Q.: Taming asynchrony for attractor detection in large Boolean networks. IEEE/ACM transactions on computational biology and bioinformatics **16**(1), 31–42 (2018)
14. Mizera, A., Pang, J., Su, C., Yuan, Q.: ASSA-PBN: A toolbox for probabilistic boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics **15**(4), 1203–1216 (2018)
15. Offermann, B., Knauer, S., Singh, A., Fernández-Cachón, M.L., Klose, M., Kowar, S., Busch, H., Boerries, M.: Boolean modeling reveals the necessity of transcriptional regulation for bistability in PC12 cell differentiation. Frontiers in Genetics **7**, 44 (2016)

16. Paul, S., Su, C., Pang, J., Mizera, A.: A decomposition-based approach towards the control of Boolean networks. In: Proc. 9th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics. pp. 11–20. ACM Press (2018)
17. Paul, S., Su, C., Pang, J., Mizera, A.: An efficient approach towards the source-target control of Boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics (2019), accepted
18. Remy, E., Rebouissou, S., Chaouiya, C., Zinovyev, A., Radvanyi, F., Calzone, L.: A modelling approach to explain mutually exclusive and co-occurring genetic alterations in bladder tumorigenesis. Cancer Research pp. canres–0602 (2015)
19. Ronquist, S., Patterson, G., Muir, L.A., Lindsly, S., Chen, H., Brown, M., Wicha, M.S., Bloch, A., Brockett, R., Rajapakse, I.: Algorithm for cellular reprogramming. Proceedings of the National Academy of Sciences **114**(45), 11832–11837 (2017)
20. 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 (2009)
21. Samaga, R., Von Kamp, A., Klamt, S.: Computing combinatorial intervention strategies and failure modes in signaling networks. Journal of Computational Biology **17**(1), 39–53 (2010)
22. del Sol, A., Buckley, N.J.: Concise review: A population shift view of cellular reprogramming. STEM CELLS **32**(6), 1367–1372 (2014)
23. Takahashi, K., Yamanaka, S.: A decade of transcription factor-mediated reprogramming to pluripotency. Nature Reviews Molecular Cell Biology **17**(3), 183193 (2016)
24. Zañudo, J.G.T., Albert, R.: Cell fate reprogramming by control of intracellular network dynamics. PLOS Computational Biology **11**, 1–24 (04 2015)