Enumeration and extension of non-equivalent deterministic update schedules in Boolean networks

Eduardo Palma[*,a], Lilian Salinas[a,b], Julio Aracena[c,b]

[a] *Departamento de Ingeniería Informática y Ciencias de la Computación, Universidad de Concepción, Edmundo Larenas 219, Piso 3, Concepción, Chile*
[b] *Centro de Investigación en Ingeniería Matemática, CI²MA, Casilla 160-C, Concepción, Chile.*
[c] *Departamento de Ingeniería Matemática, Universidad de Concepción, Av. Esteban Iturra s/n, Casilla 160-C, Concepción, Chile.*

Abstract

Boolean networks (BNs) were introduced by Stuart Kauffman in 1969 to model gene regulatory networks (GRNs). In the original model, the updating scheme was considered to be synchronous due to the difficulty of really knowing the order (if any) in which the nodes of a network update their state values. Since the dynamical behavior is very sensitive, in particular the attractors of the network, to changes in the updating scheme, it is increasingly common to use different updating rules in the modeling of GRNs to better capture an observed biological phenomenon and thus to obtain more realistic models.

In [4] equivalence classes of deterministic update schedules in BNs that yield exactly the same dynamical behavior of the network were defined according to the interaction digraph associated to the network. In this way, an upper bound of the number of different dynamical behaviors is the number of non-equivalent update schedules, which is usually much less than the total number of deterministic update schedules.

We present an efficient algorithm that allows to enumerate all non-equivalent deterministic update schedules for a given interaction digraph.

[☆]Partially supported by project Fondecyt 1131013 (J.A. and L.S.) and Master's scholarship CONICYT (E.P.).

[*]Corresponding author.

Email addresses: edupalma@udec.cl (Eduardo Palma), lilisalinas@udec.cl (Lilian Salinas), jaracena@ing-mat.udec.cl (Julio Aracena)

Also, this algorithm works in the case where there is a partial knowledge about the relative order of the updating of the node states.

An executable file of the Label algorithm made in Java is available at: www.inf.udec.cl/~lilian/UDE/

Key words: Gene regulatory networks, Boolean networks, Update schedule, Algorithms, Labeled digraph.

---

1. Introduction

Boolean networks (BNs) were introduced by Stuart Kauffman in [14] to model gene regulatory networks (GRNs). The gene expression level, in this case, is modeled by binary values, 0 or 1, indicating two transcriptional states, either active or inactive, respectively, and this level changes in time according to some local activation function which depends on the states of a set of nodes (genes). The dynamics of the network, is governed by an update schedule which determines when each node has to be updated.

In the original model, the updating scheme was considered to be synchronous, that is at each time step, the state of all nodes is updated at the same time. Numerous are the examples of GRNs modeled by Boolean networks with synchronous update schedule [1, 15, 16, 7, 25]. However, due to the synchronous scheme is considered not being very realistic many GRN modelers have used other schedule updates with different levels of asynchronism [26, 18, 10].

Several studies have shown that different updating rules may lead to distinct dynamical behaviors of a same network, specially to distinct attractor sets [6, 10, 4, 2]. Consequently, it is increasingly common to use different update schedules in the modeling of a GRN to better capture an observed biological phenomenon. In this regard, in the last years has been published examples of GRNs modeled by BNs with deterministic update schedules where the state of each node is updated once in every time step and in a given fixed order (see for example: [22, 11, 23, 24, 8, 19]). These update schedules can be seen as an extension of the parallel scheme where the node set of the network is partitioned into groups which are sequentially updated follow a given sequence and whose nodes are synchronously updated. This family of update schedules includes the sequential schedules (each group has size one), the parallel schedule (there is only one group) and the block-sequential

schedules. In the past, a lot of analytical work has been done about the dynamical behavior of BNs with this kind of scheme [21, 9, 13, 20, 5, 12].

In [4] equivalence classes of deterministic update schedules in BNs were defined according to the interaction digraph associated to the network, with labels on its arcs, which correspond to the relative order of the updating of the extreme nodes of these arcs. The authors showed in [4] that the elements of such equivalence classes yield exactly the same dynamical behavior of the network. Hence, the number of different dynamical behaviors of a BN is at most the maximum number of non-equivalent update schedules of the network, which is usually much less than the total number of deterministic update schedules [2]. In this way, it is important to have an efficient algorithm for enumerating all non-equivalent deterministic update schedules. Besides, the preserving of a certain dynamical property, as the limit cycles of a BN with different update schedules, could impose some labels on certain arcs of the interaction digraph. Thus, other important question is determining the set of non-equivalent update schedules which verify certain constraints on the order of some nodes to be updated. In this paper, we address both problems and construct efficient algorithms to solve them.

The algorithms designed use two strategies, the first one is avoid infeasible solutions using a polynomial algorithm. The second one is to make use of the structural characteristics of the digraph of interaction associated to a BN, as the presence of bridges, to divide the problem into subproblems, with smaller instances, which can be solved independently and whose solutions can be combined to determine the general solution. This procedure significantly reduces the total execution time of the main algorithm.

As example of application of the constructed algorithms we determined in few seconds the whole set of non-equivalent deterministic schemes and set of the schedules feasible of sharing the limit cycle of the synchronous BN constructed to model the mammalian cell cycle network exhibited in [10].

2. Definitions and notation

A *Boolean network* $N = (F, s)$ is defined by a finite set V of n elements; n state variables $x_v \in \{0,1\}$, $v \in V$; a function $F = (f_v)_{v \in V} : \{0,1\}^n \to \{0,1\}^n$ called *global activation function*, where its component functions f_v are called *local activation functions*, and an *update schedule* defined by a function $s : V \to \{1, \ldots, n\}$ such that $s(V) = \{1, \ldots, m\}$ for some $m \leq n$.

The update of value states of the Boolean network with an update function s is given by $x_v^{k+1} = f_v(x_u^{l_u} : u \in V)$, where $l_u = k$ if $s(v) \leq s(u)$ and $l_u = k+1$ if $s(v) > s(u)$.

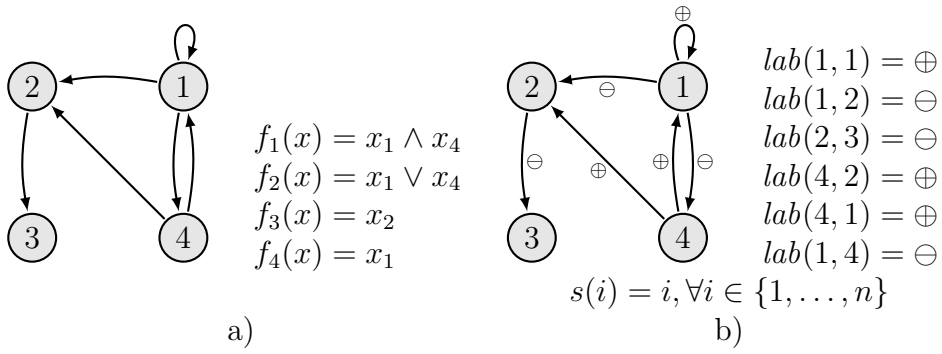

Figure 1: a) Interaction digraph associated to a Boolean network. b) Labeled digraph associated to a Boolean network and an update schedule.

Given a digraph G we will denote its set of vertices as V_G and its set of arcs as A_G.

The digraph associated to a function $F = (f_v)_{v \in V}$, called *interaction digraph*, is the directed graph G^F, where $V_{G^F} = V$ and $(u, v) \in A_{G^F}$ if and only if f_v depends on x_u, i.e., if there exists $x \in \{0,1\}^n$ such that $f_v(x) \neq f_v(\bar{x}^u)$, with $\bar{x}^u$ different of x only in position u. See an example of an interaction digraph in Figure 1a.

Given a digraph G a *label function* is any function $lab : A_G \to \{\oplus, \ominus, \bigcirc\}$. We call labeled digraph to (G, lab). In Figure 1b we see a labeled digraph in which its arcs are labeled according to its function lab. We denote $A^{\oplus}_{(G,lab)} = \{(u, v) \in A_G | lab(u, v) = \oplus\}$. Analogously we define $A^{\ominus}_{(G,lab)}$ and $A^{\bigcirc}_{(G,lab)}$.

Given a label function lab, we call *support* to the set $\mathrm{Sup}(lab) = A^{\oplus}_{(G,lab)} \cup A^{\ominus}_{(G,lab)}$. The arcs in the support are also called *labeled arcs*, remaining arcs of G will be called *unlabeled arcs*.

We say that (G, lab) is *fully labeled* if $\mathrm{Sup}(lab) = A_G$. Otherwise, we say that it is partially labeled.

The label function $\widetilde{lab} : A_G \to \{\oplus, \ominus, \bigcirc\}$ is an *extension* of $lab : A_G \to \{\oplus, \ominus, \bigcirc\}$ if $\forall a \in \mathrm{Sup}(lab), \widetilde{lab}(a) = lab(a)$. If $\mathrm{Sup}(\widetilde{lab}) = A_G$, we call $\widetilde{lab}$ a full extension.

Given a digraph G and a partial label function lab. If $lab(a) = \bigcirc$, we define the *simple extension* $lab^{a=\oplus}$, as the function where: $\forall e \in A_G \setminus \{a\}$,

$lab^{a=\oplus}(e) = lab(e)$ and $lab^{a=\oplus}(a) = \oplus$. Analogously, we define the simple extension $lab^{a=\ominus}$.

In [4] was defined the *update digraph* related to a network $N = (F, s)$ as (G^F, lab_s); where lab_s is the label function related to the scheme s, that is given by $lab_s(i,j) = \oplus$ if $s(i) \geq s(j)$ and $lab_s(i,j) = \ominus$ if $s(i) < s(j)$. In Figure 1b its shown an update digraph. We can see that $lab = lab_s$.

In this way, given any label function $lab : A_G \to \{\ominus, \oplus\}$, we say that the digraph (G, lab) is update if there exists s such that $(G, lab) = (G, lab_s)$, which can be found in polynomial time using for example the algorithm exhibited in [3] (see Appendix).

In this work we extend the concept of *update digraph* to partially labeled digraphs. We say that (G, lab) is an update digraph, if there exists a full extension lab' such that (G, lab') is update.

We denote as $\mathcal{S}(G, lab)$ the set of full extensions of lab' that make (G, lab') an update digraph.

Given a labeled digraph (G, lab), we define the *reverse digraph* (G_R, lab_R), where $V_{G_R} = V_G$,

$$A_{G_R} = \left\{(u,v) | (u,v) \in A^{\oplus}_{(G,lab)} \vee (v,u) \in A^{\ominus}_{(G,lab)}\right\}$$

and $lab_R(u,v) = \ominus$ if $(v,u) \in A^{\ominus}_{(G,lab)}$ and $lab_R(u,v) = \oplus$ otherwise. In Figure 2 we show an example of a labeled digraph and its associated reverse digraph.

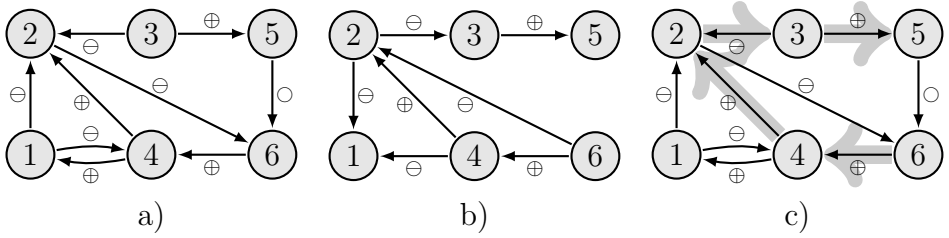

Figure 2: a) A labeled digraph (G, lab). b) The associated reverse digraph (G_R, lab_R). c) A negative reverse path: $(6, 4, 2, 3, 5)$.

Given a labeled digraph (G, lab) and G' a subdigraph of G, we define the *lab restricted to the subdigraph* G' as $lab|_{A_{G'}} : A_{G'} \to \{\oplus, \ominus, \bigcirc\}$, such that $\forall a \in A_{G'} : lab|_{A_{G'}}(a) = lab(a)$.

Given a labeled digraph (G, lab), there is a *reverse path* from node v_1 to v_n in G, if there exists a sequence of nodes $(v_1, v_2, \ldots, v_n)$ which verifies:

$$\forall i \in \{1, \ldots, n-1\}, (v_i, v_{i+1}) \in A^{\oplus}_{(G,lab)} \vee (v_{i+1}, v_i) \in A^{\ominus}_{(G,lab)}$$

In other words, there exists a path in the reverse digraph.

There is a *negative reverse path* from v_1 to v_n, if there exists a path in the reverse digraph that contains a negative arc. In Figure 2c the path marked by the gray arrows is a reverse path from 6 to 5. This is also a negative reverse path, because the arc $(3, 2)$ is on the sequence.

A *forbidden cycle* (defined in [3]) is a negative reverse path $(v_1, v_2, \ldots, v_n)$ where $v_1 = v_n$.

The previous concepts will lead us to define binary relations $\mathrm{RP} \subseteq V_G \times V_G$ and $\mathrm{NRP} \subseteq V_G \times V_G$, respectively reverse path and negative reverse path. To simplify the notation we denote $\mathrm{RP}(i, j) \iff (i, j) \in \mathrm{RP}$ and $\mathrm{NRP}(i, j) \iff (i, j) \in \mathrm{NRP}$.

In labeled digraphs, we define *positive strongly connected component* in the same way as the strongly connected component, but restricting to the digraph induced by positive arcs.

3. Complexity of update digraph extension problem

In this article we are interested in finding the extensions of a partial labeled digraph (eventually with empty support) that are update digraphs or equivalently a set of non-equivalent deterministic update schedules satisfying some constraints about the relative order of updating of some nodes of the network. More precisely, we address the following problem:

Update digraph extension (UDE): Given a labeled digraph (G, lab), to find the set $\mathcal{S}(G, lab)$ of all full extensions lab' of lab such that (G, lab') is an update digraph.

To know the computational complexity of the UDE problem, we study the following counting problem associated to UDE:

Counting update digraph extensions (CUDE): Given (G, lab) a labeled digraph, to determine the number of all full extensions lab' of lab such that (G, lab') is an update digraph.

We will prove that CUDE is a difficult problem, thereby we can conclude the complexity of the UDE problem.

Theorem 1. *CUDE is #P-complete.*

The proof (see Appendix) is based in the idea that an acyclic labeled digraph is an update digraph if and only if its reverse digraph is acyclic. This is because in the reverse digraph of an update digraph the only allowed cycles have every arc labeled as positive. In this way, it is easy to define a bijection between an update digraph and the acyclic orientation of its underlaying graph.

Note that previous result tell us that to know the total number of extensions from a partially labeled digraph is a hard problem, while that the related existence problem is known to be polynomial [3].

4. Algorithms

In this section we present the theoretical results that leads to design an algorithm that solve the UDE problem. In first place, we focus on verify the existence of one solution, then we reduce our problem contracting each positive strongly connected component in one vertex. In second place, we present the two main results of this article: they are the effect of forcing arcs, that allows to eliminate infeasible solutions in polynomial time, and the division of our problem into smaller pieces using algorithms to find bridges and strongly connected components.

4.1. Verify

First, we verify whether the labeled digraph is an update digraph. To check this, we use the ReversePaths algorithm to search any forbidden cycle. ReversePaths algorithm is an adaptation of Floyd-Warshall algorithm, where instead of finding minimum weight paths, we determine the existence of reverse paths and negative reverse paths between each pair of vertices. The algorithm returns the matrix M where: $M(u, v) = \infty$ if it does not exist a reverse path from u to v, $M(u, v) = -1$ if there is a negative reverse path from u to v and $M(u, v) = 1$ otherwise. See details of ReversePaths and Verify algorithms in Appendix.

4.2. Reduction

As we mentioned above, the UDE problem belongs to a class of problems for which there are not known polynomial algorithms that solve them. Hence, the reduction of instance of the problem is very important. In this way, we

define the reduced digraph of an update digraph which involves replacing each positive strongly connected by a single vertex.

The following lemma is a property of the update digraphs which allows to define correctly the reduced digraph of an update digraph.

Lemma 2. *Let (G, lab) be an update digraph, G_1 and G_2 two positive strongly connected components of G, and $\widetilde{lab}$ a full extension such that $(G, \widetilde{lab})$ is a fully labeled update digraph. Then $\forall a, a' \in A_G \cap (V_{G_1} \times V_{G_2})$: $\widetilde{lab}(a) = \widetilde{lab}(a')$.*

The proof of this lemma (detailed in Appendix) uses the fact that if $\widetilde{lab}(a) \neq \widetilde{lab}(a')$ then there exists a forbidden cycle in the labeled digraph.

From the previous lemma we know that we can preliminarily label some arcs. This help us to avoid a multidigraph when we obtain the reduced digraph or problems in its label function.

Definition 1. *Let (G, lab) be an update digraph and $\{G_1, \ldots, G_k\}$ its positive strongly connected components. We define its reduced labeled digraph $R(G, lab)$ by $R(G, lab) = (G_{rd} = (V_{rd}, A_{rd}), lab_{rd})$, where $V_{rd} = \{v_1, \ldots, v_k\}$ and $A_{rd} = \left\{(v_i, v_j) | \exists (u, v) \in A_G \cap \left(V_{G_i} \times V_{G_j}\right)\right\}$*

Furthermore, $lab_{rd}(v_i, v_j) = lab(u, v)$, if $\exists (u, v) \in \left(V_{G_i} \times V_{G_j}\right) \cap \mathrm{Sup}(lab)$ and $lab_{rd}(v_i, v_j) = \bigcirc$ otherwise. We say that a labeled digraph (G, lab) is reduced if $(G, lab) = R(G, lab)$.

Note that if (G, lab) is connected, then obviously $R(G, lab)$ is also connected. Furthermore, as (G, lab) is an update digraph, then $R(G, lab)$ is an update digraph, since otherwise there would be a forbidden cycle.

Example 1. *In Figure 3 an example of a reduced digraph is shown. The nodes 4, 5 and 6 are in a positive strongly connected component, so in the reduced digraph they are all represented by node v_4.*

Theorem 3. *The elements of the solution set of the UDE problem for (G, lab) are in bijection with those of the UDE problem for $R(G, lab)$.*

The proof of this theorem uses the previous lemma (details in Appendix). In fact, if we have an unlabeled arc between nodes in the same positive strongly connected component, this must be labeled positive to avoid a forbidden cycle. In terms of update schedule, that means that every node in the positive

strongly connected component is updated at the same time, so we can represent all these nodes in one. Also, the arcs between different positive strongly connected components must have the same direction in the reverse digraph to avoid forbidden cycles, hence we can represent all of them by just one that has the right direction in the reverse digraph.

The application of this results leads to Algorithm 1. In this algorithm we use SCC^+, i.e. the algorithm that returns the positive strongly connected components of a digraph. This is very easy to construct using, for example, Tarjan algorithm.

Algorithm 1 Reduce

Require: An update digraph (G, lab).
Ensure: The reduced digraph $(G_{\mathrm{rd}}, lab_{\mathrm{rd}})$.

$\{G_1, \ldots, G_k\} \leftarrow \mathsf{SCC}^+(G, lab)$
$V_{G_{\mathrm{rd}}} \leftarrow \{w_1, \ldots, w_k\}$
$A_{G_{\mathrm{rd}}} \leftarrow \emptyset$
for $i = 1$ **to** k **do**
    for $j = 1$ **to** k **do**
        if $\exists (u, v) \in A_G$ with $u \in V_{G_i}$ and $v \in V_{G_j}$ **then**
            $A_{G_{\mathrm{rd}}} \leftarrow A_{G_{\mathrm{rd}}} \cup (w_i, w_j)$
for all do$(w_i, w_j) \in A_{G_{\mathrm{rd}}}$
    if $\exists u \in V_{G_i}, v \in V_{G_j}$ and $lab(u, v) = \ominus$ **then**
        $lab_{\mathrm{rd}}(u, v) \leftarrow \ominus$
    else if $\exists u \in V_{G_i}, v \in V_{G_j}$ and $lab(u, v) = \oplus$ **then**
        $lab_{\mathrm{rd}}(u, v) \leftarrow \oplus$
    else
        $lab_{\mathrm{rd}}(u, v) \leftarrow \bigcirc$
return $(G_{\mathrm{rd}}, lab_{\mathrm{rd}})$

4.3. Force

Given an update digraph (G, lab) with $\mathrm{Sup}(lab) \neq A_G$, there are situations in which an unlabeled arc $(i, j) \in A_G$ may be labeled just in one way to keep the update digraph property. In fact, if every unlabeled arc is forced to have a unique label the solutions of the UDE problem is unique.

Example 2. *In figure 4 a) we see that there exists a negative reverse path from 3 to 2 (3,1,2) and in b) we see that there exists a reverse path from*

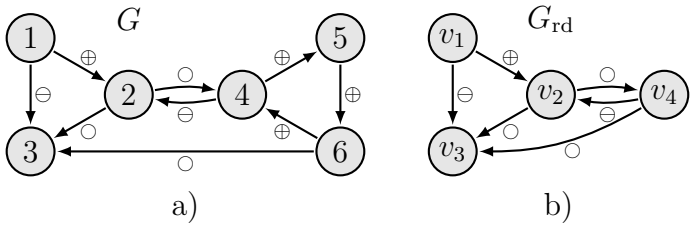

Figure 3: a) An update digraph. b) The reduced digraph.

2 to 4, then the unlabeled arcs (2,3) and (2,4) must be labeled negative and positive respectively as shown in c).

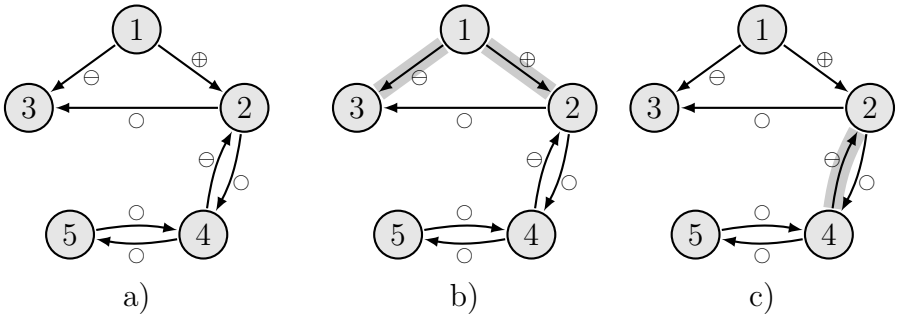

Figure 4: Example of forced arcs.

The fact that there exists an unlabeled arc forced to have an unique label, called simply *forced arc*, depends only on the existence of reverse and negative reverse paths in the interaction digraph as shown in the following proposition.

Proposition 4 (Forced arc). *Let (G, lab) be an update digraph, $(i,j) \in A_G$ and $lab(i,j) = \bigcirc$, then:*

1. $\forall (i,j) \in A_G : \text{NRP}(j,i) \iff \left(G, lab^{(i,j)=\oplus}\right)$ *is a non update digraph; then we say that the arc (i,j) is forced to be negative.*
2. $\forall (i,j) \in A_G : \text{RP}(i,j) \iff \left(G, lab^{(i,j)=\ominus}\right)$ *is a non update digraph; then we say that the arc (i,j) is forced to be positive.*

Proof. We will proof the first case, the second one is analogous.

($\Longrightarrow$) If we consider that there exists a negative reverse path from j to i, then if we label (i,j) positive, there will be a negative reverse path (forbidden cycle) from j to j. Hence, $\left(G, lab^{(i,j)=\oplus}\right)$ is a non update digraph.

($\Longleftarrow$) If $\left(G, lab^{(i,j)=\oplus}\right)$ is a non update digraph and (G, lab) is, then a forbidden cycle is produced by labeling positive the arc (i,j). Hence there exists a negative reverse path from j to i in (G, lab).

It is important to observe that the order in which the forced arcs are chosen to be labeled is irrelevant in the label obtained. This is because of the labeling of forced arcs do not give us additional information in terms of reverse and negative reverse paths in the interaction digraph. Indeed, if there exists a negative reverse path from j to i we label this negative forced arc (i,j), i.e, we add a new negative reverse path from j to i. Analogously, if there is a reverse path from i to j in the interaction digraph, we label (i,j) positive, i.e. we add a new reverse path from i to j.

To check the existence of forced arcs allows to avoid extensions that are not update schedules. Hence, we build Algorithm 2 that labels all the forced arcs. Applying this algorithm to an update digraph we obtain the *maximal extension*, which is the extension of G such that every forced arc is labeled.

Algorithm 2 Force

Require: An update digraph (G, lab).
Ensure: A maximal extension of lab

```
1: lab~ ← lab
2: M ← ReversePaths(G, lab)
3: for all a ∈ Sup(lab), with a = (i, j) do
4:     if M(i, j) ≠ ∞ then
5:         lab~ ← lab~^{a=⊕}
6:     else if M(j, i) = −1 then
7:         lab~ ← lab~^{a=⊖}
8: return lab~
```

Next, we introduce a simple algorithm, called SimpleLabel, to find all the extensions of a given partially labeled update digraph and which uses the Force algorithm. Firstly, this algorithm finds the maximal extension of the given label function. Thereafter, the algorithm labels an unlabeled arc positive and and recursively calls itself. In this way, it finds all the solutions

with this arc labeled positive, then it repeats the procedure labeling the arc negative. Finally the total solution is the union of both solution sets.

Algorithm 3 SimpleLabel

Require: An update digraph (G, lab)
Ensure: The set $\mathcal{S}(G, lab)$ denoted by S and its cardinal number $r := |S|$

```
(S, r) ← (∅, 0)
lab ← Force(G, lab)
if Sup(lab) = A(G) then
    return (lab, 1)
else
    Let be a ∈ A(G) \ Sup(lab)
    (S_1, r_1) ← SimpleLabel(G, lab^{a=⊕})
    (S_2, r_2) ← SimpleLabel(G, lab^{a=⊖})
    (S, r) ← (S_1 ∪ S_2, r_1 + r_2)
return (S, r)
```

4.4. Divide and conquer

In order to improve the efficiency of the SimpleLabel algorithm, we use structural properties of the digraph to divide the problem into subproblems, by partitioning the arc set, such that their combined solutions give us the solution of the original problem. To formalize this combination of solutions we define operator $\otimes$.

Definition 2. *Given a digraph G; $\{A_i\}_{i=1}^k$ a partition of A_G and $\{A_i\}_{i=1}^k$ a family of label functions such that for every $i \in \{1, \ldots, \}$, $L_i \subseteq \{lab | lab : A_i \to \{\oplus, \ominus, \bigcirc\}$ is label function of $G\}$, we define:*

$$L_1 \otimes \cdots \otimes L_k = \{lab : A_G \to \{\oplus, \ominus, \bigcirc\} \,|\exists lab_i \in L_i, lab|_{A_i} = lab_i\}$$

The following result is directly obtained from the previous definition.

Proposition 5. *Let (G, lab) be a digraph such that its connected components are $G_1, \ldots, G_k$, then*

$$\mathcal{S}(G, lab) = \mathcal{S}(G_1, lab|_{A_{G_1}}) \otimes \cdots \otimes \mathcal{S}(G_k, lab|_{A_{G_k}})$$

4.4.1. Division by bridges

The first division is to separate nodes joined by a bridge in the underlying graph (i.e. the graph obtained by replacing all directed edges of G with undirected edges). This idea comes from the fact that any forbidden cycle cannot contain any bridge.

Proposition 6. *Let (G, lab) be a labeled connected digraph, G_U the underlying graph of G the and uv a bridge of G_U that divides G into G_1 and G_2, we denote by $G_b = G[\{u, v\}]$ then*

$$\mathcal{S}(G, lab) = \mathcal{S}(G_1, lab|_{A_{G_1}}) \otimes \mathcal{S}(G_b, lab|_{A_{G_b}}) \otimes \mathcal{S}(G_2, lab|_{A_{G_2}}).$$

The proof of this proposition is detailed in Appendix. In Figure 5 we can see an example of division by bridges of a labeled digraph with the associated partition of arc set.

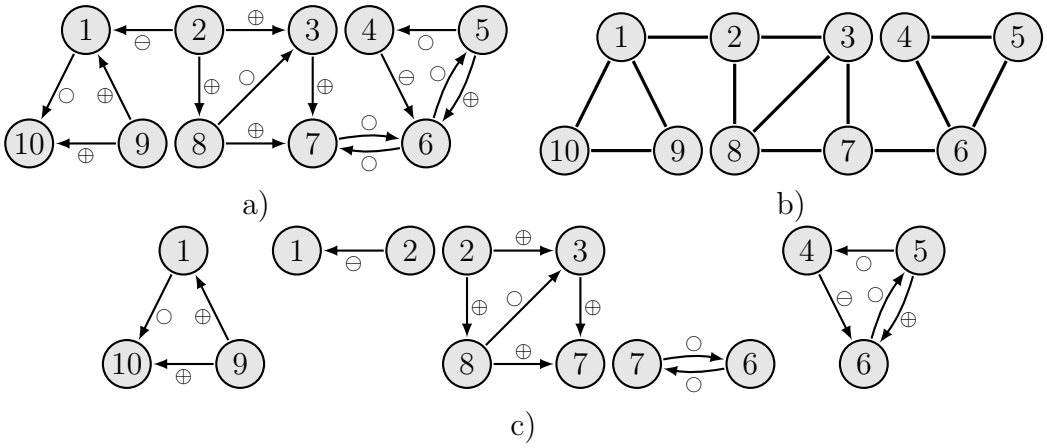

Figure 5: a) Labeled digraph. b) Underlying graph. c) Partition of arc set produced by the division by bridges.

4.4.2. Division by strongly connected components

Another way to simplify the digraph consists in divide it by its strongly connected components in the extended reverse digraph, i.e. the reverse digraph where unlabeled arcs are replaced by unlabeled arcs in both directions. As in the case of bridges, forbidden cycles cannot use arcs connecting different strongly connected components.

Proposition 7. *Let (G, lab) be an update digraph with $G_1, \ldots, G_k$ its strongly connected components of the extended reverse digraph, we define the set of arcs:*

$$A_T = \bigcup_{i,j} A_G \cap \left(V_{G_i} \times V_{G_j}\right),$$

which it is composed by the arcs between strongly connected components of the extended reverse digraph of (G, lab). Then,

$$\mathcal{S}(G, lab) = \mathcal{S}(\widetilde{G}_1, lab|_{A_{\widetilde{G}_1}}) \otimes \cdots \otimes \mathcal{S}(\widetilde{G}_k, lab|_{A_{\widetilde{G}_k}}) \otimes \{lab|_{A_T}\}$$

where $\forall i \in \{1, \ldots, k\}$, $\widetilde{G}_i = G[V_{G_i}]$

The proof of this last proposition is similar to that of the Proposition 6 shown in Appendix, and is based on the fact that (G, lab) is an update digraph if and only if every labeled digraph $(\widetilde{G}_i, lab|_{A_{\widetilde{G}_i}})$ is update digraph.

Next we define the Algorithm 4, called Label, that requires as input an update digraph with a label function without forced arcs, and which uses the divisions defined above to partition our problem. It applies the sames ideas of SimpleLabel, i.e. to force arcs, label one and apply recursively the same algorithm.

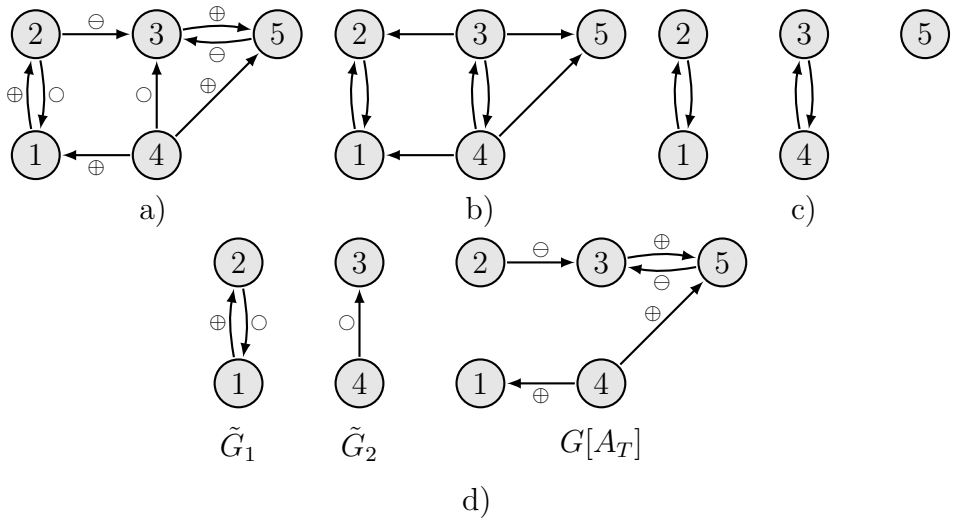

Figure 6: a) Labeled digraph. b) Extended reverse digraph. c) Strongly connected components. d) Partition of arc set produced by the division by strongly connected components.

Algorithm 4 Label

Require: (G, lab), a maximal extension of an update digraph.
Ensure: The set $\mathcal{S}(G, lab)$ denote by S and its cardinal number $r := |S|$

$(S, r) \leftarrow (\emptyset, 1)$
$\{(G_1, lab_{G_1}), \ldots, (G_k, lab_{G_k})\} \leftarrow \mathsf{Bridges}(G_U)$
for $i = 1$ to k **do**
  if $\mathrm{Sup}(lab_{G_i}) = A_{G_i}$ **then**
    $S \leftarrow S \otimes \{lab_{G_i}\}$
  else
    $\{(H_1, lab_{H_1}), \ldots, (H_p, lab_{H_p})\} \leftarrow \mathsf{SCC}^+(G_i, lab_{G_i})$
    if $p = 1$ **then**
      Let $a \in A_{H_1}$ be an arc such that $lab(a) = \bigcirc$
      $lab^+ \leftarrow \mathsf{Force}\left(H_1, lab_{H_1}^{a=\oplus}\right)$
      if $\mathrm{Sup}(lab^+) \neq A_{H_1}$ **then**
        $(\hat{S}, \hat{r}) \leftarrow \mathsf{Label}(H_1, lab^+)$
      else
        $(\hat{S}, \hat{r}) \leftarrow (\{lab^+\}, 1)$
      $lab^- \leftarrow \mathsf{Force}\left(H_1, lab_{H_1}^{a=\ominus}\right)$
      if $\mathrm{Sup}(lab^-) \neq A_{H_1}$ **then**
        $(\tilde{S}, \tilde{r}) \leftarrow \mathsf{Label}(H_1, lab^-)$
      else
        $(\tilde{S}, \tilde{r}) \leftarrow (\{lab^-\}, 1)$
      $(S, r) \leftarrow (S \otimes (\hat{S} \cup \tilde{S}), r \cdot (\hat{r} + \tilde{r}))$
    else
      for $j = 1$ to p **do**
        if $\mathrm{Sup}(lab_{H_j}) = A_{H_j}$ **then**
          $S \leftarrow S \otimes \{lab_{H_j}\}$
        else
          $(\tilde{S}, \tilde{r}) \leftarrow \mathsf{Label}(H_j, lab_{H_j})$
          $(S, r) \leftarrow (S \otimes \tilde{S}, r \cdot \tilde{r})$
return (S, r)

Finally, we present the main algorithm to solve the UDE problem, named UpdateLabel, which first checks if the labeled digraph (G, lab) received as input is an update digraph, i.e. if $\mathcal{S}(G, lab) \neq \emptyset$.

5. Comparison of the algorithm with and without division

To illustrate the efficiency of doing divisions in the main algorithm we compare the performance of our algorithm Label against SimpleLabel algorithm that only uses the idea of forced arcs. As the algorithms require an update digraph as input, we use the main algorithm (UpdateLabel algorithm) to call Label or SimpleLabel. The tests were run in complete digraphs and chains (see Figure 7) with empty support in a laptop with Processor: Intel Core i5-3317U, CPU: 1.7GHz, RAM memory: 6 GB, operating system: Windows 8.1 (64 bits). In Table 1 we can see as the main algorithm with Label runs faster than with SimpleLabel, even in the case of complete digraphs where there is a small number of divisions.

Algorithm 5 UpdateLabel

Require: A labeled digraph (G, lab)

Ensure: The set $\mathcal{S}(G, lab)$ denote by S and its cardinal number $r := |\mathcal{S}(G, lab)|$.

1: **if** Verify$(G, lab) =$ false **then**
2: **return** $(\emptyset, 0)$
3: **else**
4: $(G_{\mathrm{rd}}, lab_{\mathrm{rd}}) \leftarrow$ Reduce(G, lab)
5: $\widetilde{lab} \leftarrow$ Force$(G_{\mathrm{rd}}, lab_{\mathrm{rd}})$
6: **return** Label$(G_{\mathrm{rd}}, \widetilde{lab})$ (or SimpleLabel$(G_{\mathrm{rd}}, \widetilde{lab})$)

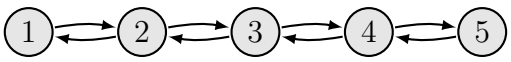

Figure 7: Example of a chain P_n where $n = 5$.

6. Application to Mammalian cell cycle network

In this section we use the implementation of the main algorithm introduced in the previous section over the Mammalian cell cycle network constructed in [10] and studied with different deterministic update schedules in [23], where some theoretical results were given. Then we analyze the obtained results showing the main advantages of this algorithm.

Graph	Nodes	Arcs	$\lvert\mathcal{S}(G, lab)\rvert$	SimpleLabel	Label
K_3	3	6	13	0.02	0.02
K_5	5	20	541	0.18	0.13
K_7	7	42	47 293	140.76	0.79
K_8	8	56	545 835	$> 22\,200.00$	1.90
P_{10}	10	18	19 683	4.43	0.02
P_{12}	12	22	177 147	313.68	0.02
P_{50}	50	98	3^{49}	–	0.09
P_{200}	200	398	3^{199}	–	0.17

Table 1: Results obtained when the main algorithm is used with and without divisions on some digraphs. Here the time is measured in seconds.

The Mammalian cell cycle network has 10 nodes and 31 arcs. A detailed description of the network can be found in Table 1 in Appendix. In Figure 8 we see the interaction digraph of the network.

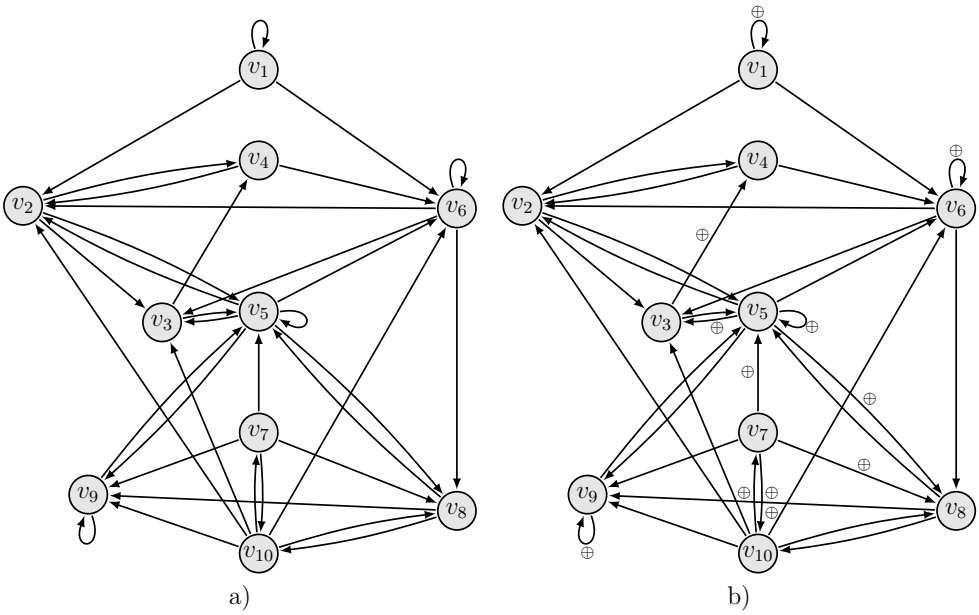

Figure 8: a) The interaction digraph of the Mammalian cell cycle network. b) Mammalian cell cycle network with some labeled arcs (G_M, lab_M).

We apply our main algorithm to the unlabeled digraph $(G_M, lab_{\bigcirc})$, where $\forall a \in A_{G_M}, lab_{\bigcirc}(a) = \bigcirc$, and we find all the full extensions, i.e. all non-

equivalent updates schedules, in 27.61 seconds. There exists a total of 466712 elements in $\mathcal{S}(G_M, lab_{\bigcirc})$.

This network synchronously updated, denoted $N^p = (F, s^p)$ has only one fixed point and one limit cycle of length 7 as attractors:

Fixed Point: $x^0 = (0,1,0,0,0,1,0,1,0,0)$

Limit Cycle: $C = \{x^0, x^1, x^2, x^3, x^4, x^5, x^6, x^7\}$, where:

$$\begin{array}{ll} x^0 = (1,0,1,0,0,0,0,1,1,0) & x^4 = (1,0,0,0,1,0,0,0,1,1) \\ x^1 = (1,0,1,1,0,0,0,1,0,0) & x^5 = (1,0,0,0,1,0,1,0,1,1) \\ x^2 = (1,0,1,1,1,0,0,1,0,0) & x^6 = (1,0,0,0,0,0,1,1,1,0) \\ x^3 = (1,0,0,1,1,0,0,0,0,0) & x^7 = (1,0,1,0,0,0,0,1,1,0) \end{array}$$

In our study we are interested in finding all the non-equivalent deterministic update schedules the yield the limit cycle C. This give us some restrictions over the label function, so we can easily label some arcs to reduce the space of search.

Example 3. *In local activation function f_{v_7} we know that when x_{v_7} is 1, $x_{v_{10}}$ is 1. Observing the limit cycle, we see that $x^6_{v_7}$ is 1 but $x^6_{v_{10}}$ is 0, so if $lab(10,7) = \ominus$ the cycle is not possible, because it means that $s(v_{10}) < s(v_7)$ ($x_{v_{10}}$ is updated before x_{v_7}).*

Similarly we can check $f_{v_{10}}$. In this case the function depends on v_7 and v_8. Specifically when $x_{v_{10}}$ is 1, both x_{v_7} and x_{v_8} must be 0. So in row sixth, we conclude that $lab(7,10) = \oplus$; but we cannot label $(8,10)$ using the same idea.

Finally is easy to see that we can label (5,3); (3,4); (7,5); (10,7); (5,8); (7,8); (8,9) and (7,10) are positive arcs if we are looking for label functions that preserve the mentioned cycle.

The label digraph obtained with the previous information is shown in Figure 8b).

Using the main algorithm with input this partially label digraph we find in just 0.40 seconds all the 1440 elements of $\mathcal{S}(G_M, lab_M)$. In [23] they studied the update digraphs that preserve the limit cycle C, but they considered each member of $\mathcal{S}(G_M, lab_{\bigcirc})$ as a possible candidate, i.e. the total of 466712 elements.

Considering the simplicity of get lab_M from $lab_{\bigcirc}$ and the great reduction in both time of execution and total solutions, we can realize the advantages taken from the use of our algorithm.

7. Conclusions

In this article we addressed the problem of enumerating all non-equivalent deterministic update schedules for a given interaction digraph. We construct an algorithm that uses two major ideas in its design. The first one is the base on the Force algorithm, which in polynomial time checks whether the given labels on some arcs uniquely determine the label in others (only possible extension). This allows to eliminate infeasible solutions in polynomial time.

The second one is to make use of the structural characteristics of the interaction digraph associated to a BN, as the presence of bridges, to divide the problem into subproblems, with smaller instances, which can be solved independently and whose solutions can be combined to determine the general solution. This idea is applied to strongly connected components in a digraph induced by the labeled. In this case we have obtain a relative order between set of nodes, that allow to find the solution as a combinations of the solution of the parts. This procedure significantly reduces the total execution time of the main algorithm as observed in Table 1.

A. Appendix

A.1. Complexity

Theorem 8. *CUDE is #P-complete.*

Proof. We reduce the Acyclic Orientation problem (AO problem) to CUDE. The AO problem consists in given a graph G, count the number of orientations of G that does not contain any cycle. It is known that the Acyclic Orientation problem is #P-complete [17].

Let G be a graph with $V_G = \{1, \ldots, n\}$, we define G' (an acyclic orientation of G) with $V_{G'} = V_G$ and $A_{G'} = \{(u, v) | uv \in E_G \wedge u > v\}$. Obviously we can construct G' in polynomial time.

We denote the sets:

$$O_G = \{G_0 | G_0 \text{ an acyclic orientation of } G\}, \text{ and}$$
$$D_{G'} = \{lab | (G', lab) \text{ is a fully labeled update digraph}\}$$

We define the function $\phi : O_G \to D_{G'}$, such that $\forall G_0 \in O_G$, $\phi(G_0) = lab_{G_0}$, where $\forall (u, v) \in A_{G'}$, $lab_{G_0}(u, v) = \oplus$ if $(u, v) \in A_{G_0}$ and $lab_{G_0}(u, v) = \ominus$ if $(v, u) \in A_{G_0}$.

Since G_0 and G' are orientations of G, then $\forall (u,v) \in A_{G'}$, $(u,v) \in A_{G_0}$ or $(v,u) \in A_{G_0}$. Thus, lab_{G_0} is a full label of G'. Besides, because G is an acyclic orientation, then (G, lab_{G_0}) is an update digraph. Hence, $lab_{G_0} \in D_{G'}$, i.e. lab_{G_0} is well defined.

Furthermore, ϕ is injective. Indeed, let $G_1 = (V, A_1)$ and $G_2 = (V, A_2)$ be two acyclic orientations of G:

$$\begin{aligned}
\phi(G_1) = \phi(G_2) &\Rightarrow lab_{G_1} = lab_{G_2} \\
&\Rightarrow \forall (u,v) \in A_{G'} : lab_{G_1}(u,v) = lab_{G_2}(u,v) \\
&\Rightarrow \ \forall (u,v) \in A_{G'} : ((u,v) \in A_1 \wedge (u,v) \in A_2) \\
&\qquad\qquad \vee ((v,u) \in A_1 \wedge (v,u) \in A_2) \\
&\Rightarrow A_1 = A_2 \Rightarrow G_1 = G_2
\end{aligned}$$

Function ϕ is also surjective. For all lab such that (G', lab) is an update digraph, we have that ${G'}_R$ (reverse digraph of G') is an acyclic orientation of G. In fact, it does not have cycles, since otherwise G' would have a forbidden cycle or a positive component. The first option is not possible, because it is an update digraph and the second one is also impossible because G' would have a cycle. Therefore ϕ is surjective and hence bijective.

A.2. Verify

The Verify algorithm checks whether the labeled digraph is an update digraph for searching forbidden cycles. For this, we use Algorithm 7, called ReversePaths which is an adaptation from Floyd-Warshall algorithm, where instead of finding minimum weight paths, we determine the existence of reverse paths and negative reverse paths between each pair of vertices.

In ReversePaths we define the function $M(u,v) = \infty$ if it does not exist a reverse path from u to v, $M(u,v) = -1$ if there is a negative reverse path from u to v and $M(u,v) = 1$ otherwise. And also the commutative binary operator $\odot$ where:

$$a \odot b = \begin{cases} \infty & \text{if } a = \infty \vee b = \infty \\ 1 & \text{if } a = b = 1 \\ -1 & \text{otherwise.} \end{cases}$$

Algorithm 6 Verify

Require: A digraph $G = (V, A)$ and a label function $lab : A \to \{\oplus, \ominus, \bigcirc\}$.
Ensure: *true* if (G, lab) is an update digraph, and *false* otherwise.

for all $u \in V(G)$ **do**
  if (there exists a negative reverse path from u to u) **then**
    return false
return true

Algorithm 7 ReversePaths

Require: A labeled digraph (G, lab)
Ensure: The matrix M of all reverse and negative paths of (G, lab)

Let M be a matrix $|V_G| \times |V_G|$
for all $(u, v) \in V_G \times V_(G)$ **do**
  if $(v, u) \in A^{\ominus}_{(G,lab)}$ **then**
    $M(u, v) \leftarrow -1$
  else if $(u, v) \in A^{\oplus}_{(G,lab)}$ **then**
    $M(u, v) \leftarrow 1$
  else
    $M(u, v) \leftarrow \infty$
for $k = 1$ **to** $|V_G|$ **do**
  for $i = 1$ **to** $|V_G|$ **do**
    for $j = 1$ **to** $|V_G|$ **do**
      if $M(i, k) \odot M(k, j) < M(i, j)$ **then**
        $M(i, j) \leftarrow M(i, k) \odot M(k, j)$
return M

A.3. Reduction

Lemma 9. *Let (G, lab) be an update digraph, G_1 and G_2 two positive strongly connected components of G, and $\widetilde{lab}$ a full extension such that $(G, \widetilde{lab})$ is a fully labeled update digraph. Then $\forall a, a' \in A_G \cap (V_{G_1} \times V_{G_2})$: $\widetilde{lab}(a) = \widetilde{lab}(a')$.*

Proof. Let $(G, \widetilde{lab})$ be a fully labeled update digraph, with G_1 and G_2 two positive strongly connected components of G; and (u, v), $(x, y) \in A_G \cap (V_{G_1} \times V_{G_2})$. Let us assume that $\widetilde{lab}(u, v) = \oplus$ and $\widetilde{lab}(x, y) = \ominus$. Thus,

we have that $\mathrm{RP}(u,v)$ and $\mathrm{NRP}(y,x)$. Furthermore, $\mathrm{RP}(v,y)$ and $\mathrm{RP}(x,u)$, because $u,x \in V_{G_1}$ and $v,y \in V_{G_2}$. Then, we deduce from the properties of both relations that $\mathrm{NRP}(u,u)$, i.e., there exists a forbidden cycle. This lead us to a contradiction, because $(G,\widetilde{lab})$ is an update digraph. Hence, we have that $\widetilde{lab}(u,v) = \widetilde{lab}(x,y)$.

Theorem 10. *The elements of the solution set of the UDE problem for (G, lab) are in bijection with those of the UDE problem for $R(G, lab)$.*

Proof. Let $\phi : \mathcal{S}(G,lab) \to \mathcal{S}(R(G,lab))$, be a function defined by: $\forall lab' \in \mathcal{S}(G,lab), \phi(lab') = lab''$, such that $\forall (v_i,v_j) \in A_{\mathrm{rd}},\ lab''(v_i,v_j) = lab'(u,v)$; where $u \in V_{G_i}$ and $v \in V_{G_j}$.

Now, we proof that (G_{rd}, lab'') is an update digraph.

Let us suppose that there exists a forbidden cycle in (G_{rd}, lab''), then there exists a sequence $(u_1,\ldots,u_l)$ such that:

$$\forall i \in \{1,\ldots,l-1\}, (u_i,u_{i+1}) \in A^{\oplus}_{(G_{\mathrm{rd}},lab'')} \vee (u_{i+1},u_i) \in A^{\ominus}_{(G_{\mathrm{rd}},lab'')}$$

Without loss of generality, we say that u_i represents the positive strongly connected component G_i. Then, by definition of $A_{G_{\mathrm{rd}}}$ we know that given $(u_j,u_k),(u_k,u_h) \in A_{G_{\mathrm{rd}}}$ there exists $(v_1,v_2),(v_3,v_4) \in A_G$ with $v_2,v_3 \in G_k$, $v_1 \in G_j$, $v_4 \in G_h$. Hence, we know that given a reverse path or a negative reverse path in G_{rd}, there exists one in G. Then, if there exists a forbidden cycle in (G_{rd}, lab'') there will be one in (G, lab'). Therefore ϕ is well defined.

Furthermore ϕ is injective. Indeed, given $lab_1, lab_2 \in \mathcal{S}(G,lab)$, $\phi(lab_1) = lab_1''$, $\phi(lab_2) = lab_2''$ and $\{G_1,\ldots,G_k\}$ the positive strongly connected components of (G,lab) then $lab_1 \neq lab_2$ implies there exists $(u,v) \in A_G$, such that $lab_1(u,v) \neq lab_2''(u,v)$, where $u \in V_{G_i}$, $v \in V_{G_j}$, and $i \neq j$. Then, $lab_1''(w_i,w_j) \neq lab_2''(w_i,w_j)$.

We also have that ϕ is surjective. Given $lab'' \in \mathcal{S}(R(G,lab))$, we define the function lab' by:

$\forall (u,v) \in A_G$; $\forall i,j \in \{1,\ldots,k\}$, $i \neq j$, with $u \in G_i, v \in G_j$: $lab'(u,v) = lab''(v_i,v_j)$ and $\forall (u,v) \in A_G$; $\forall i \in \{1,\ldots,k\}$, with $u,v \in G_i$: $lab'(u,v) = \oplus$.

It is easy to see that $lab' \in \mathcal{S}(G,lab)$ and $\phi(lab') = lab''$

A.4. Force

Proposition 11. *Let (G, lab) be an update digraph and $a \in A_G$ a positive forced arc, then:*

1. $\forall i,j \in V_G : \mathrm{NRP}(i,j)$ *in* $(G, lab) \iff \mathrm{NRP}(i,j)$ *in* $(G, lab^{a=\oplus})$
2. $\forall i,j \in V_G : \mathrm{RP}(i,j)$ *in* $(G, lab) \iff \mathrm{RP}(i,j)$ *in* $(G, lab^{a=\oplus})$

The previous result is analogous for a negative forced arc.

Proof. We prove only the first case, the second one is analogous.

($\Longrightarrow$) It is direct, because $\left(G, lab^{(i,j)=\oplus}\right)$ is an extension of (G, lab). As in a extension there are at least the same labeled arcs. Hence, it is easy to see that if there exists a negative reverse path in the original digraph, then there exists in its extension.

($\Longleftarrow$) Considering a negative reverse path in $(G, lab^{a=\oplus})$ we show that there exists a negative reverse path in (G, lab). The only difference between both digraphs is the positive forced arc a. We know that any forced arc do not add information in terms of reverse and negative reverse paths.

A.5. Divide and conquer

A.5.1. Divide by bridges

Proposition 12. *Let (G, lab) be a labeled connected digraph, G_U the underlying graph of G and $uv \in E(G_U)$ a bridge that divide G in G_1 and G_2, we denote by $G_b = G[\{u, v\}]$ then*

$$\mathcal{S}(G, lab) = \mathcal{S}(G_1, lab|_{A_{G_1}}) \otimes \mathcal{S}(G_b, lab|_{A_{G_b}}) \otimes \mathcal{S}(G_2, lab|_{A_{G_2}})$$

Proof. Given $lab' \in \mathcal{S}(G, lab)$, then it is clear that $lab' \in \mathcal{S}(G_1, lab|_{A_{G_1}}) \otimes \mathcal{S}(G_b, lab|_{A_{G_b}}) \otimes \mathcal{S}(G_2, lab|_{A_{G_2}})$.

Given $lab' \in \mathcal{S}(G_1, lab_1) \otimes \mathcal{S}(G_b, lab|_{A_{G_b}}) \otimes \mathcal{S}(G_2, lab_2)$, let suppose that $u \in G_1$ and $v \in G_2$, and $lab' \notin \mathcal{S}(G, lab)$, since is clear that lab' is a fully label extension, then (G, lab') is a non update digraph and it contains a forbidden cycle. Without loss of generality, we say that there exists a cycle starting in u_1 which is part of G_1 component, as (G_1, lab_1) is an update digraph, it cannot exists a cycle in it. This implies that (u, v) must be part of the sequence, but it could not exist a forbidden cycle in G_b, so the cycle should be in G_2, which is impossible. Hence, $lab \in \mathcal{S}(G, lab')$.

A.5.2. Divide by strongly connected components

Another way to simplify the digraph consists in to divide it by its strongly connected components in the extended reverse digraph, i.e. the reverse digraph where unlabeled arcs are replaced by unlabeled arcs in both directions.

Proposition 13. *Let (G, lab) be an update digraph with $G_1, \ldots, G_k$ its strongly connected components of the extended reverse digraph, we define a set of arcs:*

$$A_T = \bigcup_{i,j} A_G \cap \left(V_{G_i} \times V_{G_j}\right),$$

which it is composed by the arcs between SCC of the extended reverse digraph of (G, lab). Then, we verify that:

$$\mathcal{S}(G, lab) = \mathcal{S}(\widetilde{G}_1, lab|_{A_{G_1}}) \otimes \cdots \otimes \mathcal{S}(\widetilde{G}_k, lab|_{A_{G_k}}) \otimes \{lab|_{A_T}\}$$

where $\forall i \in \{1, \ldots, k\}$, $\widetilde{G}_i = G[V_{G_i}]$

Proof. Given $lab' \in \mathcal{S}(G, lab)$ if there is not a forbidden cycle in G, neither there is a cycle in any of its strongly connected components over its extended reverse digraph, therefore $lab' \in \mathcal{S}(\widetilde{G}_1, lab|_{A_{G_1}}) \otimes \cdots \otimes \mathcal{S}(\widetilde{G}_k, lab|_{A_{G_k}}) \otimes \{lab|_{A_T}\}$. Furthermore, $lab' \in \mathcal{S}(\widetilde{G}_1, lab|_{A_{G_1}}) \otimes \cdots \otimes \mathcal{S}(\widetilde{G}_k, lab|_{A_{G_k}}) \Longrightarrow lab' \in \mathcal{S}(G, lab)$. This is easy to prove, because as they are strongly connected components, there are not back and forth paths between them.

A.6. Application

name	id	function
CycD	v_1	x_{v_1}
Rb	v_2	$(\neg x_{v_1} \wedge \neg x_{v_{10}}) \wedge ([\neg x_{v_4} \wedge \neg x_{v_5}] \vee x_{v_6})$
E2F	v_3	$(\neg x_{v_2} \wedge \neg x_{v_5} \wedge \neg x_{v_{10}}) \vee (x_{v_6} \wedge \neg x_{v_2} \wedge \neg x_{v_{10}})$
CycE	v_4	$x_{v_3} \wedge \neg x_{v_2}$
CycA	v_5	$(\neg x_{v_2} \wedge \neg x_{v_7} \wedge \neg(x_{v_8} \wedge x_{v_9})) \wedge (x_{v_3} \vee x_{v_5})$
p27	v_6	$(\neg x_{v_1} \wedge \neg x_{v_{10}}) \wedge ([\neg x_{v_4} \wedge \neg x_{v_5}] \vee [x_{v_6} \wedge \neg(x_{v_4} \wedge x_{v_5})])$
Cdc20	v_7	$x_{v_{10}}$
Cdh1	v_8	$(\neg x_{v_5} \wedge \neg x_{v_{10}}) \vee x_{v_7} \vee (x_{v_6} \wedge \neg x_{v_{10}})$
UbcH10	v_9	$\neg x_{v_8} \vee (x_{v_8} \wedge x_{v_9} \wedge [x_{v_7} \vee x_{v_5} \vee x_{v_{10}}])$
CycB	v_{10}	$\neg x_{v_7} \wedge \neg x_{v_8}$

Table 2: Notation for the nodes of the mammalian cell cycle network (G_M)

References

[1] Albert, R., Othmer, H. G., 2003. The topology of the regulatory interactions predicts the expression pattern of the drosophila segment polarity genes. Journal of Theoretical Biology 223, 1–18.

[2] Aracena, J., Demongeot, J., Fanchon, E., Montalva, M., 2013. On the number of different dynamics in boolean networks with deterministic update schedules. Mathematical biosciences 242 (2), 188–194.

[3] Aracena, J., Fanchon, E., Montalva, M., Noual, M., 2011. Combinatorics on update digraphs in Boolean networks. Discrete Applied Mathematics 159 (6), 401–409.

[4] Aracena, J., Goles, E., Moreira, A., Salinas, L., 2009. On the robustness of update schedules in Boolean networks. Biosystems 97, 1–8.

[5] Aracena, J., Gómez, L., Salinas, L., 2013. Limit cycles and update digraphs in Boolean networks. Discrete Applied Mathematics 161, 1–2.

[6] Chaves, M., Réka, A., Sontag, E., 2005. Robustness and fragility of Boolean models for genetic regulatory networks. Journal of Theoretical Biology 235, 431–449.

[7] Davidich, M. I., Bornholdt, S., 2008. Boolean network model predicts cell cycle sequence of fission yeast. PloS one 3 (2), e1672.

[8] Demongeot, J., Hazgui, H., Amor, H. B., Waku, J., 2014. Stability, complexity and robustness in population dynamics. Acta biotheoretica 62 (3), 243–284.

[9] Elena, A., 2009. Robustesse des réseaux d'automates booleéns a seuil aux modes d'itération. application a la modélisation des réseaux de régulation génétique. Ph.D. thesis, Université Joseph Fourier (Grenoble I), Grenoble, France.

[10] Fauré, A., Naldi, A., Chaouiya, C., Thieffry, D., 2006. Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. Bioinformatics 22, 124–131.

[11] Goles, E., Montalva, M., Ruz, G., 2013. Deconstruction and dynamical robustness of regulatory networks: Application to the yeast cell cycle networks. Bull Math Biol 75, 939–966.

[12] Goles, E., Noual, M., 2012. Disjunctive networks and update schedules. Advances in Applied Mathematics 48, 646–662.

[13] Goles, E., Salinas, L., 2008. Comparison between parallel and serial dynamics of Boolean networks. Theoretical Computer Science 396, 247–253.

[14] Kauffman, S., 1969. Metabolic stability and epigenesis in randomly connected nets. Journal of Theoretical Biology 22, 437–67.

[15] Kauffman, S., Peterson, C., Samuelsson, B., Troein, C., 2003. Random Boolean network models and the yeast transcriptional network. Proceedings of the National Academy of Sciences 100, 14796–14799.

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

[17] Linial, N., 1986. Hard enumeration problems in geometry and combinatorics. SIAM Journal on Algebraic Discrete Methods 7 (2), 331–335.

[18] Mendoza, L., Alvarez-Buylla, E., 1998. Dynamics of the genetic regulatory network for arabidopsis thaliana flower morphogenesis. Journal of Theoretical Biology 193, 307–319.

[19] Meng, M., Feng, J., 2014. Function perturbations in Boolean networks with its application in a d. melanogaster gene network. European Journal of Control 20 (2), 87 – 94.
URL http://www.sciencedirect.com/science/article/pii/S0947358014000028

[20] Mortveit, H., Reidys, C., 2001. Discrete, sequential dynamical systems. Discrete Mathematics 226, 281–295.

[21] Robert, F., 1986. Discrete Iterations: A Metric Study. Springer-Verlag, Berlin.

[22] Ruz, G. A., Goles, E., 2013. Learning gene regulatory networks using the bees algorithm. Neural Computing and Applications 22 (1), 63–70.

[23] Ruz, G. A., Goles, E., Montalva, M., Fogel, G. B., 2014. Dynamical and topological robustness of the mammalian cell cycle network: A reverse engineering approach. Biosystems 115, 23–32.

[24] Ruz, G. A., Timmermann, T., Barrera, J., Goles, E., 2014. Neutral space analysis for a boolean network model of the fission yeast cell cycle network. Biological Research 47 (1), 64.

[25] Singh, A., Nascimento, J. M., Kowar, S., Busch, H., Boerries, M., 2012. Boolean approach to signalling pathway modelling in hgf-induced keratinocyte migration. Bioinformatics 28 (18), i495–i501.

[26] Thomas, R., 1973. Boolean formalization of genetic control circuits. Journal of Theoretical Biology 42, 563–585.