Bioinformatics, 32(5), 2016, 722–729
doi: 10.1093/bioinformatics/btv628
Advance Access Publication Date: 31 October 2015
Original Paper

Systems biology

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

Eduardo Palma[1], Lilian Salinas[1,3,*] and Julio Aracena[2,3]

[1]Departamento de Ingeniería Informática y Ciencias de la Computación, Universidad de Concepción, Piso 3, Concepción, Chile. [2]Departamento de Ingeniería Matemática, Universidad de Concepción, Casilla 160-C, Concepción, Chile and [3]Centro de Investigación en Ingeniería Matemática, CI²MA, Casilla 160-C, Concepción, Chile

*To whom correspondence should be addressed.

Associate Editor: Igor Jurisica

Received on March 13, 2015; revised on October 13, 2015; accepted on October 25, 2015

Abstract

Motivation: Boolean networks (BNs) are commonly used to model genetic regulatory networks (GRNs). Due to the sensibility of the dynamical behavior to changes in the updating scheme (order in which the nodes of a network update their state values), 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 Aracena *et al.* equivalence classes of deterministic update schedules in BNs, that yield exactly the same dynamical behavior of the network, were defined according to a certain label function on the arcs of the interaction digraph defined for each scheme. Thus, the interaction digraph so labeled (update digraphs) encode the non-equivalent schemes.

Results: We address the problem of enumerating all non-equivalent deterministic update schedules of a given BN. First, we show that it is an intractable problem in general. To solve it, we first construct an algorithm that determines the set of update digraphs of a BN. For that, we use divide and conquer methodology based on the structural characteristics of the interaction digraph. Next, for each update digraph we determine a scheme associated. This algorithm also works in the case where there is a partial knowledge about the relative order of the updating of the states of the nodes. We exhibit some examples of how the algorithm works on some GRNs published in the literature.

Availability and implementation: An executable file of the UpdateLabel algorithm made in Java and the files with the outputs of the algorithms used with the GRNs are available at: www.inf.udec.cl/ ~lilian/UDE/

Contact: lilisalinas@udec.cl

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Gene regulation networks (GRNs) consist of interacting genes and gene products, which give rise to complex cell behaviors. Due mainly to the lack of detailed kinetic information and quantitative data, qualitative logical models have been used for a better understanding and simulation of GRNs. Examples of these models are the Boolean networks (BNs), which despite their simplicity are able to capture key dynamical features and predict some activation patterns of GRNs (Albert and Othmer, 2003; Davidich and Bornholdt, 2008).

Boolean networks were introduced by Stuart Kauffman in Kauffman (1969) to model GRNs. The gene expression level, in this case, is modeled by binary values, 1 or 0, 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 or gene products). The interactions between the elements of a BN is represented by a directed graph (digraph), named interaction digraph, where there is an arc (i, j) in the interaction digraph if the state value of the node j depends on the state value of the node i. 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. Some examples of GRNs modeled by synchronous Boolean networks are exhibited in Albert and Othmer (2003), Kauffman *et al.* (2003), Li *et al.* (2004), Davidich and Bornholdt (2008) and Singh *et al.* (2012). However, due to the synchronous scheme is considered not being very realistic many GRN modelers have used other update schedules with different levels of asynchronism (Fauré *et al.*, 2006; Mendoza and Alvarez-Buylla, 1998; Thomas, 1973).

The difficulty of determining time delays of updating in a GRN makes modelers often sample different update schedules to better capture an observed biological phenomenon. In this context, some used schemes are the deterministic update schedules, where the set of nodes of a network is partitioned into blocks, which are updated sequentially while within the blocks, the nodes are updated synchronously in every time step, (see for example Demongeot *et al.*, 2014; Goles *et al.*, 2013; Ruz and Goles, 2013; Ruz *et al.*, 2014a,b; Meng and Feng, 2014). This family of update schedules, introduced by F. Robert in Robert (1986), 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 (Aracena *et al.*, 2013a; Goles and Noual, 2012; Elena, 2009; Goles and Salinas, 2008; Mortveit and Reidys, 2001; Robert, 1986).

In BNs with deterministic update schedules the relative order of updating of two nodes interacting can be represented by a label on the respective arc of the interaction digraph. More precisely, we put on the arc (i, j) the label $\oplus$ if the node i is updated after or at the same time than j, according to a given update schedule of the network, and the label $\circleddash$ otherwise. The interaction digraph of a BN labeled in this way is called update digraph (Aracena *et al.*, 2009). In this paper we also deal with deterministic update schedules which are not fully defined, that is, the relative order of updating for some pairs of nodes is unknown. In such cases we put on the corresponding arcs the label $\bigcirc$.

In Aracena *et al.* (2009) was proved that given a BN with two different deterministic update schedules, if the schemes have the same update digraph associated, then the dynamical behaviors of the networks under these schemes are equal. Hence, equivalence classes of deterministic update schedules in a given BN can be defined according to the update digraph associated, and such that two equivalent schemes yield exactly the same dynamical behavior of the network. Thus, in order to know the different dynamical behaviors of a BN (for example to study the robustness of the dynamics against to changes in the updating scheme), modelers of GNRs need to test only non-equivalent update schedules, being this set of schemes usually much smaller than the total set (see Supplementary information and Aracena *et al.*, 2013b). Similarly, if someone wants to determine the dynamics of a BN with updating schemes that are compatible with a set of constraints (on the relative order of updating of some pairs of nodes) then it is sufficient to test the non-equivalent extensions of this update schedule partially defined; this is useful, for example, when there are constraints that come from wanting to keep some dynamical property of a BN.

In this way, two important problems to solve are determining all non-equivalent deterministic update schedules of a BN and the non-equivalent extensions of an updating scheme partially defined. In this paper, we address both problems and construct efficient algorithms to solve them. For that, we first build algorithms that determine the update digraphs associated to a given BN, which encoding the non-equivalent updates schedules. Next, we use the algorithm introduced in Aracena *et al.* (2011), and shown in Supplementary information, to determine in polynomial time a scheme associated to each found update digraph and hence to enumerate all non-equivalent update schedules of the network.

The algorithms designed to calculate the update digraphs associated to a BN use two strategies. The first one is to 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 of four Boolean models of GRNs published in the literature: Arabidopsis Thaliana regulatory network (Sánchez-Corrales *et al.*, 2010), Yeast transcriptional network (Kauffman *et al.*, 2003), the network for Body segmentation in Drosophila Melanogaster (Albert and Othmer, 2003) and Mammalian Cell Cycle network (Fauré *et al.*, 2006). Besides, for this latter network we determine the non-equivalent extensions of an update schedule partially defined, whose restrictions are necessary in order to keep the unique limit cycle of the network synchronously updated.

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 $f_v : \{0,1\}^n \to \{0,1\}$ are called *local activation functions*, and an *update schedule* defined by a function $s : V \to \{1, \ldots, n\}$, where $s(v) = k$ means that in each unit of time the state of node v is updated in the kth place (see an example in Supplementary information). An update schedule s is also denoted by $s = \{v : s(v) = 1\}\{v : s(v) = 2\} \cdots \{v : s(v) = n\}$.

The state values in a Boolean network with update function s are 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 $s(v) > s(u)$.

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 Fig. 1).

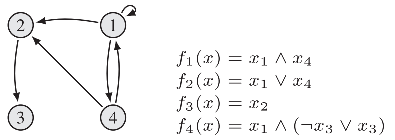
$f_1(x) = x_1 \wedge x_4$
$f_2(x) = x_1 \vee x_4$
$f_3(x) = x_2$
$f_4(x) = x_1 \wedge (\neg x_3 \vee x_3)$

Fig. 1. Interaction digraph associated to a Boolean network. Notice that f_4 does not depend on x_3

Given a digraph G a *label function* is any function $lab : A_G \to \{\oplus, \ominus, \bigcirc\}$. We call labeled digraph to (G, 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* of lab 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 (see Example 1 in Supplementary information).

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 Aracena *et al.* (2009) was defined the *update digraph* associated to a BN $N = (F, s)$ 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 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 Aracena *et al.* (2011) (see Supplementary information).

Example 1: In Figure 2 we show two labeled digraphs. In (a) it is shown an update digraph. Indeed, for the update schedule $s = \{3,4\}\{5\}\{1,2\}$ we have that $lab_1 = lab_s$. In (b), the labeled digraph (G, lab_2) is not an update digraph. Since otherwise, for any given update schedule s, $s(3) < s(5) < s(4) \leq s(3)$, which is a contradiction.

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 (see Example 2 in Supplementary information).

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

The condition required for a labeled digraph to be an update digraph it is related to the notion of reverse graph defined as follows: Given a labeled digraph (G, lab), we define the *reverse digraph* (G_R, lab_R), where $V_{G_R} = V_G$,

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

and $lab_R(u,v) = \ominus$ if $(v,u) \in A^{\ominus}_{(G,lab)}$ and $lab_R(u,v) = \oplus$ otherwise. In Figure 3 we show an example of a labeled digraph and its associated reverse digraph.

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, if there exists a path from v_1 to v_n in the reverse digraph.

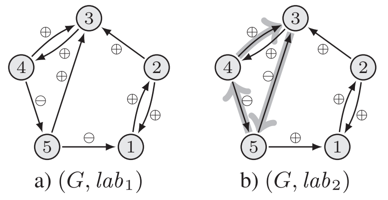

Fig. 2. **(a)** Example of update digraph. **(b)** Example of non-update digraph, where the gray arrows form a forbidden cycle

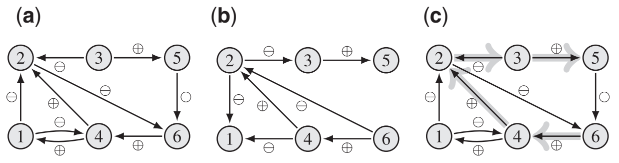

Fig. 3. **(a)** A labeled digraph (G, lab). **(b)** The associated reverse digraph (G_R, lab_R). **(c)** A negative reverse path (marked by gray arrows): $(6,4,2,3,5)$ in (G, lab)

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 3c 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* is a negative reverse path $(v_1, v_2, \ldots, v_n)$ where $v_1 = v_n$ (see an example in Fig. 2b, where the forbidden cycle is defined by the gray arrows). It was proved in Aracena *et al.* (2011) that a labeled digraph is an update digraph if and only if there does not exist any forbidden cycle.

3 Complexity of update digraph extension problem

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

Update digraph extension (UDE): Given a labeled digraph (G, lab), 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, 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 Supplementary information) is based on 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 the enumeration of all extensions from a partially labeled digraph is a hard problem, while the related existence problem is known to be polynomial (Aracena *et al.*, 2011).

4 Algorithms

In this section we present the theoretical results that lead to design an algorithm that solves 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* (i.e. strongly connected component in the digraph induced by the positive arcs of the labeled digraph) 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: $M(u,v) = -1$ if there exists a negative reverse path from u to v; $M(u,v) = 1$ if there is a reverse path from u to v, but not a negative one and $M(u,v) = \infty$ otherwise. See details of ReversePaths and Verify algorithms in Supplementary information.

4.2 Reducing the size of the instances of UDE problem

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 decrease in size of an instance of UDE 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 Supplementary information) 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 $R(G, lab) = (G_{\mathrm{rd}}, lab_{\mathrm{rd}})$, where $G_{\mathrm{rd}} = (V_{\mathrm{rd}}, A_{\mathrm{rd}})$, $V_{\mathrm{rd}} = \{v_1, \ldots, v_k\}$ and $A_{\mathrm{rd}} = \{(v_i, v_j) | \exists (u,v) \in A_G \cap (V_{G_i} \times V_{G_j})\}$

Furthermore, $lab_{\mathrm{rd}}(v_i, v_j) = lab(u,v)$, if $\exists (u,v) \in (V_{G_i} \times V_{G_j}) \cap \mathrm{Sup}(lab)$ and $lab_{\mathrm{rd}}(v_i, v_j) = \bigcirc$ 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.

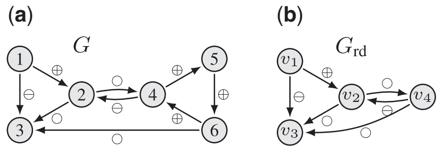

Fig. 4. (a) An update digraph. **(b)** The reduced digraph

Example 2: In Figure 4 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 Supplementary information). 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 (Tarjan, 1972).

Algorithm 1 Reduce

```
Require: An update digraph $(G, lab)$.
Ensure: The reduced digraph $(G_{\mathrm{rd}}, lab_{\mathrm{rd}})$.
 1: $\{G_1, \ldots, G_k\} \leftarrow \mathrm{SCC}^+(G, lab)$
 2: $V_{G_{\mathrm{rd}}} \leftarrow \{w_1, \ldots, w_k\}$
 3: $A_{G_{\mathrm{rd}}} \leftarrow \emptyset$
 4: for $i = 1$ to $k$ do
 5:   for $j = 1$ to $k$ do
 6:     if $\exists (u,v) \in A_G$ with $u \in V_{G_i}$ and $v \in V_{G_j}$ then
 7:       $A_{G_{\mathrm{rd}}} \leftarrow A_{G_{\mathrm{rd}}} \cup (w_i, w_j)$
 8: for all do$(w_i, w_j) \in A_{G_{\mathrm{rd}}}$
 9:   if $\exists u \in V_{G_i}, v \in V_{G_j}$ and $lab(u,v) = \ominus$ then
10:     $lab_{\mathrm{rd}}(u,v) \leftarrow \ominus$
11:   else if $\exists u \in V_{G_i}, v \in V_{G_j}$ and $lab(u,v) = \oplus$ then
12:     $lab_{\mathrm{rd}}(u,v) \leftarrow \oplus$
13:   else
14:     $lab_{\mathrm{rd}}(u,v) \leftarrow \bigcirc$
15: 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 3: In Figure 5a we see that there exist negative reverse paths (marked for gray arrows) from 3 to 2 (3,1,2) and from 2 to 4 (2,4), then the unlabeled arcs (2,3) and (2,4) must be labeled negative and positive respectively, as shown in b.

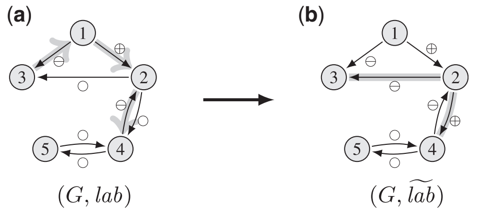

Fig. 5. **(a)** Example of update digraph where the arcs (2, 3) and (2, 4) are forced to be negative and positive, respectively. The gray arrows define reverse paths. **(b)** $(G, \widetilde{lab})$ is the maximal extension of (G, lab)

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. For all $(i, j) \in A_G$ there exists a negative reverse path from j to i if and only if $(G, lab^{(i,j)=\oplus})$ is a nonupdate digraph; then we say that the arc (i, j) is forced to be negative.
2. For all $(i, j) \in A_G$ there exists a reverse path from i to j if and only if $(G, lab^{(i,j)=\ominus})$ is a nonupdate digraph; then we say that the arc (i, j) is forced to be positive.

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

($\Rightarrow$) 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 nonupdate digraph.

($\Leftarrow$) If $\left(G, lab^{(i,j)=\oplus}\right)$ is a nonupdate 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. Because, the labeling of forced arcs does 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 negative the 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: $\widetilde{lab} \leftarrow lab$
2: $M \leftarrow \text{ReversePaths}(G, lab)$
3: for all $a \in \text{Sup}(lab)$, with $a = (i, j)$ do
4:     if $M(i, j) \neq \infty$ then
5:         $\widetilde{lab} \leftarrow \widetilde{lab}^{a=\oplus}$
6:     else if $M(j, i) = -1$ then
7:         $\widetilde{lab} \leftarrow \widetilde{lab}^{a=\ominus}$
8: return $\widetilde{lab}$
```

Next, we introduce a simple algorithm that name SimpleLabel, to find all the extensions of a given partially labeled update digraph and which uses Force algorithm. Firstly, this algorithm finds the maximal extension of the given label function. Thereafter, the algorithm labels an unlabeled arc positive 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 (see Algorithm 3).

Algorithm 3 SimpleLabel

```
Require: An update digraph $(G, lab)$
Ensure: The set $\mathcal{S}(G, lab)$ denoted by $S$ and its cardinal $r := |S|$
1: $(S, r) \leftarrow (\emptyset, 0)$
2: $lab \leftarrow \text{Force}(G, lab)$
3: if $\text{Sup}(lab) = A(G)$ then
4:     return $(lab, 1)$
5: else
6:     Let be $a \in A(G) \setminus \text{Sup}(lab)$
7:     $(S_1, r_1) \leftarrow \text{SimpleLabel}(G, lab^{a=\oplus})$
8:     $(S_2, r_2) \leftarrow \text{SimpleLabel}(G, lab^{a=\ominus})$
9:     $(S, r) \leftarrow (S_1 \cup S_2, r_1 + r_2)$
10: 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 the operator $\otimes$.

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

$$L_1 \otimes \ldots \otimes L_k = \{lab : A_G \to \{\oplus, \ominus, \bigcirc\} | \forall i = 1, \ldots, k, lab|_{A_i} \in L_i\}.$$

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

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

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 Supplementary information. In Figure 6 we can see an example of division by bridges of a labeled digraph with the associated partition of the arc set.

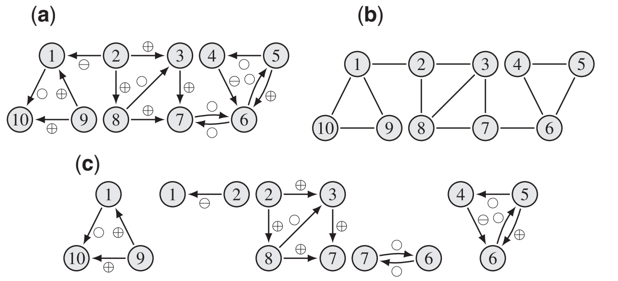

Fig. 6. (a) Labeled digraph. **(b)** Underlying graph. **(c)** Partition of the arc set produced by the division by bridges

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 $G_1, \dots, G_k$ its strongly connected components of the extended reverse digraph, we define the set of arcs:

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

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

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

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

In Figure 7 we can see an example of division by strongly connected components of an update digraph with the associated partition of the arc set.

The proof of this last proposition is similar to that of the Proposition 6 shown in Supplementary information, 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 same ideas of SimpleLabel, i.e. to force arcs, label one and apply recursively the same algorithm.

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 $r := |S|$
1: $(S, r) \leftarrow (\emptyset, 1)$
2: $\{(G_1, lab_{G_1}), \dots, (G_k, lab_{G_k})\} \leftarrow \text{Bridges}(G_U)$
3: for $i = 1$ to $k$ do
4:   if $\text{Sup}(lab_{G_i}) = A_{G_i}$ then
5:     $S \leftarrow S \otimes \{lab_{G_i}\}$
6:   else
7:     $\{(H_1, lab_{H_1}), \dots, (H_p, lab_{H_p})\} \leftarrow \text{SCC}^+(G_i, lab_{G_i})$
8:     if $p = 1$ then
9:       Let $a \in A_{H_1}$ be an arc such that $lab(a) = \bigcirc$
10:      $lab^+ \leftarrow \text{Force}(H_1, lab_{H_1}^{a=\oplus})$
11:      if $\text{Sup}(lab^+) \neq A_{H_1}$ then
12:        $(\hat{S}, \hat{r}) \leftarrow \text{Label}(H_1, lab^+)$
13:      else
14:      $(\hat{S}, \hat{r}) \leftarrow (\{lab^+\}, 1)$
15:        $lab^- \leftarrow \text{Force}(H_1, lab_{H_1}^{a=\ominus})$
16:        if $\text{Sup}(lab^-) \neq A_{H_1}$ then
17:          $(\widetilde{S}, \widetilde{r}) \leftarrow \text{Label}(H_1, lab^-)$
18:        else
19:          $(\widetilde{S}, \widetilde{r}) \leftarrow (\{lab^-\}, 1)$
20:          $(S, r) \leftarrow (S \otimes (\hat{S} \cup \widetilde{S}), r \cdot (\hat{r} + \widetilde{r}))$
21:    else
22:      for $j = 1$ to $p$ do
23:        if $\text{Sup}(lab_{H_j}) = A_{H_j}$ then
24:          $S \leftarrow S \otimes \{lab_{H_j}\}$
25:        else
26:          $(\widetilde{S}, \widetilde{r}) \leftarrow \text{Label}(H_j, lab_{H_j})$
27:          $(S, r) \leftarrow (S \otimes \widetilde{S}, r \cdot \widetilde{r})$
28: 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 $\mathcal{S}(G, lab) \neq \emptyset$ (see Algorithm 5).

4.5 Comparison of algorithms with and without divisions

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 UpdateLabel algorithm to call both. The tests were run in complete digraphs and chains (see Fig. 8) of different sizes and with empty supports in a laptop with Processor: 2.4 GHz Intel Core i5, RAM memory: 8 GB 1600 MHz DDR3, operating system: OS X 10.9.5. In Table 1 we can see as the main algorithm with Label runs (column Label) faster than with SimpleLabel (column SimpleL), even in the case of complete digraphs where there is a small number of divisions. However, when the number of solutions (i.e. $|\mathcal{S}(G, lab)|$) is big it is convenient to use SimpleLabel instead of Label, because the latter uses a lot of RAM memory (see for example the case of K_9). It is important to mention that the times listed in columns SimpleL and Label of Table 1 correspond to the runtimes of the implementation of UpdateLabel algorithm available at www.inf.udec.cl/~lilian/UDE/ with output the list of non-equivalent schemes associated to the update digraphs of $S(G, lab)$. In the cases of columns Coded and Count, they correspond to the implementation with output $S(G, lab)$ and $|S(G, lab)|$ respectively.

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

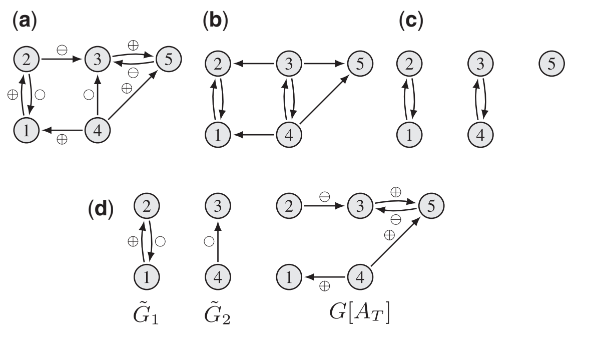

Fig. 7. (a) Update digraph (G, lab). (b) Extended reverse digraph (G', lab'). (c) Strongly connected components of (G', lab'). (d) Partition of the arc set of G produced by the division by strongly connected components

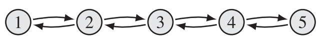

Fig. 8. Example of a chain P_n where $n=5$

Table 1. Results obtained when the main algorithm is used with and without divisions on some digraphs K_n and P_n. Here, $|V|$ and $|A|$ denote the number of nodes and arcs of each digraph, respectively

Graph	$\|V\|$	$\|A\|$	$\|\mathcal{S}(G, lab)\|$	Count (s)	SimpleL (s)	Label (s)	Coded (s)
K_3	3	6	13	0.002	0.003	0.002	0.002
K_4	4	12	75	0.003	0.008	0.006	0.007
K_5	5	20	541	0.015	0.029	0.016	0.020
K_6	6	30	4 683	0.011	0.067	0.051	0.015
K_7	7	42	47 293	0.022	0.254	0.113	0.071
K_8	8	56	545 835	0.164	3.325	1.065	0.328
K_9	9	72	7 087 261	1.254	51.945	–	3.326
P_{10}	10	18	19 683	0.001	0.217	0.078	0.001
P_{20}	20	38	1 162 261 467	0.001	–	–	0.001
P_{30}	30	58	1 094 942 099	0.001	–	–	0.001

5 Application to genetic regulatory networks

We apply our algorithm UpdateLabel to enumerate the non-equivalent deterministic update schedules of four Boolean models of GRNs published in the literature: Mammalian Cell Cycle network (Fauré *et al.*, 2006), Arabidopsis Thaliana regulatory network (Sánchez-Corrales *et al.*, 2010), Yeast transcriptional network (Kauffman *et al.*, 2003) and the network for body segmentation in Drosophila Melanogaster (Albert and Othmer, 2003), also studied in (Marques-Pita and Rocha, 2013).

The mammalian cell cycle network has 10 nodes and 31 arcs. A detailed description of the network can be found in Supplementary information. For this network we used the implementation of UpdateLabel algorithm available at www.inf.udec.cl/~lilian/UDE/ without divisions and with divisions to enumerate all non-equivalent deterministic update schedules of the network (row Mammalian1 of Table 2) and those that are extensions of the partially labeled interaction digraph shown in Supplementary information, and corresponding to label eight arcs of the network as positive, in order to preserve the limit cycle C of the synchronous dynamical behavior also specified in Supplementary information (row Mammalian2 of Table 2). The times listed in Table 2 correspond to the same of those

Table 2. Results obtained when the main algorithm is used with and without divisions on some networks. Here, $|V|$ and $|A|$ denote the number of nodes and arcs of the interaction digraph of each network, respectively

Graph	$\|V\|$	$\|A\|$	$\|\mathcal{S}(G, lab)\|$	Count (s)	SimpleL (s)	Label (s)	Coded (s)
Mammalian1	10	31	466 712	0.755	5.230	2.966	1.327
Mammalian2	10	23	1 440	0.005	0.016	0.033	0.009
Arabidopsis	13	39	7 062 567	11.349	113.851	–	19.237
Yeast	30	22	206 427	0.222	30.768	3.595	0.322
Drosophila	18	12	2 368	0.001	0.093	0.023	0.001

specified in Table 1. It is important to observe that the quantities 466712 and 1440 represent an upper bound for the total number of different dynamics and for the number of possible dynamics exhibiting the limit cycle C, of the mammalian cell cycle network with deterministic update schedules, respectively. Besides, these number are much less than the total number of deterministic update schedules for the network, which it is approximately 1×10^8 (see Supplementary information). We also applied the implementation of the algorithms to the other mentioned networks labeling some arcs as negative when the 'head' vertex of the arc has associated a constant function as local activation function. The corresponding partially labeled interaction digraphs are shown in Supplementary information. In the case of Arabidopsis Thaliana we only use SimpleLabel algorithm because the size is too big to process in RAM Memory. The results obtained are shown in Table 2, where $|\mathcal{S}(G, lab)|$ is an upper bound of the number of possible dynamics of each network when are modeled by BNs with deterministic update schedules. The files with the outputs of the algorithms for each network can be found at: www.inf.udec.cl/~lilian/UDE/Files/.

6 Discussion

The problem of testing different deterministic update schedules of a BN modeling a GRN, for example to better capture an observed biological phenomenon or to study the robustness of the dynamics against to changes in the updating scheme, is reduced to use only non-equivalent schemes. This set, defined according to the update digraphs associated to a network, can be much smaller than the total set of schemes (Supplementary information and Aracena *et al.* 2013b). In this article we addressed the problems of determining all non-equivalent deterministic update schedules of a BN and the non-equivalent extensions of an updating scheme partially defined. To solve them, we first construct an algorithm, named UpdateLabel, determining the label functions on the arcs of the interaction digraph of a BN that have an update schedule associated (i.e. the set of update digraphs of a BN). The UpdateLabel algorithm uses two major ideas in its design. The first one is the base of the Force algorithm, which in polynomial time checks whether the given labels on some arcs uniquely determine the label in others (only possible extension), allowing 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 procedure significantly reduces the total execution time of the main algorithm as observed in Table 1. Next, for each update digraph found with UpdateLabel algorithm we determine an update schedule scheme belonging to the class, by using a polynomial algorithm introduced in Aracena *et al.* (2011) and exhibited in Supplementary information. We illustrate the application of our algorithms on four GRNs published in the literature. The results obtained and exhibited in Table 2 show that in few seconds we can obtain the whole set of non-equivalent update schedules of each studied network, whose cardinals correspond to the maximum number of possible dynamical behaviors of the studied networks when they are modeled by BNs with deterministic update schedules. It is important to note that despite the UpdateLabel algorithm with divisions on the interaction digraph is faster, it is limited by the RAM memory of the computer. So for large networks is advisable to use simply the UpdateLabel algorithm without division, when the number of non-equivalent schemes so allows.

Funding

This work was partially supported by project Fondecyt 1131013 (J.A. and L.S.) and Master's scholarship CONICYT (E.P.).

Conflict of Interest: none declared.

References

Albert,R. and Othmer,H.G. (2003) The topology of the regulatory interactions predicts the expression pattern of the drosophila segment polarity genes. *J. Theor. Biol.*, **223**, 1–18.

Aracena,J. *et al.* (2009). On the robustness of update schedules in Boolean networks. *Biosystems*, **97**, 1–8.

Aracena,J. *et al.* (2011). Combinatorics on update digraphs in Boolean networks. *Discrete Appl. Math.*, **159**, 401–409.

Aracena,J. *et al.* (2013a) Limit cycles and update digraphs in Boolean networks. *Discrete Appl. Math.*, **161**, 1–2.

Aracena,J. *et al.* (2013b) On the number of different dynamics in boolean networks with deterministic update schedules. *Math. Biosci.*, **242**, 188–194.

Davidich,M.I. and Bornholdt,S. (2008) Boolean network model predicts cell cycle sequence of fission yeast. *PloS One*, **3**, e1672.

Demongeot,J. *et al.* (2014) Stability, complexity and robustness in population dynamics. *Acta Biotheoretica*, **62**, 243–284.

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.

Fauré,A. *et al.* (2006) Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics*, **22**, 124–131.

Goles,E. and Noual,M. (2012) Disjunctive networks and update schedules. *Adv. Appl. Math.*, **48**, 646–662.

Goles,E. and Salinas,L. (2008) Comparison between parallel and serial dynamics of Boolean networks. *Theor. Comput. Sci.*, **396**, 247–253.

Goles,E. *et al.* (2013) Deconstruction and dynamical robustness of regulatory networks: application to the yeast cell cycle networks. *Bull. Math. Biol.*, **75**, 939–966.

Kauffman,S. (1969) Metabolic stability and epigenesis in randomly connected nets. *J. Theor. Biol.*, **22**, 437–67.

Kauffman,S. *et al.* (2003) Random Boolean network models and the yeast transcriptional network. *Proc. Natl Acad. Sci.*, **100**, 14796–14799.

Li,F. *et al.* (2004) The yeast cell-cycle network is robustly designed. *Proc. Natl Acad. Sci. USA*, **101**, 4781–4786.

Marques-Pita,M. and Rocha,L.M. (2013) Canalization and control in automata networks: Body segmentation in drosophila melanogaster. *PLoS ONE*, **8**, e55946.

Mendoza,L. and Alvarez-Buylla,E. (1998) Dynamics of the genetic regulatory network for arabidopsis thaliana flower morphogenesis. *J. Theor. Biol.*, **193**, 307–319.

Meng,M. and Feng,J. (2014) Function perturbations in Boolean networks with its application in a d. melanogaster gene network. *Eur. J. Control*, **20**(2), 87–94.

Mortveit,H. and Reidys,C. (2001) Discrete, sequential dynamical systems. *Discrete Math.*, **226**, 281–295.

Robert,F. (1986) *Discrete Iterations: A Metric Study*. Springer-Verlag, Berlin.

Ruz,G.A. and Goles,E. (2013) Learning gene regulatory networks using the bees algorithm. *Neural Comput. Appl.*, **22**, 63–70.

Ruz,G.A. *et al.* (2014a) Dynamical and topological robustness of the mammalian cell cycle network: A reverse engineering approach. *Biosystems*, **115**, 23–32.

Ruz,G.A. *et al.* (2014b) Neutral space analysis for a boolean network model of the fission yeast cell cycle network. *Biol. Res.*, **47**, 64.

Sánchez-Corrales,Y.-E. *et al.* (2010) The arabidopsis thaliana flower organ specification gene regulatory network determines a robust differentiation process. *J. Theor. Biol.*, **264**, 971–983.

Singh,A. *et al.* (2012) Boolean approach to signalling pathway modelling in hgf-induced keratinocyte migration. *Bioinformatics*, **28**, i495–i501.

Tarjan,R. (1972) Depth-first search and linear graph algorithms. *SIAM J. Comput.*, **1**, 146–160.

Thomas,R. (1973) Boolean formalization of genetic control circuits. *J. Theor. Biol.*, **42**, 563–585.