Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Informática y Ciencias de la Computación

Dynamically Equivalent Linear Networks

por

Benjamín Rodrigo Schleef Sepúlveda

Tesis presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al grado de Magíster en Ciencias de la Computación

Profesora Guía: Dra. Lilian Salinas (DIICC, UdeC)

Abril de 2026,
Concepción, Chile.

Acknowledgements

Agradezco a mi familia, en especial a mi madre, Lorena, a mi hermana, Isabel, y a mis abuelos, Néstor y Laura, por su apoyo incondicional y constante.

A mi profesora guía, por su orientación, compromiso y permanente buena disposición en el desarrollo de esta tesis.

Contents

1 Introduction **1**

2 Definitions and notations **4**
2.1 Interaction digraph 4
2.2 Labeled digraph 6

3 Literature review **8**
3.1 Parallel digraph 8
3.2 DEN problem 10

4 L-DEN fixing f and s **12**

5 L-DEN problem **18**
5.1 Non-trivial dynamically equivalent networks 18
5.1.1 Acyclic case 20
5.1.2 Strongly connected case 25
5.1.3 Extended findings 28
5.2 Trivial dynamically equivalent networks 34
5.3 Algorithm for L-DEN problem 38

6 L-DEN fixing f and h **43**
6.1 Solve-and-Force algorithm 52
6.1.1 Complexity, confidence and examples 55
6.2 Block-Iteration algorithm 66
6.2.1 Complexity and examples 70
6.3 Comparison between Block Iteration and Solve-and-Force Algorithms . 74

7 Conclusion **77**

A Force **80**

B Construct schedule **82**

List of Figures

2.1 Example of interaction graph and its local activation functions 5
2.2 Example of a labeled digraph (G, lab_s) 6
2.3 Example of forbidden cycle in reverse graph 7

3.1 Parallel digraph vs $G(f^s)$ 9
3.2 Parallel digraph vs $G(f^s)$ for a linear Boolean network 10

4.1 Figures of Example 4.1 and Example 4.2. 17
4.2 Two solutions of h, given f and s. 17

5.1 Counterexample of Lemma 3.1 in linear Boolean networks 21
5.2 Example of Corollary 5.1 22
5.3 Example of Corollary 5.2 23
5.4 Example of Corollary 5.3. 25
5.5 Example of Proposition 5.4 28
5.6 A different solution for L-DEN problem 30
5.7 A new solution with fewer blocks 34
5.8 A simple cycle C such that $N_f^+(C) \neq \emptyset$ and a h updated on two blocks. 35
5.9 Sink simple cycles of one vertex 38
5.10 Figures of Example 5.9 41

6.1 A condition based on forbidden cycle for the equation system. 45
6.2 Example of solution of system of linear equations. 48
6.3 Example of Algorithm 6.2. 49
6.4 Labeling of h and f 50
6.5 Example of an inconsistent equation system. 51
6.6 Execution time of Solve-and-Force. 56
6.7 Worst example of Solve-and-Force assigning positive labels first 59
6.8 Worst example of Solve-and-Force assigning positive labels first with n vertices. 60
6.9 Worst example of Solve-and-Force assigning negative labels first 63
6.10 Worst case with heuristic 65
6.11 Example for T_1 definition, with $\{u, w\} \subseteq B_{k+1}$ and $\{u, z\} \subseteq B_{k+2}$. . . . 69
6.12 Example for T_2 definition, with $\{w\} \subseteq B_k^*$, $\{v, x\} \subseteq B_{k+1}$ and $\{u, z\} \subseteq B_{k+2}$. 70

6.13 Third example of Block-Iteration 72
6.14 An example of T_1 of Algorithm 6.7 73
6.15 An example of T_2 of Algorithm 6.7 74
6.16 Performance comparison of Force-and-Solve vs. Block-Iteration. 75
6.17 Execution time to decide if there is no solution of Force-and-Solve and Block-Iteration 76

Chapter 1

Introduction

A discrete dynamical system is a mathematical model that describes how variables evolve over time. When these variables take values of 0 or 1, we are in the presence of a Boolean network. Boolean networks were introduced by Kauffman (1969) to address the study of gene regulatory networks, where genes, in this case, are represented by variables, which can be inhibited (0) or activated (1), and their interaction are given by these Boolean functions, which in this context are called local activation functions.

One way to perturb a Boolean network is by changing the time at which each variable is updated. That is, it is no longer just allowed to update all the variables simultaneously (parallel scheme), but now different variables could be updated sequentially (block-sequential scheme).

Robert (1995) defined the Gauss-Seidelization process on Boolean networks as a sequential network update operator, which he used to study the dynamics and convergence to fixed points and limit cycles under different update schedules. Currently, the concept of sequential vertex updating has been expanded to block-sequential update schedules. To analyze these schemes, a tool known as the parallel digraph is used, which allows us to calculate the potential dependencies between the variables.

Recent studies have emerged on update schedules in Boolean networks. Aracena et al. (2013) prove that deciding the existence of two update schedules with different dynamics is an NP-complete problem. Moreover, Aracena et al. (2009) study the change in network dynamics under different update schedules, more precisely, they introduced the labeled digraph and the equivalence classes of update schedules and prove that all schedules belonging to the same equivalence class generate the same dynamic behavior.

Furthermore, there are approaches that study the behavior of update schemes on certain families of Boolean networks. For example, Goles and Noual (2012) classify disjunctive networks according to the changes in dynamics under different update schemes.

Cabrera (2024) introduces the Dynamically Equivalent Networks (DEN) problem, which consists of finding a Boolean network and an update schedule that, when this network is updated under this scheme, results in the same dynamics as another given Boolean network. Also, Cabrera (2024) demonstrates that the DEN problem is NP-Hard, but if the problem is restricted to disjunctive networks, it can be solved in

polynomial time.

An interesting feature of disjunctive networks is that the interaction graph of the resulting network after being updated under a block-sequential schedule is equal to the parallel digraph, since the presence of the variable in the activation function is sufficient to depend on it. This is not always the case, as in the general scenario, the interaction graph of the updated network is contained in the parallel digraph, and calculating the actual dependencies of each variable is an NP-hard problem. A similar situation occurs in Boolean networks with local activation functions based on addition modulo 2, known as linear Boolean networks. Although the interaction graph of the updated network is also contained in the parallel digraph (meaning that both graphs may differ), the dependencies can be computed in polynomial time, where they depend on the parity in which each variable appears. This motivates us to define the analogous problem of finding a linear Boolean network h and an update schedule s that is dynamically equivalent to the original linear Boolean network f, that is, $h^s = f$.

Analogously to the subproblems proposed for Dynamically Equivalent Disjunctive Networks (D-DEN) in the work of Cabrera (2024), this document presents two subproblems for Dynamically Equivalent Linear Networks (L-DEN). The first one consists of fixing the linear Boolean network f and an update scheme s, and determining whether there exists another linear Boolean network h such that (h, s) is dynamically equivalent to f. It will be shown that, in order to construct the linear Boolean network h, it is convenient to represent the dynamics of the original network using matrix operations, which reduces the problem to solving systems of linear equations with variables belonging to the field $\mathbb{F}_2$, which will be defined later.

In this way, Elspas (1959) studied how to analyze and synthesize the dynamic behavior of linear sequential networks using matrix algebra over finite fields. Besides, Qi and Cheng (2009) introduced the Semi-Tensor Product (STP) which generalizes standard matrix multiplication to study the dynamics of Boolean networks, given that it allows to convert logical equations into standard algebraic matrix equations.

In the second subproblem, the objective is to find an update scheme s given two Boolean networks f and h, such that (h, s) is dynamically equivalent to f. To this end, two algorithms that solve the problem are presented. The first one, Solve-and-Force, is based on the construction of a system of equations whose unknown variables correspond to the labels of the digraph h, together with the algorithm `Force` introduced by Palma et al. (2015), which labels arcs in such a way that the digraph is update. The second algorithm, Block-Iteration, operates by iteratively constructing the blocks of the update schedule according to specific criteria (which will be defined later).

It will be shown that both algorithms run in polynomial time when a solution is found in the first branch. Accordingly, simulations are presented to show the proportion of cases in which a solution is found in the first branch. Although this proportion reaches 100% for each network size in the simulations, atypical cases are identified in which a solution is not found in the first branch.

Regarding to applications of linear networks, one of them is in solvability problems, where the goal is to determine whether a given network can simultaneously transmit a

message to multiple destinations. Some works in this direction include Gadouleau et al. (2014) and Li et al. (2011).

In addition, other researchers have concentrated on exploring the dynamics of linear Boolean networks, Ho (2009) proves that under certain conditions the network has a unique fixed point as its attractor. Then, Noual et al. (2012) studies the convergence of linear networks in terms of the number of iterations under the parallel schedule. Recently, Chandrasekhar et al. (2023) studied properties of the dynamics of linear networks, more precisely, they managed to establish formulas for the average number of fixed points in linear systems (and therefore also for linear Boolean networks) and for the number of periodic states within the network, where they also conclude that results on fixed points remain consistent when changing the update schedule.

This work is divided into 7 chapters, with the first being the introduction. In Chapter 2, definitions and notations that are used throughout the document are introduced. Chapter 3 focuses on results from literature that are extensively used. Chapter 4 presents the L-DEN problem of fixing f and s, and a solution approach based on matrix representation of the network. Chapter 5 presents the main problem along with its solutions for two cases of the interaction graph of the network: acyclic and strongly connected; also in this chapter we present some results which can be useful to explore new ideas in future. In Chapter 6 L-DEN problem is defined fixing both linear Boolean networks f and h and two algorithms that solve the problem. Finally, Chapter 7 contains the conclusions.

Chapter 2

Definitions and notations

A Boolean network with n components is a discrete dynamical system usually defined by a global transition function:

$$f : \mathbb{B}^n \to \mathbb{B}^n,\ x \to f(x) = (f_1(x), \ldots, f_n(x)),$$

where $\mathbb{B} := \{0, 1\}$ and each function $f_u : \mathbb{B}^n \to \mathbb{B}$ associated to the component u is called *local activation function*.

Any vector $x = (x_1, \ldots, x_n) \in \mathbb{B}^n$ is called a *state* of the network f with local state x_u on each component u.

An *update schedule* is a function $s : \{1, \ldots, n\} \to \{1, \ldots, n\}$, where there exists $m \in \{1, \ldots, n\}$ such that $s(\{1, \ldots, n\}) = \{1, \ldots, m\}$. Using this definition, we can write $s = B_1 \cdots B_m$, where each set B_j is a block of s, and $B_j = s^{-1}(\{j\}) = \{u \in \{1, \ldots, n\} : s(u) = j\}$. If $m < n$, s is a *block-sequential scheme*; if $m = n$, s is a *sequential scheme*; if $m = 1$, s is the parallel or synchronous scheme.

Then, the dynamic behavior of $f : \mathbb{B}^n \to \mathbb{B}^n$, where $\forall j \in \{1, \ldots, n\}, f_j : \mathbb{B}^n \to \mathbb{B}$, with scheme s, is described by $x(t + 1) = f^s(x(t))$, with:

$$f^s = f^{B_m} \circ f^{B_{m-1}} \circ \cdots \circ f^{B_1}, \tag{2.1}$$

where $\forall i \in \{1, \ldots, m\}, \forall x \in \mathbb{B}^n, \forall j \in \{1, \ldots, n\}$:

$$f_j^{B_i}(x) = \begin{cases} f_j(x) & \text{if } j \in B_i, \\ x_j & \text{if } j \notin B_i. \end{cases} \tag{2.2}$$

2.1 Interaction digraph

We say that $f_v : \mathbb{B}^n \to \mathbb{B}^n$ depends on variable x_u if there exists $x \in \mathbb{B}^n$ such that,

$$f_v(x_1, \ldots, x_{u-1}, 0, x_{u+1}, \ldots, x_n) \neq f_v(x_1, \ldots, x_{u-1}, 1, x_{u+1}, \ldots, x_n).$$

Let $f : \mathbb{B}^n \to \mathbb{B}^n$ be a Boolean network, then its interaction digraph $G(f) = (V(f), A(f))$ is defined by:

$$V(f) = \{1, \ldots, n\}$$
$$A(f) = \{(u, v) : f_v \text{ depends on the variable } x_u\}.$$

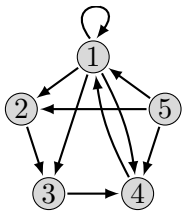
In addition, for each $u \in \{1, \ldots, n\}$ we define the in-neighborhood and the out-neighborhood of u as:

$$N_f^-(u) = \{v \in V(f) : (v, u) \in A(f)\}$$
$$N_f^+(u) = \{v \in V(f) : (u, v) \in A(f)\}.$$

Furthermore, the in-degree and the out-degree of u are defined as:

$$d^-(u) = |N_f^-(u)|, \quad d^+(u) = |N_f^+(u)|$$

Example 2.1. Let $f : \{0,1\}^5 \to \{0,1\}^5$ be a Boolean network and $G(f)$ its interaction graph shown in Figure 2.1:



$$f_1(x) = (x_1 \wedge x_4) \vee x_5$$
$$f_2(x) = x_1 \wedge x_5$$
$$f_3(x) = x_1 \vee x_2$$
$$f_4(x) = x_1 \veebar x_3 \veebar x_5$$
$$f_5(x) = 0$$

Figure 2.1: Example of interaction graph and its local activation functions

A *path* in the interaction graph $G(f)$ is a sequence of vertices and arcs without repetition. A *cycle* is a closed path. A *chord* in a cycle is an arc defined by two non-consecutive vertices. A *simple cycle* is a cycle without chords.

A *disjunctive Boolean network* is a Boolean network such that each local activation function is an OR function, and a *linear Boolean network* when its local activation functions f_i are:

$$f_i(x_{i_1}, \ldots, x_{i_l}) = x_{i_1} + \cdots + x_{i_l} \tag{2.3}$$

where $+$ is the addition modulo 2.

Notice that (2.3) can also be written as:

$$f_i(x_1, \ldots, x_n) = \sum_{j=1}^{n} c_{i,j} x_j$$

where $c_{i,j} \in \{0, 1\}$, and analogously the in-neighborhood of $i \in V(f)$ can be established as:

$$N_f^-(i) = \{j : c_{i,j} = 1\} \tag{2.4}$$

In this way, we have:

$$f_i(x_1, \ldots, x_n) = \sum_{j \in N_f^-(i)} x_j \tag{2.5}$$

Which is equivalent to:

$$N_f^-(i) = \bigtriangleup_{j \in N_f^-(i)} \{j\} \tag{2.6}$$

Remark 2.1. Since the symmetric difference operation is commutative and associative, the following holds for any finite collection of sets:

$$\bigtriangleup_{i \in I} A_i = \{x : |\{i \in I : x \in A_i\}| \text{ is odd}\}$$

where I is a finite set of indexes.

Remark 2.2. Given that it is possible to define each local activation function according to the input neighborhood of each vertex associated with the interaction graph, working with the linear Boolean network is equivalent to working with the interaction graph of that network.

2.2 Labeled digraph

Definition 2.1. Let $G = (V, A)$ be a digraph and s an update schedule. We define the label function $\text{lab}_s : A \to \{\ominus, \oplus\}$ for all $(u, v) \in A$:

$$\text{lab}_s(u, v) = \begin{cases} \oplus & \text{if } s(u) \geq s(v) \\ \ominus & \text{if } s(u) < s(v) \end{cases}$$

An arc $a \in A$ such that $\text{lab}_s(a) = \oplus$ is called an arc with positive labeling and an arc $a \in A$ such that $\text{lab}_s(a) = \ominus$ is called an arc with negative labeling. We define the labeled digraph (G, lab_s) labeling each arc of G by a label function lab_s (see Figure 2.2).

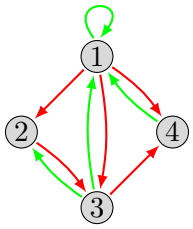


Figure 2.2: Example of a labeled digraph (G, lab_s), where $s = \{1\}\{2\}\{3\}\{4\}\{5\}$

For easy notation, each positive labeled arc is painted green, and each negative labeled arc is painted red.

Definition 2.2. A labeled digraph (G, lab) is an *update digraph* if there exists an update schedule s such that for each $(u, v) \in A(G)$ we have:

$$\text{lab}(u, v) = \ominus \iff s(u) < s(v).$$

Remark 2.3. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a Boolean network and $s = B_1 \dots B_m$ an update schedule. We define the label function $\text{lab}_s : A(f) \to \{\ominus, \oplus\}$ by:

$$\forall (u, v) \in A(f), \text{lab}_s(u, v) = \ominus \iff s(u) < s(v)$$

Then $(G(f), \text{lab}_s)$ is an update digraph. For this work, we simply use $G(f, s)$.

Definition 2.3. Let (G, lab) be a label digraph, we define the *reverse digraph* (G^r, lab^r) where G^r is defined by:

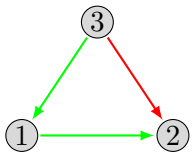
$$V(G^r) = V(G)$$
$$A(G^r) = \{(u, v) : (v, u) \in A(G) \wedge \text{lab}(v, u) = \ominus\} \cup \{(u, v) : (u, v) \in A(G) \wedge \text{lab}(u, v) = \oplus\}$$

and lab^r is defined by:

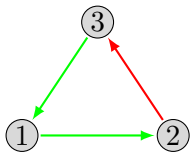
$$\text{lab}^r(u, v) = \begin{cases} \ominus & \text{if } (v, u) \in A(G) \wedge \text{lab}(v, u) = \ominus \\ \oplus & \text{otherwise} \end{cases}$$

With the previous definition, a reverse path is a path in the reverse graph. A forbidden cycle is a cycle in the reverse graph with an arc labeled with $\ominus$. Montalva (2011) proved that labeled digraph is an update digraph if and only if there does not exists a forbidden cycle in the reverse graph.

Example 2.2. Let (G, lab) be a labeled digraph as in Figure 2.3. Note that in this example, there is a forbidden cycle in (G^r, lab^r), so (G, lab) is not an update digraph.



(a) (G, lab)



(b) (G^r, lab^r)

Figure 2.3: Example of forbidden cycle in reverse graph

Chapter 3

Literature review

This section covers the properties explored in the work of Aracena et al. (2009) and Montalva (2011) based on the labeled digraph. In addition, we introduce the definition of the parallel digraph with some insights that motivate the investigation of our central topic. Finally, the general problem applied to Boolean networks and disjunctive Boolean networks will be introduced, together with the results of Cabrera (2024).

In this way, given $f : \{0,1\}^n \to \{0,1\}^n$ a Boolean network, we can define the equivalence relation $\sim_f$ in the set of block-sequential update schedules of n elements (denoted by S_n) in the following way:

$$\forall s, s' \in S_n,\ s \sim_f s' \iff G(f,s) = G(f,s')$$

The equivalence class of $s \in S_n$ is defined as:

$$[s]_f = \{s' \in S_n : s' \sim_f s\}.$$

Aracena et al. (2009) proves that if $s \sim_f s'$, then $f^s = f^{s'}$

Montalva (2011) proves the next theorem which is used in Chapter 6 to show the existence of a update digraph.

Theorem 3.1. *Let be G a digraph and G' a subdigraph of G. If (G', lab') is an update digraph, then there exists a labeled function* lab *over $A(G)$ such that (G, lab) is an update digraph and* $\mathrm{lab}\,|_{A(G')} = \mathrm{lab}'$.

3.1 Parallel digraph

The next definition (used in the work of Cabrera (2024)) is related to the arcs of the interaction digraph of an updated network under a block-sequential scheme.

Definition 3.1. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a Boolean network and $s = B_1 \dots B_k$ an update schedule, the *potential dependencies digraph of the equivalent parallel network*

(in short, *parallel digraph*), denoted as $G_P(f,s) = (V(f), A_P)$ is defined as:

$$\forall v \in B_1,\ (u,v) \in A_P \iff (u,v) \in A(f) \tag{3.1}$$

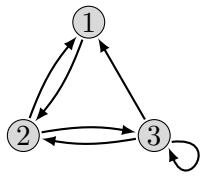
$$\forall v \notin B_1,\ (u,v) \in A_P \iff (\exists w \in N_f^-(v),\ s(w) < s(v) \wedge (u,w) \in A_P) \vee ((u,v) \in A_P \wedge s(u) \geq s(v)) \tag{3.2}$$

which is equivalent to:

$$(u,v) \in A_P \iff [(\exists w \in N_f^-(v), s(w) < s(v) \wedge (u,w) \in A_P) \vee ((u,v) \in A(f) \wedge s(u) \geq s(v))]$$

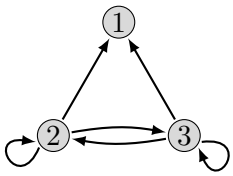
Note that equations (3.1) and (3.2) are well defined, since the arc (u,v) is constructed from the arc (u,w), which has previously been constructed in A_P.

The parallel digraph calculates the potential dependencies of the variables, which may not be the ones with effective dependencies when performing the actual calculation of the function under the update schedule.



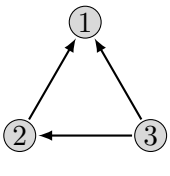
$f_1(x) = x_2 \wedge \neg x_3$
$f_2(x) = x_1 \vee \neg x_3$
$f_3(x) = x_2 \wedge x_3$
$s = \{1\}\{2\}\{3\}$

(a) A Boolean network f and an update schedule s



$f_1^s(x) = x_2 \wedge \neg x_3$
$f_2^s(x) = (x_2 \wedge \neg x_3) \vee \neg x_3$
$f_3^s(x) = ((x_2 \wedge \neg x_3) \vee \neg x_3) \wedge x_3$

(b) The parallel digraph $G_P(f,s)$



$f_1^s(x) = x_2 \wedge \neg x_3$
$f_2^s(x) = \neg x_3$
$f_3^s(x) = 0$

(c) The effective network f^s

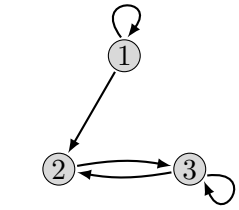
Figure 3.1: Parallel digraph vs $G(f^s)$

For example, in Figure 3.1b, we observe that f_3 is written in terms of the variables x_2 and x_3; however, when calculating the actual dependencies, f_3 does not depend on any variable, that is, it is constant. For this reason, in the general case of a Boolean network, it follows that $G(f^s) \subseteq G_P(f,s)$.

In the same way, since the OR function is closed under compositions, if we consider a disjunctive Boolean network and an update schedule, the parallel digraph is equal to the network updated under an update scheme. Therefore, given $f, h : \{0,1\}^n \to \{0,1\}^n$ two disjunctive Boolean networks and an update schedule s, then $h^s = f$ is equivalent to $G_P(h,s) = G(f)$.

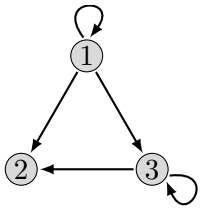
However, while the addition modulo 2 is also closed under compositions, the dependency of a variable in a function depends on how many times it appears, specifically whether the variable appears an odd number of times. Therefore, $G(f^s) \subseteq G_P(f,s)$.

In Figure 3.2b it is shown that x_3 appears an even number of times in f_3^s, thus it does not depend on this variable (Figure 3.2c).



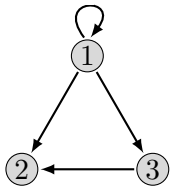
$f_1(x) = x_1$
$f_2(x) = x_1 + x_3$
$f_3(x) = x_2 + x_3$
$s = \{1, 2\}\{3\}$

(a) A linear Boolean network f and an update schedule s



$f_1^s(x) = x_1$
$f_2^s(x) = x_1 + x_3$
$f_3^s(x) = (x_1 + x_3) + x_3$

(b) The parallel digraph $G_P(f, s)$



$f_1^s(x) = x_1$
$f_2^s(x) = x_1 + x_3$
$f_3^s(x) = (x_1 + x_3) + x_3 = x_1$

(c) The effective network f^s

Figure 3.2: Parallel digraph vs $G(f^s)$ for a linear Boolean network

3.2 DEN problem

In this part of the chapter, we present the DEN and D-DEN problems, which were studied by Cabrera (2024). In addition, we discuss some of his results that are widely used in the development of this work.

The following definition will help us better formulate the problem:

Definition 3.2. Let $f, h : \{0,1\}^n \to \{0,1\}^n$ be two Boolean networks and s be an update schedule. We say that (h, s) is dynamically equivalent to f if $h^s = f$. Moreover, if $h \neq f$, or $h = f$ and $s \not\sim_h s_p$, we say that (h, s) and f are non-trivially dynamically equivalent.

Cabrera (2024) introduced the DEN (Dynamically Equivalent Networks) problem, which consists of finding a Boolean network with an update scheme that is non-trivially dynamically equivalent to another Boolean network. It was proved that this problem is NP-Hard. However, it was also proved that if the problem is restricted to disjunctive Boolean networks (D-DEN problem), it is possible to find its solution in polynomial time.

Some discoveries from Cabrera (2024) were useful for the completion of this work. One of them is the following theorem, which states that if there is no solution in two blocks for the DEN problem, then there will be no solution with a scheme of more than two blocks:

Theorem 3.2. *If there exists a solution to the DEN problem, then there exists a solution to the DEN problem with a block-sequential update schedule with two blocks.*

Several outcomes from Cabrera (2024) are based on the following lemma, which connects the labels of the arcs in the solution with the neighborhoods of the vertices in the original network:

Lemma 3.1. *Let $h, f : \mathbb{B}^n \to \mathbb{B}^n$ be two disjunctive Boolean networks and s an update schedule such that $h^s = f$. Then, for $u, v \in [n]$:*

$$[(u, v) \in A(h) \wedge \mathrm{lab}_s(u, v) = \ominus] \Longrightarrow N_f^-(u) \subseteq N_f^-(v)$$

The following result from Cabrera (2024) establishes necessary and sufficient conditions for the existence of a solution with only one arc with a negative label:

Proposition 3.1. *Let f a disjunctive Boolean network. There exists a disjunctive Boolean network h and an update schedule s, such that (h, s) is non-trivially dynamically equivalent to f and with only one negative arc $(u, v) \in A(h)$ if and only if the following conditions are satisfied:*

1. $N_f^-(u) \subseteq N_f^-(v)$
2. $u \notin N_f^-(u) \setminus N_f^-(v)$
3. *For every vertex $w \in N_f^-(u) \setminus N_f^-(v)$, there does not exist a path from u to w in $G(f) - v$.*

Chapter 4

L-DEN fixing f and s

In this chapter we address the following problem:

Dynamically equivalent linear Boolean network with fixed schedule problem (L-DENs)

Input: A linear Boolean network f and an update schedule s.

Question: Does there exist h a linear Boolean network such that $h^s = f$?

To address the above problem, we introduce an alternative way to model the dynamics of a linear Boolean network, which will later help us to set up systems of linear equations to find the dependencies of the solution network h.

In this way, given a linear Boolean network $f : \{0,1\}^n \to \{0,1\}^n$, and the finite field of $\{0,1\}$ denoted by $\mathbb{F}_2$, the dynamic of f can be expressed as follows:

$$f(x) = M(f)x$$

Where each element of $M(f) \in \mathcal{M}_{n\times n}(\mathbb{F}_2)$ is defined as:

$$M(f)_{i,j} = \begin{cases} 1 & \text{if } f_i \text{ depends on } x_j \\ 0 & \text{otherwise} \end{cases}$$

Now, if we consider a linear Boolean network like the one above, updated under a scheme $s = B_1 \dots B_m$, then the dynamics of the network over block k would be given by:

$$f^{B_k}(x) = M(f)^{B_k}x,$$

where the rows of the matrix $M(f)^{B_k} \in \mathcal{M}_{n\times n}(\mathbb{F}_2)$ are given by:

$$M(f)^{B_k}_{i,j} = \begin{cases} M(f)_{i,j} & \text{if } i \in B_k \\ 1 & \text{if } i \notin B_k \ \wedge \ i = j \\ 0 & \text{if } i \notin B_k \ \wedge \ i \neq j \end{cases} \tag{4.1}$$

Thus, resorting to concepts of linear transformations, more specifically to their composition, it follows that the dynamics of f^s is given by:

$$\begin{aligned} f^s(x) &= (f^{B_m} \circ f^{B_{m-1}} \circ \cdots \circ f^{B_1})(x) \\ &= (M(f^{B_m}) \cdots M(f^{B_1}))x \\ &= M(f^s)x, \end{aligned}$$

where $M(f^{B_m}) \cdots M(f^{B_1}) = M(f^s)$.

From this, in order to solve the proposed problem, a system of linear equations can be proposed for each entry of the matrix $M(h)$:

$$h(x) = M(h)x = \begin{pmatrix} z_{1,1} & \cdots & z_{1,n} \\ \vdots & \ddots & \vdots \\ z_{n,1} & \cdots & z_{n,n} \end{pmatrix} x$$

Here, each element $z_{i,j}$ is unknown in $\mathbb{F}_2$.

From now on, we consider the next indexing for the block $k \in \{1, \ldots, m\}$,

$$B_k = \{i : p_k \leq i \leq q_k\},$$

where $p_1 = 1$, $q_m = n$, and for all $k \in \{1, \ldots, m-1\}$, $q_k + 1 = p_{k+1}$.

With the above indexing, the matrix which models the dynamic behavior of h^{B_k} is given by:

$$M(h)^{B_k} = \begin{pmatrix} I^{q_{k-1}} & 0 & 0 \\ z_{p_k,p_1} & \cdots & z_{p_k,q_m} \\ \vdots & \ddots & \vdots \\ z_{q_k,p_1} & \cdots & z_{q_k,q_m} \\ 0 & 0 & I^{n-q_k} \end{pmatrix}$$

We note that for h to be a solution of $h^s = f$, it is necessary that $M(h^s) = M(f)$, which means:

$$M(h)^{B_m} \cdots M(h)^{B_1} = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{q_m,1} & \cdots & a_{q_m,n} \end{pmatrix} = M(f) \tag{4.2}$$

In this way, linear equations can be formulated for each element in different blocks.

The next lemma provides a useful formulation for the system on each block.

Lemma 4.1. *Let $f,\ h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean networks and $s = B_1 \ldots B_m$ a block sequential update schedule such that $h^s = f$. Then, given $k \in \{1, \ldots, m-1\}$, it follows,*

$$M(h)^{B_1} = M(f)^{B_1} \tag{4.3}$$

$$M(h)^{B_{k+1}} M(f)^{B_k^*} = M(f)^{B_{k+1}^*}, \tag{4.4}$$

where $B_k^* = \bigcup_{j=1}^{k} B_j$.

Proof. The proof is by induction on k.

Basis. For the first block, it follows,

$$M(h)^{B_1} = M(f)^{B_1}$$

Hypothesis of induction. Let us suppose that the statement is true for k, this is,

$$M(h)^{B_k} M(f)^{B_{k-1}^*} = M(f)^{B_k^*}$$

Inductive step. Due to $h^s = f$,

$$M(h^s) = M(h)^{B_m} M(h)^{B_{m-1}} \cdots M(h)^{B_1} = M(f)$$

By definition, considering $v \in B_{k+1}$, we have,

$$\forall l > k+1,\ \forall w \in [n],\ M(h)_{v,w}^{B_l} = I_{v,w}^n$$

where $I^n \in \mathcal{M}_{n\times n}(\mathbb{F}_2)$ is the identity matrix.

This implies that,

$$M(h)_v^{B_m} M(h)^{B_{m-1}} \cdots M(h)^{B_{k+2}} = I_v^n,$$

where $M(h)_v^{B_m}$ and I_v^n are the v-row of $M(h)^{B_l}$ and I^n, respectively.

Therefore,

$$\begin{aligned} M(f)_v &= M(h)_v^{B_m} M(h)^{B_{m-1}} \cdots M(h)^{B_{k+2}} M(h)^{B_{k+1}} \cdots M(h)^{B_1} \\ &= I_v^n M(h)^{B_{k+1}} \cdots M(h)^{B_1} \\ &= M(h)_v^{B_{k+1}} M(h)^{B_k} \cdots M(h)^{B_1} \end{aligned}$$

Given that,

$$\begin{aligned} M(h)^{B_k} \cdots M(h)^{B_1} &= M(h)^{B_k} M(f)^{B_{k-1}^*} \\ &= M(f)^{B_k^*} \qquad \text{(HI)} \end{aligned}$$

It follows,

$$M(h)_v^{B_{k+1}} M(f)^{B_k^*} = M(f)_v = M(f)_v^{B_{k+1}} = M(f)_v^{B_{k+1}^*}$$

Moreover,

$$\forall v \notin B_{k+1},\ M(h)_v^{B_{k+1}} = I_v^n$$

Therefore, we can conclude that,

$$M(h)^{B_{k+1}} M(f)^{B_k^*} = M(f)^{B_{k+1}^*}$$

□

Remark 4.1. From the proof of the above lemma and considering h as unknown, we get the next equations for the first block of s:

$$\forall u \in \{p_1, \ldots, q_1\},\ \forall j \in \{1, \ldots, n\},\ z_{u,j} = a_{u,j}$$

On the other hand, for $u \in B_{k+1}$, it follows from (4.4):

$$a_{u,l} = \begin{cases} \sum\limits_{j<p_{k+1}} a_{j,l} z_{u,j} & \text{if } l \in \{p_1, \ldots, q_k\} \\ \sum\limits_{j<p_{k+1}} a_{j,l} z_{u,j} + z_{u,l} & \text{if } l \in \{p_{k+1}, \ldots, q_m\} \end{cases}$$

Theorem 4.1. *Given a linear Boolean network $f : \{0,1\}^n \to \{0,1\}^n$, and an update schedule $s = B_1 \cdots B_m$. Then, L-DENs has solution if and only if there exists a linear Boolean network h with matrix $M(h)$ which satisfies the equations:*

$$\forall u \in \{p_1, \ldots, q_1\},\ \forall j \in \{1, \ldots, n\},\ z_{u,j} = a_{u,j}, \tag{4.5}$$

and for all $u \in B_k$:

$$a_{u,l} = \begin{cases} \sum\limits_{j<p_{k+1}} a_{j,l} z_{u,j} & \text{if } l \in \{p_1, \ldots, q_k\} \\ \sum\limits_{j<p_{k+1}} a_{j,l} z_{u,j} + z_{u,l} & \text{if } l \in \{p_{k+1}, \ldots, q_m\} \end{cases} \tag{4.6}$$

Proof. Note that equations (4.5) and (4.6) are derived from the system of equations established in Lemma 4.1, where f and s are fixed. Therefore, finding the value of $z_{u,j}$ in the system of equations is equivalent to determining whether h_u depends on x_j, that is, finding a linear Boolean network h that solves the L-DENs problem (if a solution exists). Otherwise, if the linear system of equations has no solution, it means that there does not exists a linear Boolean network h that satisfies (4.2), i.e., the L-DEN problem will have no solution. □

In the next example we observe a known interaction graph of a linear Boolean network that has no solution for a two-blocks update schedule.

Example 4.1. Let $f : \{0,1\}^4 \to \{0,1\}^4$ be a linear Boolean network, where its interaction graph is given in Figure 4.1a), and $s = \{1,2\}\{3,4\}$ be an update schedule. Note that $M(f)$ is given by,

$$M(f) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Now, let us see if there exists h linear Boolean network such that $h^s = f$. In this way, the unknown matrix would be given by,

$$M(h) = \begin{pmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} \\ z_{2,1} & z_{2,2} & z_{2,3} & z_{2,4} \\ z_{3,1} & z_{3,2} & z_{3,3} & z_{3,4} \\ z_{4,1} & z_{4,2} & z_{4,3} & z_{4,4} \end{pmatrix}$$

Applying the equations of the first block (4.5), which correspond to determining the dependencies of h_1 and h_2,we have that $h_1 = f_1$ and $h_2 = f_2$. On the other hand, the system for the vertices in the second block is set up as:

$$M(h^{B_2})M(h^{B_1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ z_{3,1} & z_{3,2} & z_{3,3} & z_{3,4} \\ z_{4,1} & z_{4,2} & z_{4,3} & z_{4,4} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = M(f)$$

From here, we have the next system of equations in order to determine the dependencies of h_3:

$$\begin{aligned} z_{3,2} &= 0 \\ 0z_{3,1} + 0z_{3,2} + 0z_{3,3} + 0z_{3,4} &= 1 \\ z_{3,3} &= 0 \\ z_{3,1} + z_{3,4} &= 0 \end{aligned}$$

Note that from the second equation of the system (which yields $0 = 1$), it is clear there does not exist a solution, this is, there does not exists a linear Boolean network h such that $h^s = f$. This result fits with what was obtained from the Corollary 5.4.

Example 4.2. Given an update schedule $s = \{1, 2\}\{3\}\{4\}$ and a linear Boolean network $f : \{0, 1\}^4 \to \{0, 1\}^4$ where its interaction graph is given in Figure 4.1b and its corresponding matrix of dependencies by:

$$M(f) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

In order to determine the linear Boolean network, we calculate from equation (4.5) the dependencies of vertices which are in the first block. This is, $h_1 = f_1$ and $h_2 = f_2$. Now, for the block B_2, we have:

$$M(h^{B_2})M(h^{B_1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ z_{3,1} & z_{3,2} & z_{3,3} & z_{3,4} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Here, the equations for h_3 are given by:

$$\begin{aligned} z_{3,1} &= 0 \\ z_{3,2} &= 1 \\ z_{3,1} + z_{3,2} + z_{3,3} &= 1 \\ z_{3,2} + z_{3,4} &= 0 \end{aligned}$$

Solving the system, it gives $(z_{3,1}, z_{3,2}, z_{3,3}, z_{3,4}) = (0, 1, 0, 1)$.

Finally, thanks to Lemma 4.1, it follows for h_4:

$$M(h^{B_3})M(h^{B_2})M(h^{B_1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ z_{4,1} & z_{4,2} & z_{4,3} & z_{4,4} \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

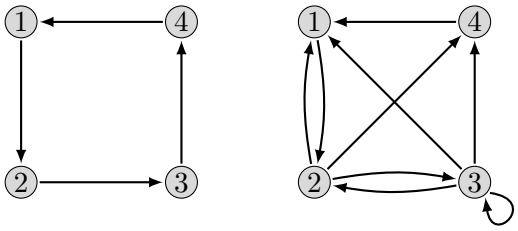
From the above, we have the next system of linear equations:

$$\begin{array}{rcrcrcl} & & z_{4,2} & & & = & 0 \\ z_{4,1} & + & & & z_{4,3} & = & 1 \\ z_{4,1} & + & z_{4,2} & + & z_{4,3} & = & 1 \\ z_{4,1} & + & & & & z_{4,4} = & 0 \end{array}$$

This gives the next solutions:

$$(z_{4,1}, z_{4,2}, z_{4,3}, z_{4,4}) = (1, 0, 0, 1) \ \vee \ (z_{4,1}, z_{4,2}, z_{4,3}, z_{4,4}) = (0, 0, 1, 0)$$

Both solutions are exposed in two interaction graphs of h in Figure 4.2.



(a) A simple cycle

(b) $G(f)$, where a solution exists

Figure 4.1: Figures of Example 4.1 and Example 4.2.

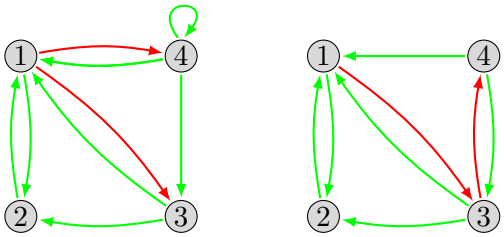


Figure 4.2: Two solutions of h, given f and s.

Chapter 5

L-DEN problem

5.1 Non-trivial dynamically equivalent networks

Throughout this section, we establish necessary and sufficient conditions for the existence of a linear Boolean network that, when updated under a scheme not equivalent to the parallel, results the same dynamics as another linear Boolean network.

A remarkable aspect in disjunctive Boolean networks, is that the parallel digraph updated under block-sequential schemes results equal to the interaction digraph of the updated network. This is not the case for linear Boolean networks, where the effective dependency depends on the parity in the number of times each variable appears (see Figure 3.2), which implies that the parallel digraph is included in the interaction graph of the updated network. This motivates us to define the following problem:

Dynamically equivalent linear Boolean networks problem (L-DEN)
Input: A linear Boolean network f.
Question: Does there exist a linear Boolean network h and s an update schedule, such that (h, s) is non-trivially dynamically equivalent to f?

The solutions proposed for the L-DEN problem in various forthcoming results consist of a two-block update scheme, which resolves the general case for more blocks. This holds because if Theorem 3.2 is restricted to the family of linear Boolean networks, it states that if there is no solution for the problem with a two-block schedule, then there is no solution with more than two blocks. Then there must necessarily exists a solution with a scheme that has two blocks.

The following definition will be useful to simplify the notation:

Definition 5.1. Given a Boolean network $f : \{0,1\}^n \to \{0,1\}^n$ and an update schedule s, we define the in-neighborhood with positive and negative labels for all $u \in V(f)$ as

follows:

$$\begin{aligned} N^-_{f,\ominus}(u) &= \{w \in N^-_f(u) : \mathrm{lab}_s(w,u) = \ominus\} \\ &= N^-_f(u) \cap \{z : s(z) < s(u)\} \\ N^-_{f,\oplus}(u) &= \{w \in N^-_f(u) : \mathrm{lab}_s(w,u) = \oplus\} \\ &= N^-_f(u) \cap \{z : s(z) \geq s(u)\} \end{aligned}$$

Analogously, the out-neighborhood with positive and negative labels are defined.

The property stated below is useful because it allows us to relate the neighborhoods of each vertex in the interaction digraph of an updated network to the neighborhoods of the variables that are updated before and after.

Property 5.1. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network and $s = B_1 \dots B_m$ be an update schedule. Then, for each $v \in V(f)$,*

$$N^-_{f^s}(v) = \left(N^-_f(v) \setminus N^-_{f,\ominus}(v)\right) \Delta \left(\bigtriangleup_{w \in N^-_{f,\ominus}(v)} N^-_{f^s}(w) \right) \tag{5.1}$$

Proof. Let $v \in B^k$, then by (2.1),

$$\begin{aligned} f^s_v(x) &= f^{B_k}_v(f^{B_{k-1}} \circ f^{B_{k-2}} \circ \cdots \circ f^{B_1})(x) \\ &= f^{B_k}_v(x') \end{aligned}$$

Where,

$$x'_w = \begin{cases} x_w & \text{if } s(w) \geq k \\ f^s_w(x) & \text{if } s(w) < k \end{cases}$$

Now, using (2.5) and Definition 5.1, we have:

$$\begin{aligned} f^s_v(x) &= \sum_{w \in N^-_f(v)} x'_w \\ &= \sum_{w \in N^-_{f,\oplus}(v)} x'_w + \sum_{w \in N^-_{f,\ominus}(v)} x'_w && (N^-_{f,\ominus}(v) = N^-_f(v) \cap \{z : s(z) < s(v)\}) \\ &= \sum_{w \in N^-_{f,\oplus}(v)} x_w + \sum_{w \in N^-_{f,\ominus}(v)} f^s_w(x) \\ &= \sum_{w \in N^-_{f,\oplus}(v)} x_w + \sum_{w \in N^-_{f,\ominus}(v)} \sum_{j \in N^-_{f^s}(w)} x_j && (\text{by (2.5)}) \end{aligned}$$

This is equivalent to:

$$
\begin{aligned}
N^-_{f^s}(v) &= \bigtriangleup_{w \in N^-_{f,\oplus}(v)} \{w\} \,\Delta\, \bigtriangleup_{w \in N^-_{f,\ominus}(v)} \left(\bigtriangleup_{j \in N^-_{f^s}(w)} \{j\} \right) \\
&= \bigcup_{w \in N^-_{f,\oplus}(v)} \{w\} \,\Delta\, \bigtriangleup_{w \in N^-_{f,\ominus}(v)} \left(\bigcup_{j \in N^-_{f^s}(w)} \{j\} \right) \\
&= N^-_{f,\oplus}(v) \,\Delta\, \bigtriangleup_{w \in N^-_{f,\ominus}(v)} N^-_{f^s}(w) \\
&= \left(N^-_f(v) \setminus N^-_{f,\ominus}(v) \right) \Delta \bigtriangleup_{w \in N^-_{f,\ominus}(v)} N^-_{f^s}(w)
\end{aligned}
$$

□

The following proposition is what makes the problem interesting to study, as it establishes that the graph of the updated network on linear Boolean networks can be computed in polynomial time.

Proposition 5.1. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network and s be an update schedule. Then, the interaction graph of f^s can be calculated in polynomial time.*

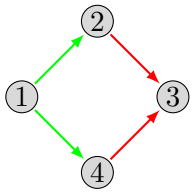
Proof. Considering the above proposition, we can compute $N^-_{f^s}(v)$ for each vertex v in $G(f,s)$ and compute the symmetric difference between two sets which, in the worst case, has a time complexity of $\mathcal{O}(n)$. In this way, assuming all arcs that a vertex v receives are negative, we need to operate the symmetric difference with all of them, which has a time complexity of $\mathcal{O}(n^2)$. Consequently, since this sequence of symmetric differences must be computed for every vertex, the interaction graph of f^s can be calculated in polynomial time, that is $\mathcal{O}(n^3)$. □

5.1.1 Acyclic case

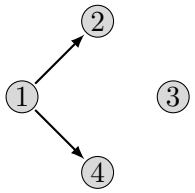
Notice that many of the results of Cabrera (2024) are based on Lemma 3.1, which establishes that for each negative labeled arc (u,v) in $G(h,s)$, $N^-_f(u) \subseteq N^-_f(v)$. However, this result does not hold for the case of linear Boolean networks, as shown in the following example.

Example 5.1. Consider $f, h : \{0,1\}^4 \to \{0,1\}^4$ two linear Boolean networks such that its interaction graph is represented in Figure 5.1, and an update schedule $s = \{1,2,4\}\{3\}$ such that $f = h^s$.

Notice that in this example, $(2,3) \in A(h)$ with $\mathrm{lab}(2,3) = \ominus$, but $N^-_f(2) = \{1\} \not\subseteq N^-_f(3) = \emptyset$.



(a) $G(h, s)$



(b) $G(f)$

Figure 5.1: Counterexample of Lemma 3.1 in linear Boolean networks

However, it is possible to establish analogously Proposition 3.1 presented by Cabrera (2024).

Proposition 5.2. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. There exists a linear Boolean network $h : \{0,1\}^n \to \{0,1\}^n$ and an update schedule s, such that (h, s) is non-trivially dynamically equivalent to f and with only one negative arc $(u, v) \in A(h)$ if and only if the following conditions are satisfied:*

1. $N_f^-(u) \subseteq N_f^-(v)$
2. $u \notin N_f^-(v) \setminus N_f^-(u)$
3. *For every vertex $w \in N_f^-(v) \setminus N_f^-(u)$, there does not exist a path from u to w in $G(f) - v$*

Proof. [$\Rightarrow$ **1.**] Let us suppose that there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f and with only one negative arc $(u, v) \in A(h)$. By contradiction, let us suppose that $N_f^-(u) \not\subseteq N_f^-(v)$, this is $N_f^-(u) \setminus N_f^-(v) \neq \emptyset$, then there exists a vertex $w \in V(h)$ such that $w \in N_f^-(u)$ and $w \notin N_f^-(v)$.

In this way, using (5.1) over the vertex v, we have:

$$N_{h^s}^-(v) = N_{h,\oplus}^-(v) \,\Delta \bigtriangleup_{z \in N_{h,\ominus}^-(v)} N_{h^s}^-(z)$$

Since, $N_{h,\ominus}^-(v) = \{u\}$,

$$\begin{aligned} N_{h^s}^-(v) &= N_{h,\oplus}^-(v) \,\Delta\, N_{h^s}^-(u) \\ &= N_{h,\oplus}^-(v) \,\Delta\, N_f^-(u) \quad (h^s = f) \end{aligned}$$

From here, $w \notin N_{h,\oplus}^-(v)$ and $w \in N_f^-(u)$, which imply that $w \in N_{h^s}^-(v) = N_f^-(v)$, leading to a contradiction.

Therefore, $N_f^-(u) \subseteq N_f^-(v)$.

[⇒ **2.**] The reasoning is analogous to Cabrera (2024).

[⇒ **3.**] The reasoning is analogous to Cabrera (2024).

[**1,2,3** ⇒ . . .] The reasoning is analogous to Cabrera (2024).

□

Although many of the implications are analogous, it is important to highlight that implication [⇒ 1.] is obtained without using Lemma 3.1, this is due to the hypothesis that there is only one negative arc in the network h.

Corollary 5.1. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. If there exist $u, v \in V(f)$ such that $N_f^-(u) = N_f^-(v)$, then there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.*

Proof. If there exist $u, v \in V(f)$ such that $N_f^-(u) = N_f^-(v)$, then the conditions of Proposition 5.2 are satisfied. □

Given an acyclic digraph, the next example shows that if there exist two vertices with empty in-neighborhood, then Corollary 5.1 is satisfied.

Example 5.2. Consider a linear Boolean network $f : \{0,1\}^5 \to \{0,1\}^5$, with its interaction graph shown in Figure 5.2a. Notice that $N_f^-(1) = N_f^-(2)$. Consequently, there exists an update schedule s and a linear Boolean network h represented in Figure 5.2b.

(a) $G(f)$ **(b)** $G(h, s)$

Figure 5.2: Example of Corollary 5.1

Remark 5.1. Notice that if $G(f)$ would be an acyclic disconnected digraph, there always exist two vertices such that its in-neighborhood is empty.

Given a disconnected interaction digraph, the next corollary shows under certain conditions there exists a connected solution:

Corollary 5.2. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network such that $G(f)$ is disconnected. Let $\tilde{G} \subseteq G$ be a connected component such that $\tilde{G}$ is a simple cycle or a vertex. If there exists a vertex u in $V(f) \setminus V(\tilde{G})$ such that $d^-(u) = 0$, then there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.*

Proof. Let $u \in V(f) \setminus V(\tilde{G})$ be a vertex such that $N_f^-(u) = \emptyset$ and $v \in V(\tilde{G})$. It is easy to see that the conditions of Proposition 5.2 are satisfied:

1. $N_f^-(u) \subseteq N_f^-(v)$.
2. $u \notin N_f^-(v) \setminus N_f^-(u)$.
3. For every vertex $w \in N_f^-(v) \setminus N_f^-(u)$, there does not exist a path from u to w in $G(f) - v$, since $N_f(\tilde{G}) \setminus V(\tilde{G}) = (N_f^+(\tilde{G}) \cup N_f^-(\tilde{G})) \setminus V(\tilde{G}) = \emptyset$.

Thus, using Proposition 5.2, there exists a linear Boolean network $h : \{0,1\}^n \to \{0,1\}^n$ and an update schedule s, such that (h, s) is non-trivially dynamically equivalent to f and it has only one negative arc $(u, v) \in A(h)$. □

Example 5.3. Let $f : \{0,1\}^7 \to \{0,1\}^7$ be a linear Boolean network with its interaction graph shown in Figure 5.3a. Choosing $\tilde{G} = G(f)[\{5,6,7\}]$, which satisfies $N_f(\tilde{G}) \setminus V(\tilde{G}) = \varnothing$, and the vertex $u = 4$, where $N_f^-(4) = \emptyset$, as per the notation of the previous corollary; then there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f, as shown in Figure 5.3b.

(a) $G(f)$ **(b)** $G(h, s)$

Figure 5.3: Example of Corollary 5.2

The upcoming proposition proposes other conditions for the existence of a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f, which concludes with an interesting corollary that covers the remaining cases for constructing a solution for any acyclic digraph.

Proposition 5.3. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. If there exist $u \neq v$ such that $N_f^-(u) \cup \{u\} = N_f^-(v)$, then there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent f.*

Proof. Let $u \neq v$ such that $N_f^-(u) \cup \{u\} = N_f^-(v)$. If $u \in N_f^-(u)$, we have $N_f^-(u) \cup \{u\} = N_f^-(u) = N_f^-(v)$, then by Corollary 5.1, there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.

We now turn to the case where $u \notin N_f^-(u)$, then $N_f^-(u) \subsetneq N_f^-(v)$.

Let $s = V \setminus \{u\}, \{u\}$ be an update schedule, and let $h : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network such that $V(h) = V(f)$ and the arcs defined by the in-neighborhood of each vertex in the following way:

- For $w \in V \setminus \{u\}$, $N_h^-(w) = N_f^-(w)$
- For $u \in V$, $N_h^-(u) = \{u, v\}$

We proceed to show that $h^s = f$.

If $w \in V \setminus \{u\}$, then w is in the first block, therefore all the arcs in the in-neighborhood of w are positive, which leads to $N_{h^s}^-(w) = N_h^-(w) = N_f^-(w)$.

Furthermore, for $u \in V$, it follows that $N_{h,\ominus}^-(u) = \{v\}$, then using (5.1),

$$\begin{aligned} N_{h^s}^-(u) &= (N_h^-(u) \setminus N_{h,\ominus}^-(u)) \,\Delta\, N_{h^s}^-(v) \\ &= (N_h^-(u) \setminus N_{h,\ominus}^-(u)) \,\Delta\, N_f^-(v) \\ &= (\{u, v\} \setminus \{v\}) \,\Delta (N_f^-(u) \cup \{u\}) \\ &= \{u\} \,\Delta (N_f^-(u) \cup \{u\}) \qquad (u \notin N_f^-(u)) \\ &= N_f^-(u) \end{aligned}$$

Finally, $h^s = f$. □

Corollary 5.3. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. If $G(f)$ is acyclic and for all $u \neq v \in V(f)$, $N_f^-(u) \neq N_f^-(v)$, then there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent f.*

Proof. If $G(f)$ is acyclic and for all $u \neq v \in V(f)$, $N_f^-(u) \neq N_f^-(v)$, then there exists only one vertex u with in-degree equal to zero, otherwise there would be two vertices with the same in-neighborhood.

Similarly, $G(f) - u$ is acyclic, then there exists only one vertex v with in-degree zero in $G(f) - u$, otherwise there would be another vertex z in $G(f)$ such that $N_f^-(v) = N_f^-(z) = \{u\}$.

Then we have that $N_f^-(u) \cup \{u\} = N_f^-(v)$ and by Proposition 5.3, there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f. □

Lemma 5.1. *Given a linear Boolean network $f : \{0,1\}^n \to \{0,1\}^n$ such that $G(f)$ is acyclic and non-trivial, then L-DEN problem has always solution. Also, it is possible to find a solution in polynomial time.*

Proof. Since $G(f)$ is acyclic, we have two cases:

1. There exists $u \neq v \in V(f)$ such that $N_f^-(u) = N_f^-(v)$. In this case, we can apply Corollary 5.1 to get a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.
2. For all $u \neq v \in V(f)$, $N_f^-(u) \neq N_f^-(v)$, then the hypothesis of Corollary 5.3 are satisfied. Thus, there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.

Algorithm 5.1: AcyclicSolve

Input: A linear Boolean network f such that $G(f)$ is acyclic and not trivial
Output: A linear Boolean network h and an update schedule s, such that (h, s) is non-trivially dynamically equivalent to f.

$V(h) \leftarrow V(f)$
if $\exists u, v \in V(f), u \neq v \wedge N_f^-(u) = N_f^-(v)$ **then**
  $A(h) \leftarrow (A(f) \setminus \{(w, u) : w \in N_f^-(u)\}) \cup \{(v, u)\}$
else
  Let $u \neq v \in V(f)$ such that $d^-(u) = 0$ and $N_f^-(v) = \{u\}$
  $A(h) \leftarrow A(f) \cup \{(u, u), (v, u)\}$
end if
$s \leftarrow V(f) \setminus \{u\}, \{u\}$
return (h, s)

Considering the previous cases, we can present the Algorithm 5.1 to construct a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.

Here, AcyclicSolve can be done in polynomial time, as it only needs to verify the in-neighborhood of each vertex to construct the linear network h. □

The following is an example of the previous corollary:

Example 5.4. Let $f : \{0, 1\}^4 \to \{0, 1\}^4$ be a linear Boolean network represented in Figure 5.4 by $G(f)$ a path of length 3. Here the update schedule is $s = \{2, 3, 4\}\{1\}$ and as in the proof of Corollary 5.3, $u = 1$ and $v = 2$.

(a) $G(h, s)$ **(b)** $G(f)$

Figure 5.4: Example of Corollary 5.3.

Remark 5.2. It is worth noting that the directed path is a digraph in which the in-neighborhoods of each vertex are all incomparable. Therefore, if we restrict to the family of disjunctive Boolean networks, a solution dynamically equivalent to the original network does not exist (Cabrera (2024)).

5.1.2 Strongly connected case

In this section, the only result is the main one for addressing the general problem, as it allows for the formulation of a polynomial-time algorithm capable of finding strongly connected components different to a simple cycle to define a solution.

To simplify the notation in some results, the out-neighborhood of a cycle is defined as follows:

Definition 5.2. Given $f : \{0,1\}^n \to \{0,1\}^n$ a linear Boolean network and $C = [v_0, \ldots, v_{k-1}, v_k = v_0]$ a cycle in $G(f)$, we define $N_f^+(C) = \bigcup_{i=0}^{k-1} N_f^+(v_i) \setminus V(C)$.

Proposition 5.4. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. If there exists a simple cycle $C = [v_0, \ldots, v_{k-1}, v_k = v_0]$ in $G(f)$ such that $N_f^+(C) \neq \emptyset$, then there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent f.*

Proof. Let $C = [v_0, \ldots, v_{k-1}, v_k = v_0]$ be a simple cycle in $G(f)$ such that $N_f^+(C) \neq \emptyset$.

Let $s = V(C), V(f) \setminus V(C)$ be an update schedule and let $h : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network such that $V(h) = V(f)$ and its arcs defined by the in-neighborhood of each vertex in the following way:

- For $w \notin N_f^+(C)$, $N_h^-(w) = N_f^-(w)$
- For $w \in N_f^+(C)$, let $\{v_{i_1}, \ldots, v_{i_m}\} = V(C) \cap N_f^-(w)$, then

$$N_h^-(w) = \left(N_f^-(w) \,\Delta \left(\sum_{j=1}^{m} N_f^-(v_{i_j+1}) \right) \right) \cup \{v_{i_j+1}\}_{j=1}^m$$

Now, we proceed to show that $h^s = f$.

It is clear that for all $i \in \{0, \ldots, k-1\}$, $N_{h^s}^-(v_i) = N_h^-(v_i) = N_f^-(v_i)$, since each v_i is in the first block of s.

Also, for all $w \in B_2 \setminus N_f^+(C)$, $N_{h,\ominus}^-(w)$ is empty, because $N_h^-(w) \subseteq B_2$, hence $N_{h^s}^-(w) = N_h^-(w) = N_f^-(w)$.

Now, consider $w \in N_f^+(C)$, and notice from definition of h that $N_{h,\ominus}^-(w) = \{v_{i_j+1}\}_{j=1}^m$, then using (5.1), we have:

$$\begin{aligned}
N_{h^s}^-(w) &= \left(N_h^-(w) \setminus N_{h,\ominus}^-(w)\right) \Delta \left(\sum_{z \in N_{h,\ominus}^-(w)} N_{h^s}^-(z) \right) \\
&= \left(\left[\left(N_f^-(w) \,\Delta \left(\sum_{j=1}^{m} N_f^-(v_{i_j+1}) \right) \right) \cup \{v_{i_j+1}\}_{j=1}^m \right] \setminus \{v_{i_j+1}\}_{j=1}^m \right) \Delta \left(\sum_{j=1}^{m} N_{h^s}^-(v_{i_j+1}) \right) \\
&= \left(\left(N_f^-(w) \,\Delta \left(\sum_{j=1}^{m} N_f^-(v_{i_j+1}) \right) \right) \setminus \{v_{i_j+1}\}_{j=1}^m \right) \Delta \left(\sum_{j=1}^{m} N_{h^s}^-(v_{i_j+1}) \right)
\end{aligned}$$

From this point, given $v_{i_k} \in \{v_{i_j+1}\}_{j=1}^m$, we can deduce the following:

- If $v_{i_k} \in N_f^-(w)$, we have that $v_{i_k+1} \in \{v_{i_j+1}\}_{j=1}^m$, and since v_{i_k} is in a simple cycle, there exists only one vertex in $\{v_{i_j+1}\}_{j=1}^m$ (which is v_{i_k+1}) such that $v_{i_k} \in N_f^-(v_{i_k+1})$. Consequently, $v_{i_k} \notin N_f^-(w) \,\Delta\, N_f^-(v_{i_k+1})$, which imply that $v_{i_k} \notin N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right)$. Therefore,

$$N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right) = \left(N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right)\right) \setminus \{v_{i_k}\}$$

- Likewise, if $v_{i_k} \notin N_f^-(w)$, then $v_{i_k+1} \notin \{v_{i_j+1}\}_{j=1}^m$, which leads to $v_{i_k} \notin \bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})$, and consequently $v_{i_k} \notin N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right)$. Thus,

$$N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right) = \left(N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right)\right) \setminus \{v_{i_k}\}$$

From the above, it follows that,

$$\begin{aligned} N_{h^s}^-(w) &= \left(N_f^-(w) \,\Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right)\right) \Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right) \\ &= N_f^-(w) \,\Delta \left(\left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right) \Delta \left(\bigtriangleup_{j=1}^m N_f^-(v_{i_j+1})\right)\right) \\ &= N_f^-(w) \end{aligned}$$

Finally, $h^s = f$. □

Example 5.5. Consider a linear Boolean network $f : \{0,1\}^6 \to \{0,1\}^6$ with its interaction graph shown in Figure 5.5a. By choosing $V(C) = \{1,2,3,4\}$ with $N_f^+(C) \neq \emptyset$, the hypothesis of Proposition 5.4 is satisfied. Therefore, there exists a linear Boolean network h and an update schedule s such that (h,s) shown in Figure 5.5b such that is non-trivially dynamically equivalent to f.

Here, $s = \{1,2,3,4\}\{5,6\}$, and the arcs of $G(h)$ are given by $N_h^-(i) = N_f^-(i)$, $i \in \{1,2,3,4\}$, and for the vertices 5 and 6:

$$\begin{aligned} N_h^-(5) &= \left(N_f^-(5) \,\Delta \bigtriangleup_{i=1}^4 N_f^-(i)\right) \cup \{1,2,3,4\} \\ &= (\{1,2,3,4\} \,\Delta\{4\} \,\Delta\{1,5,6\} \,\Delta\{2\} \,\Delta\{3,5\}) \cup \{1,2,3,4\} \\ &= \{1,2,3,4,6\} \\ N_h^-(6) &= \left(N_f^-(5) \,\Delta\, N_f^-(2) \,\Delta\, N_f^-(4)\right) \cup \{2,4\} \\ &= (\{1,3,5\} \,\Delta\{1,5,6\} \,\Delta\{3,5\}) \cup \{2,4\} \\ &= \{2,4,5,6\} \end{aligned}$$

(a) $G(f)$ **(b)** $G(h, s)$

Figure 5.5: Example of Proposition 5.4 where the cycle is defined by the vertices {1,2,3,4}

Remark 5.3. Given the Proposition 5.4, many cases where no solution existed for disjunctive Boolean networks now have a solution, even where the in-neighborhoods of each vertex are not comparable. The only requirement is that for a simple cycle in the interaction graph of a linear Boolean network, it must have only one arc going to another vertex. This can be achieved, for example, in any non-trivial strongly connected digraph that is not a simple cycle. The cases presented by Cabrera (2024) (complete graphs, double cycles, and double chains with loops at the extremes) demonstrate the existence of a non-trivial network and update schedule dynamically equivalent to the original network where the in-neighborhoods of its interaction graph are non-comparable.

5.1.3 Extended findings

In this part of the document, we present results that may be useful for constructing a different solution to the L-DEN problem.

For this purpose, consider the next equivalent version of Property 5.1.

Let $h : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network and s an update schedule. Then, for each $u, v \in V(h)$,

$$\begin{aligned}(u,v) \in A(h^s) \iff\ & [(u,v) \in A(h) \wedge \operatorname{lab}(u,v) = \oplus] \\ & \veebar [|\{w : (u,w) \in A(h^s) \wedge (w,v) \in A(h) \wedge \operatorname{lab}(w,v) = \ominus\}| \text{ is odd}] \quad (5.2)\end{aligned}$$

The ensuing proposition establishes necessary conditions under which a vertex can only belong to the first block of a two-block update scheme:

Proposition 5.5. *Let $f, h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean networks such that $f = h^s$ with $s = B_1B_2$ and $|B_1| = n - 1$. Let $v \in V(f)$, if there exists $z \in N_f^-(v)$ such that $|N_f^+(z)| = 1$, then $v \notin B_2$.*

Proof. Let us suppose that there exists a $z \in N_f^-(v)$ such that $|N_f^+(z)| = 1$ and, by contradiction, $v \in B_2$ with $|B_2| = 1$. By definition, $(z, v) \in A(f)$ is equivalent to

$$[(z,v) \in A(h) \wedge \operatorname{lab}(z,v) = \oplus] \quad (5.3)$$

$$\veebar [|\{x : (z,x) \in A(f) \wedge (x,v) \in A(h) \wedge \operatorname{lab}(u,v) = \ominus\}| \text{ is odd}] \quad (5.4)$$

Note that (5.3) is not satisfied, because $z \in B_1$ and $v \in B_2$, which is $\text{lab}(z, v) = \ominus$ in G_h.

Furthermore, note that $\{x : (z, x) \in A(f) \wedge (x, v) \in A(h) \wedge \text{lab}(x, v) = \ominus\} = \emptyset$, which means that (5.4) is not satisfied neither.

In summary, (5.3) and (5.4) are not satisfied, then $(z, v) \notin A(f)$, which is a contradiction. □

The following statement establishes conditions under which it is possible to reduce the size of the second block in an update scheme of a given solution:

Proposition 5.6. *Let $f, h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean network such that $h^s = f$ with $s = B_1B_2$. If there exists $u \in B_2$ such that $\emptyset \neq N_h^-(u) \subseteq B_1$, then there exists a non-trivially (h', s') such that $h'^{s'} = f$, with $s' = B_1'B_2' = V \setminus \{u\}, \{u\}$.*

Proof. Consider the vertex $u \in B_2$ and the update schedule s' be as described in the statement. Define h' as a linear Boolean network where, for each $w \in B_1' = V \setminus \{u\}$, the in-neighborhood is given by $N_{h'}^-(w) = N_f^-(w)$, and for $u \in B_2'$, $N_{h'}^-(u) = N_h^-(u)$.

Since for each $w \in B_1'$, we have $N_{h',\ominus}^-(w) = \emptyset$, it follows that $N_{h's'}^-(w) = N_f^-(w) = N_{h^s}^-(w)$.

Now, consider $u \in B_2'$, then:

$$\begin{aligned} N_{h's'}^-(u) &= N_{h',\oplus}^-(u) \,\Delta \bigwedge_{v \in N_{h',\ominus}^-(u)} N_{h's'}^-(v) \\ &= N_{h,\oplus}^-(u) \,\Delta \bigwedge_{v \in N_{h,\ominus}^-(u)} N_{h's'}^-(v) && (N_{h'}^-(u) = N_h^-(u)) \end{aligned}$$

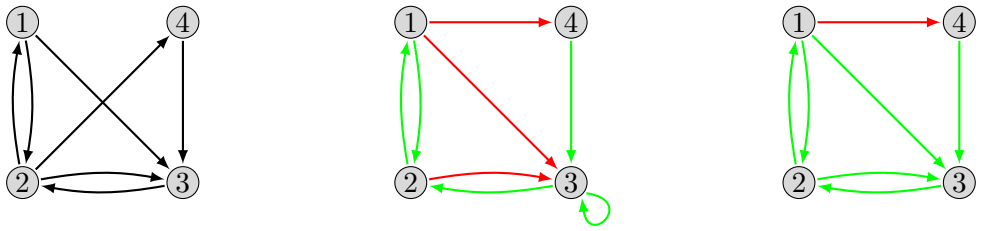
And given that $N_{h,\ominus}^-(u) = N_h^-(u) \subseteq B_1 \subseteq B_1'$, it follows:

$$\begin{aligned} N_{h's'}^-(u) &= N_{h,\oplus}^-(u) \,\Delta \bigwedge_{v \in N_{h,\ominus}^-(u)} N_{h^s}^-(v) \\ &= N_{h^s}^-(u) && \text{(definition of } h^s) \\ &= N_f^-(u) && (h^s = f) \end{aligned}$$

Therefore, $h'^{s'} = f$.

□

Example 5.6. Let $f : \{0,1\}^4 \to \{0,1\}^4$ be linear Boolean network defined by its interaction graph shown in Figure 5.6a , this example illustrates that while both $G(h, s)$ (Figure 5.6b) and $G(h', s')$ (Figure 5.6c) represent the same original network, $G(h, s) \neq G(h', s')$. The difference arises because, by applying Proposition 5.6 with the vertex $u = 4$, we obtain a distinct non-trivial solution, which is (h', s').



(a) $G(f)$ **(b)** $G(h, s)$, $s = \{1, 2\}\{3, 4\}$ **(c)** $G(h', s')$, $s' = \{1, 2, 3\}\{4\}$

Figure 5.6: A different solution for L-DEN problem

Definition 5.3. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. Let us define the in-neighborhood and out-neighborhood of a subset $U \subsetneq V(f)$:

$$N_f^-(U) = \bigcup_{v \in U} N_f^-(v) \setminus U$$
$$N_f^+(U) = \bigcup_{v \in U} N_f^+(v) \setminus U$$

The following proposition establishes conditions under which it is possible to extend an update schedule to additional blocks.

Proposition 5.7. *Let be two linear Boolean networks $f, h : \{0,1\}^n \to \{0,1\}^n$, and $s = B_1 \cdots B_m$ an update schedule such that (h, s) is non-trivially dynamically equivalent to f. If there exist a simple cycle C such that $V(C) \subsetneq B_m$, and the following conditions holds:*

1. $\forall v \in N_f^-(W) \cap U,\ v \notin N_f^-(C)$
2. $\forall v \in N_f^-(W) \cap V(C),\ v \notin N_f^-(U)$
3. $\forall v_k \in V(C) \cap N_f^-(W),\ \nexists w \in U,\ w \in N_f^-(v_{k+1})$
4. $N_f^+(C) \cap N_h^-(W) \cap U = \emptyset$

where $W = B_m \setminus V(C)$ and $U = V(f) \setminus B_m$.

Then, there exists (h', s') non-trivially dynamically equivalent to f such that $s' = B_1 \cdots B_{m-1} B'_m B'_{m+1}$, where $B'_m = V(C)$ and $B'_{m+1} = W$.

Proof. Considering the first hypothesis, let $v \in N_f^-(W) \cap U$ such that $v \notin N_f^-(C)$, then there exists a vertex w in $N_f^+(v) \cap U$ such that $w \in N_h^-(W)$, if not $v \notin N_f^-(W) = N_{h^s}^-(W)$.

On the other hand, by the second hypothesis, let $v_i \in N_f^-(W) \cap V(C)$ such that $v_i \notin N_f^-(U)$, this implies that there exists a $v_{i+1} \in N_f^+(v_i) \cap V(C)$ such that $v_{i+1} \in N_h^-(W)$.

From this, we can define $H = \{v_{i+1} \in V(C) : v_i \in N_f^-(W) \cap V(C)\}$, and using third hypothesis, for all $w \in U$, $w \notin N_f^-(H)$, which implies $N_{h^s}^-(H) \subseteq V(f) \setminus U$.

On the other hand, given the fourth hypothesis, it follows that for all $w \in U \cap N_h^-(W)$, $N_{h^s}^-(w) \subseteq V(f) \setminus V(C)$.

Now, we define an update schedule s' as in the statement, and a linear Boolean network h' such that $\forall v \in U \cup V(C),\ N_{h'}^-(v) = N_h^-(v)$. With this definition, it follows, $\forall v \in U \cup V(C),\ N_{h'^{s'}}^-(v) = N_{h^s}^-(v) = N_f^-(v)$.

Now, we will prove that we can find the in-neighborhood in h' for $v \in W$ solving a linear Boolean system.

With all of the above, we have proven that the dependencies of $G[W]$ in the previous blocks are disjoint between the vertices of $G[U]$ and C, meaning that, on one hand, the arcs coming from $G[U]$ to $G[W]$ only build dependencies of $G[U]$ and not of C; similarly, the negative arcs coming from C to $G[W]$ only build dependencies of C when updating the network.

Now, given that $N_{h',\ominus}^-(v)$ is fixed, let us show that $N_{h',\oplus}^-(v)$ can be constructed by solving a linear equation system.

Using Lemma 4.1 on block B'_{m+1}, it follows:

$$M(h')^{B'_{m+1}} M(f)^{{B'_m}^*} = M(f)$$

And given a $v \in B'_{m+1}$, we have,

$$M(h')_v^{B'_{m+1}} M(f)^{{B'_m}^*} = M(f)_v$$

where $M(h)_v^{B'_{m+1}}$ and $M(f)_v$ are the $v-$row of $M(h')^{B'_{m+1}}$ and $M(f)$, respectively.

Let us set $M(h')_v^{B_{m+1}} = (z_1 \cdots z_n)$ as the vector which models dependencies of v in h'.

Without loss of generality, let us suppose that $z_1, \ldots, z_k$ models dependencies of h_v with respect of ${B'_m}^*$ and $z_{k+1}, \ldots, z_n$ the dependencies of W. With the above, results the next system,

$$\begin{pmatrix} A & 0 \\ B & I \end{pmatrix} \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

Here,

$$(M(f)^{{B'_m}^*})^t = \begin{pmatrix} A & 0 \\ B & I \end{pmatrix},\ M(f)_v^t = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

Notice that $z_1, \ldots, z_k$ are fixed, then we only need to determine $z_{k+1}, \ldots, z_n$. This implies that we can reduce our matrix $(M(f)^{{B'_m}^*})^t$ in the system to,

$$\begin{pmatrix} B & I \end{pmatrix} \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} a_{k+1} \\ \vdots \\ a_n \end{pmatrix}$$

This last is equivalent to,

$$B\begin{pmatrix} z_1 \\ \vdots \\ z_k \end{pmatrix} + I\begin{pmatrix} z_{k+1} \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} a_{k+1} \\ \vdots \\ a_n \end{pmatrix}$$

Which is,

$$I\begin{pmatrix} z_{k+1} \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} a_{k+1} \\ \vdots \\ a_n \end{pmatrix} - B\begin{pmatrix} z_1 \\ \vdots \\ z_k \end{pmatrix}$$

Since the rank of the identity matrix is complete, there exists dependencies $z_{k+1}, \ldots, z_n$ of v in h', and they can be calculate by the above system. □

Proposition 5.8. *Let be two linear Boolean networks $f, h' : \{0,1\}^n \to \{0,1\}^n$, and $s' = B'_1 \cdots B'_m$ an update schedule such that (h', s') is dynamically equivalent to f. Then there exists (h, s) non-trivially dynamically equivalent to f such that s has fewer blocks than s'.*

Proof. From the fact that $h^s = f$ and since the matrix product is associative, give two consecutive indices $p, q \in \{1, \ldots m\}$, it follows,

$$\begin{aligned} M(f) &= M(h'^{s'}) \\ &= M(h')^{B'_m} \cdots M(h')^{B'_1} \\ &= M(h')^{B'_m} \cdots M(h')^{B'_q} M(h')^{B'_p} \cdots M(h')^{B'_1} \\ &= M(h')^{B'_m} \cdots M(h')^{B'_{qp}} \cdots M(h')^{B'_1}, \end{aligned}$$

where $M(h')^{B'_{qp}} = M(h')^{B'_q} M(h')^{B'_p}$.

More explicitly, defining $M(h')^{B'_q}$ and $M(h')^{B'_p}$ as the next matrices by blocks,

$$M(h')^{B'_p} = \begin{pmatrix} I & 0 & 0 & 0 \\ A_1 & A_2 & A_3 & A_4 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{pmatrix}, \; M(h')^{B'_q} = \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ B_1 & B_2 & B_3 & B_4 \\ 0 & 0 & 0 & I \end{pmatrix},$$

where I is the corresponding identity matrix, we have that $M(h')^{B'_{qp}}$ is given by,

$$\begin{aligned} M(h')^{B'_{qp}} &= \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ B_1 & B_2 & B_3 & B_4 \\ 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 & 0 \\ A_1 & A_2 & A_3 & A_4 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{pmatrix} \\ &= \begin{pmatrix} I & 0 & 0 & 0 \\ A_1 & A_2 & A_3 & A_4 \\ B_1 + B_2 A_2 & B_2 A_2 & B_2 A_3 + B_3 & B_2 A_4 + B_4 \\ 0 & 0 & 0 & I \end{pmatrix} \end{aligned}$$

From here, we can define an update schedule $s = B'_1 \cdots B'_{qp} \cdots B'_m$ and h a linear Boolean network by,

$$\forall j \in V \setminus B'_q,\ \forall u \in B_j,\ h_u = h'_u$$
$$\forall x \in \{0,1\}^n,\ \forall u \in B'_q,\ h_u(x) = \sum_{v:M(h')_{uv}^{B'_{qp}}=1} x_v$$

such that $h^s = f$.

□

Example 5.7. Let us consider $f, h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean networks, s be a block-sequential schedule. Here, the dependencies of f are given in Figure 5.7a, dependencies of h in Figure 5.7b. In this way, dependencies of h are represented in the next matrix:

$$h = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Then, by definition we have:

$$f = h^s = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Applying Proposition 5.8, we can define $s' = \{1\}\{2,3\}\{4\}$, where $B_{23} = \{2,3\}$ we can get the following dependencies for the vertices 2 and 3 according to h:

$$M(h)^{B_{32}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

From this, the new linear Boolean network h' is represented in Figure 5.7c.

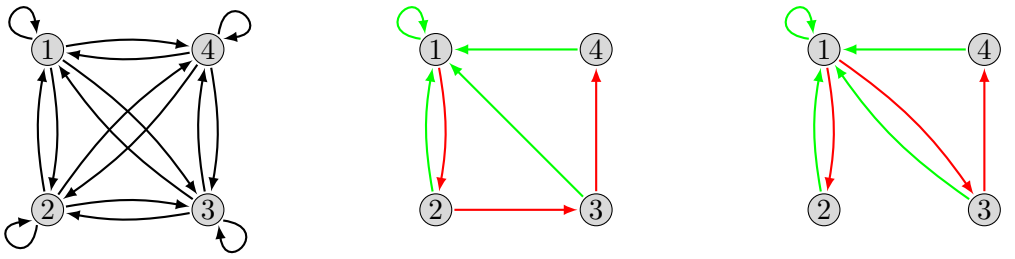
In this example, note that the only dependencies that change are those of vertex 3.

Remark 5.4. It is worth nothing that Proposition 5.8 can be applied multiple times to the same scheme. In particular, if the aforementioned proposition is applied to a solution consisting of a complete graph with loops and a sequential scheme, the

complete spectrum of solutions can be constructed by applying the proposition to any sequential schedule. In such a case, the number of solutions obtained is equal to the total number of update digraphs. This bound is demonstrated by Aracena et al. (2011), which corresponds to:

$$\{(G, \text{lab}) : (G, \text{lab}) \text{ is an update digraph}\} \leq T_n,$$

where, $T_n = \sum_{k=0}^{n-1} \binom{n}{k} T_k$, and $T_0 = 1$.



(a) $G(f)$ **(b)** $G(h, s)$, $s = \{1\}\{2\}\{3\}\{4\}$ **(c)** $G(h', s')$, $s' = \{1\}\{2, 3\}\{4\}$

Figure 5.7: A new solution with fewer blocks

5.2 Trivial dynamically equivalent networks

Within this section, we establish conditions on the architecture of the original network to conclude that the only possible linear Boolean network and update schedule are the trivial ones for the L-DEN problem.

Given two linear Boolean networks and an update scheme in two blocks, the following proposition provides a condition that must be met for the vertices of a simple cycle to be included in only one block of the scheme. This proposition will be useful later to prove that the only solution is the trivial one for different interaction graphs of a linear Boolean network with a simple cycle which does not have outgoing arcs.

Proposition 5.9. *Let $f, h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean networks and an update schedule $s = B_1 B_2$ such that $h^s = f$. If there exists a simple cycle C such that $N_f^+(C) = \emptyset$, then $V(C) \subseteq B_1$ or $V(C) \subseteq B_2$.*

Proof. Let $C = [v_0, \ldots, v_{m-1}, v_m = v_0]$ be a simple cycle.

By contradiction, let us assume that $V(C) \not\subseteq B_1$ and $V(C) \not\subseteq B_2$. With this, it is easy to see that $V(C) \cap B_1 \neq \emptyset$ and $V(C) \cap B_2 \neq \emptyset$.

Now define $j := \max\{i \in V(C) : v_i \in B_1\}$ and consider $v_{j+1} \in B_2$, applying (5.1), we have:

$$N^-_{h^s}(v_{j+1}) = N^-_{h,\oplus}(v_{j+1}) \,\Delta \left(\bigtriangleup_{z \in N^-_{h,\ominus}(v_{j+1})} N^-_{h^s}(z) \right) \tag{5.5}$$

Here, since $v_j \in B_1$, we have $v_j \notin N^-_{h,\oplus}(v_{j+1})$.

On the other hand, if there exists w in B_1 such that $(v_j, w) \in A(h)$, then $(v_j, w) \in A(f)$, which is a contradiction, because $N^+_f(C) = \emptyset$; this implies that $N^+_h(v_j) \subseteq B_2$.

In this way, due to $N^-_{h,\ominus}(v_{j+1}) \subseteq B_1$, it follows that,

$$\forall z \in N^-_{h,\ominus}(v_{j+1}), \qquad v_j \notin N^-_{h,\ominus}(z) \qquad (N^+_h(v_j) \subseteq B_2)$$

From the above and considering (5.5), we have that $v_j \notin N^-_{h^s}(v_{j+1}) = N^-_f(v_{j+1})$, which is a contradiction.

Consequently, $V(C) \subseteq B_1$ or $V(C) \subseteq B_2$. □

Remark 5.5. Note that the condition imposed in the previous proposition is necessary. As a counterexample, consider that there exists a linear Boolean network h and an update scheme $s = \{3, 4\}, \{1, 2\}$ such that they are non-trivially dynamically equivalent to a linear Boolean network f (given in Figure Figure 5.8a) with a simple cycle $V(C) = \{1, 2, 3\}$ that has an outgoing edge to vertex 4, that is, $N^+_f(C) = \{4\}$. From this, it is clear that $V(C) \not\subseteq B_1$ and $V(C) \not\subseteq B_2$.

(a) $G(f)$ **(b)** $G(h, s)$

Figure 5.8: A simple cycle C such that $N^+_f(C) \neq \emptyset$ and a h updated on two blocks.

From the proof of the above proposition, it is clear that the only possible solution (h, s) for a simple cycle is the trivial one.

Corollary 5.4. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. If $G(f)$ is a simple cycle, then the only solution for the L-DEN problem is the trivial one.*

Proof. Let be an update schedule $s = B_1B_2$ with two blocks and a linear Boolean network h such that $f = h^s$. Given that $G(f) = C$, Proposition 5.9 implies that either $V(C) \subseteq B_1$ or $V(C) \subseteq B_2$. This condition restricts the possible solutions to the trivial case only. □

Remark 5.6. In Cabrera (2024), it was proven that if a solution exists for the DEN problem, then a solution exists in two blocks (Theorem 3.2). This can also be stated using the contrapositive, which asserts that if no solution exists in two blocks for the DEN problem, then no solution exists for the DEN problem. This is equivalent to saying that if there is no update schedule in 2 or more blocks, then the only solution is the trivial (h, s) for the DEN problem.

Besides, it is important to highlight that Theorem 3.2 can be applied to the particular case of linear Boolean networks.

The proposition stated below provides a scenario concerning to the interaction digraph of a linear Boolean network, if the graph is connected and only has simple cycles sinks of the same vertex, then the only possible solution is the trivial one.

Proposition 5.10. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. Let $C_1, \ldots, C_m$ simple cycles of $G(f)$. If there exists only one vertex $u \in V(f) \setminus (V(C_1) \cup \cdots \cup V(C_m))$ such that $d^-(u) = 0$ and,*

$$\forall i \in \{1, \ldots, m\}, \exists v^i \in V(C_i), (u, v^i) \in A(f)$$

Then, the only solution for the L-DEN problem is the trivial one.

Proof. Let us show that there does not exist an update schedule $s = B_1B_2$ with two blocks and a linear Boolean network h such that $f = h^s$.

By Proposition 5.9, for all $i \in \{1, \ldots, m\}$, $V(C_i) \subseteq B_1$ or $V(C_i) \subseteq B_2$.

Then we have the next cases:

1. An arc between two cycles from different blocks. Without loss of generality, let $C_1 \subseteq B_1$ and $C_2 \subseteq B_2$, with $v_j^1 \in V(C_1)$ and $v_k^2 \in V(C_2)$. If $(v_j^1, v_k^2) \in A(h)$, then $\text{lab}(v_j^1, v_k^2) = \ominus$, and by (5.1), we have:

$$\begin{aligned} N^-_{h^s}(v_k^2) &= N^-_{h,\oplus}(v_k^2) \,\Delta \left(\bigtriangleup_{w \in N^-_{h,\ominus}(v_k^2)} N^-_{h^s}(w) \right) \\ &= (N^-_{h,\oplus}(v_k^2) \,\Delta\, N^-_{h^s}(v_j^1)) \,\Delta \left(\bigtriangleup_{w \in N^-_{h,\ominus}(v_k^2) \setminus \{v_j^1\}} N^-_{h^s}(w) \right) \end{aligned}$$

   Since $v_{j-1}^1 \in B_1$, we know $v_{j-1}^1 \notin N^-_{h,\oplus}(v_k^2)$. Furthermore, as all cycles are sinks of u, we have $v_{j-1}^1 \notin \bigtriangleup_{w \in N^-_{h,\ominus}(v_k^2) \setminus \{v_j^1\}} N^-_{h^s}(w)$. However, $v_{j-1}^1 \in N^-_{h^s}(v_j^1)$, which implies $v_{j-1}^1 \in N^-_{h^s}(v_k^2)$. This contradicts the fact that $N^-_{h^s}(v_k^2) \subseteq \{v_{k-1}^2, u\}$.

2. $u \in B_1$ and there exists $z \in B_2$ such that $(u, z) \in A(h)$ and $(u, z) \in A(f)$. Given

that $N^-_{h,\ominus}(z) = \{u\}$ and $N^-_{h^s}(u) = N^-_f(u) = \emptyset$, we can deduce:

$$\begin{aligned} N^-_{h^s}(z) &= N^-_{h,\oplus}(z) \,\Delta \left(\bigtriangleup_{w \in N^-_{h,\ominus}(z)} N^-_{h^s}(w) \right) \\ &= N^-_{h,\oplus}(z) \,\Delta\, N^-_{h^s}(u) \\ &= N^-_{h,\oplus}(z) \end{aligned}$$

As $u \notin N^-_{h,\oplus}(z)$, we conclude that $u \notin N^-_{h^s}(z) = N^-_f(z)$, which is a contradiction.

3. $u \in B_1$ and there exists $z \in B_2$ such that $(u,z) \in A(h)$ and $(u,z) \notin A(f)$. Considering $z \in C_i$, then for all $w \in C_i$ such that $(u,w) \in A(f)$ and $(u,w) \in A(h)$, we have $\text{lab}(u,w) = \ominus$, which leads to a contradiction given that we are in the same situation of previous case.

4. $u \in B_2$ and assume there exists a $v^i_j \in B_1 \cap V(C_i)$ such that $(v^i_j, u) \in A(h)$. We know that $v^i_{j-1} \in N^-_{h^s}(v^i_j)$ and $v^i_{j-1} \notin \bigtriangleup_{w \in N^-_{h,\ominus}(u) \setminus \{v^i_j\}} N^-_{h^s}(w)$. Additionally, since $v^i_{j-1} \in B_1$, we have $v^i_{j-1} \notin N^-_{h,\oplus}(u)$. Applying (5.1), we obtain:

$$\begin{aligned} N^-_{h^s}(u) &= N^-_{h,\oplus}(u) \,\Delta \left(\bigtriangleup_{w \in N^-_{h,\ominus}(u)} N^-_{h^s}(w) \right) \\ &= (N^-_{h,\oplus}(u) \,\Delta\, N^-_{h^s}(v^i_j)) \,\Delta \left(\bigtriangleup_{w \in N^-_{h,\ominus}(u) \setminus \{v^i_j\}} N^-_{h^s}(w) \right) \end{aligned}$$

This implies $v^i_{j-1} \in N^-_{h^s}(u) = N^-_f(u)$, contradicting the fact that $N^-_f(u) = \emptyset$.

For all the above reasons, there does not exist an arc labeled negative in h, which means that there does not exist a non-trivially (h,s) with two blocks, and for Theorem 3.2, there does not exist a non-trivially (h,s) with two or more blocks.

Therefore, the only solution to be dynamically equivalent to f is the trivial one. □

Example 5.8. Given a linear Boolean network $f : \{0,1\}^7 \to \{0,1\}^7$ where its interaction graph is represented in Figure 5.9. Here $\{1,2,3\}$ and $\{5,6,7\}$ represent a simple cycle, where each one is sink of the vertex 4.

The following corollary is a direct result of the proof of the previous proposition, which analyzes the case where the interaction graph consists only of simple cycles as non-trivial strongly connected components.

Corollary 5.5. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network. If $G(f)$ only has vertex-disjointed cycles, then the only solution for the L-DEN problem is the trivial one.*

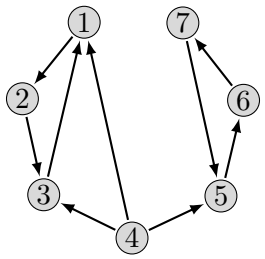


Figure 5.9: Sink simple cycles of one vertex

Proof. Let $s = B_1B_2$ be an update schedule with two blocks and a linear Boolean network h such that $f = h^s$.

Since all the cycles are sink from a empty set of vertices, we can apply Proposition 5.9, then for all cycles C in $G(f)$, $V(C) \subseteq B_1$ or $V(C) \subseteq B_2$.

Notice that including an arc in h between two cycles results is the same scenario as the first case of the previous proposition, and the proof is analogous.

Therefore, the only solution such that is dynamically equivalent to f is the trivial one. □

5.3 Algorithm for L-DEN problem

Given that the problem where the interaction graph of a linear Boolean network is acyclic and strongly connected has already been studied, and having established the interaction graphs for which the only solution is the trivial one, it is possible to state the following theorem which characterizes the solution for the L-DEN problem:

Theorem 5.1. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a linear Boolean network, and let $P = \{u \in V(f) : u$ is not a vertex of a cycle in $G(f)\}$. There exists a solution for L-DEN problem if and only if one of the following conditions are satisfied:*

1. *There exists a simple cycle C such that $|N_f^+(C)| > 0$*
2. *$|P| > 1$*
3. *$G(f)$ is disconnected and $|P| = 1$*

Besides, if a solution exists, it can be found in polynomial time.

Proof. Considering all the above results of this section, we can establish Algorithm 5.2

Let $G(f)$ be the interaction digraph of a linear Boolean network $f : \{0,1\}^n \to \{0,1\}^n$. Then consider the following cases:

1. The calculation of P is done in polynomial time, as selecting vertices that are not in a cycle is equivalent to them not being in a non-trivial strongly connected component.

Algorithm 5.2: L-DENPSolve

Input: The interaction graph $f : \{0,1\}^n \to \{0,1\}^n$ a linear Boolean network f such that $G(f)$ is not trivial

Output: If there exists a solution, a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f. Otherwise, Null.

Let $h : \{0,1\}^n \to \{0,1\}^n$ be such that $h(x) = (0, \ldots, 0)$
$P \leftarrow \{u \in V(f) : u \text{ is not a vertex of a cycle in } G(f)\}$
if *there exists a simple cycle* C *such that* $|N_f^+(C)| > 0$ **then**
  for $w \notin N_f^+(C)$ **do**
    $h(w) \leftarrow f(w)$
  end for
  for $w \in N_f^+(C)$ **do**
    let $\{v_{i_1}, \ldots, v_{i_m}\} = V(C) \cap N_f^-(w)$;
    $N_h^-(w) \leftarrow \left(N_f^-(w) \,\Delta \left(\sum_{j=1}^{m} N_f^-(v_{i_j+1}) \right) \right) \cup \{v_{i_j+1}\}_{j=1}^{m}$
  end for
  $s \leftarrow V(C), V(f) \setminus V(C)$;
  return (h, s)
else
  if $|P| > 1$ **then**
    $(h', s') \leftarrow \text{AcyclicSolve}(G(f)[P])$;
    $A(h) \leftarrow \{(w, z) \in A(f) : w \notin P\} \cup A(h')$;
    $s(w) \leftarrow \begin{cases} s'(w), & \text{if } w \in P \\ 1, & \text{if } w \notin P \end{cases}$;
    return (h, s)
  else
    if $G(f)$ *is disconnected and* $|P| = 1$ **then**
      Let $u \in V(f)$ such that $d^-(u) = 0$;
      Let $\tilde{G} \subseteq G$ be a connected component such that $u \notin V(\tilde{G})$ and let $\tilde{v} \in V(\tilde{G})$;
      $A(h) \leftarrow A(f) \cup \{(u, \tilde{v})\}$;
      $s \leftarrow V(f) \setminus V(\tilde{G}), V(\tilde{G})$
    else return *Null*;
  end if
end if

2. If there exists a simple cycle C such that $|N_f^+(C)| > 0$, then by Proposition 5.4, there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f. Moreover, searching for a simple cycle with this condition is equivalent to searching for a simple cycle in a strongly connected component that is not a simple cycle itself, which is done in polynomial time.

3. If the first condition is not met, then the only possible strongly connected components are simple sink cycles (which only receive arcs). Then, it is possible to search for two vertices that do not belong to a cycle with the same input neighborhood, which by Corollary 5.1 implies the existence of a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f. On the other hand, if all vertices that are not part of a cycle have different neighborhoods, then it is possible to apply Corollary 5.3, which implies that there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f. Besides, note that the input neighborhoods of each vertex in the new updated network h under s remain the same, because the vertices that are in a cycle were incorporated in the first block of the scheme s (line 17), and there is no outgoing arc from any of these vertices to another vertex in the acyclic component; otherwise, the first condition would have been met.

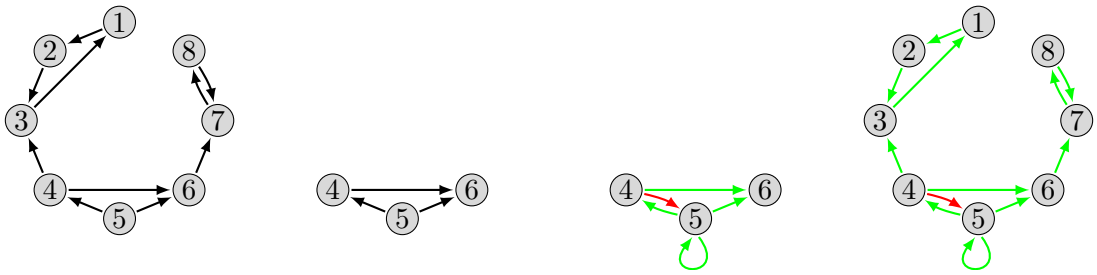
4. If there is only one vertex v that is not part of a cycle, then this vertex can be connected to a cycle or it can be a connected component by itself. Then, for a solution to exist, the graph must be disconnected, where each strongly connected component can only be a cycle or the vertex v. Given that v does not receive any arcs (otherwise the first proposition would have been satisfied), it is possible to apply Corollary 5.2, and therefore, there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f.

5. Given that none of the conditions were met, there are only 3 cases where there does not exist a non-trivially solution dynamically equivalent to f: $G(f)$ is a simple cycle (Corollary 5.4), $G(f)$ has only simple cycles as strongly connected components (Corollary 5.5), or $G(f)$ is a single-vertex sink cycles (Proposition 5.10).

□

Example 5.9. Given a linear Boolean network (Figure 5.10a), the first step is to identify the vertices that do not belong to a cycle, which is $P = \{4, 5, 6\}$. Then, given that all strongly connected components are vertices, and simple cycles without outgoing arcs, the solution is calculated for the acyclic component $G(f)[P]$ (Figure 5.10b), in which the in-neighborhoods of vertices 4, 5, and 6 are all different. Therefore, we proceed according to lines 5-8 of the AcyclicSolve algorithm, which outputs (h', s') is reflected in the labeled digraph of Figure 5.10c, where $s' = \{4, 6\}\{5\}$.

Next, continuing with the execution of the L-DENPSolve algorithm, the missing arcs are incorporated into the graph $G(h)$, and the new update scheme s is defined based on s', where $s = \{1, 2, 3, 4, 6, 7, 8\}\{5\}$. Thus, it exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f, which is presented in Figure 5.10d.



(a) $G(f)$ **(b)** $G(f)[P]$ **(c)** $G(h', s')$ **(d)** $G(h, s)$

Figure 5.10: Figures of Example 5.9

Theorem 5.2. *L-DEN can be decided in polynomial time.*

Proof. There are only three configurations of the interaction graph $G(f)$ where the only solution is the trivial one:

- If $G(f)$ is a simple cycle, then by Corollary 5.4, the only solution is the trivial one.
- If $G(f)$ only has simple cycles as strongly connected components, then by Corollary 5.5, the only solution is the trivial one.
- If $G(f)$ is a single-vertex sink cycles, then by Proposition 5.10, the only solution is the trivial one.

Based on these cases, we can construct an algorithm to determine if the interaction graph $G(f)$ differs from the above cases and thus, if there exists a linear Boolean network h and an update schedule s such that (h, s) is non-trivially dynamically equivalent to f:

Algorithm 5.3: L-DENDecider

Input: The interaction graph $G(f)$ of a linear Boolean network f where $G(f)$ is non-trivial

Output: **True** if a non-trivial solution to the L-DEN problem with instance $G(f)$ exists, or **False** otherwise

```
1  if G(f) is a simple cycle then
2  |  return False
3  else
4  |  if G(f) only has simple cycles as strongly connected components then
5  |  |  return False
6  |  else
7  |  |  if G(f) is a single-vertex sink cycles then return False;
8  |  |  else return True;
9  |  end if
10 end if
```

This algorithm can be executed in polynomial time, since it only requires calculating the strongly connected components and verifying if they form a simple cycle or have the structure of a single-vertex sink cycles.

□

Chapter 6

L-DEN fixing f and h

In this chapter, we study the problem of finding an update schedule s given two linear Boolean networks h and f such that (h, s) and f are dynamically equivalent. Accordingly, we define the following decision problem.

Dynamically equivalent linear Boolean networks with fixed Boolean network problem (L-DENh)
Input: h and f be two linear Boolean networks.
Question: Does there exist an update schedule such that $h^s = f$?

To study this problem, we consider the definitions presented in Chapter A, which extend the notion of a labeled digraph to partial labeling; that is, given $f : \{0,1\}^n \to \{0,1\}^n$, we define lab : $A(f) \to \{\oplus, \ominus, \bigcirc\}$. In addition, unlabeled arcs will be represented in brown in the figures; for example, given $(u, v) \in A(f)$, $\text{lab}(u, v) = \bigcirc$.

Let us note that a system of linear equations can be constructed in order to find a labeled digraph. Let $f, h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean networks, and let s be an update schedule such that $h^s = f$. For any $(u, v) \in A(h)$, we define the variable $a_{uv} \in \mathbb{F}_2$ by,

$$a_{uv} = \begin{cases} 1 & \text{if } \text{lab}(u,v) = \oplus \\ 0 & \text{if } \text{lab}(u,v) = \ominus \end{cases} \tag{6.1}$$

Thus, for each $v \in V(h)$ and each $x \in \{0,1\}^n$, the function h^s_v can be written as,

$$\begin{aligned} h^s(x)_v &= h(a_{1v}x_1 + (a_{1v}+1)f(x)_1, \ldots, a_{nv}x_n + (a_{nv}+1)f(x)_n) \\ &= \sum_{u \in N_h^-(v)} a_{uv}x_u + (a_{uv}+1)f(x)_u \end{aligned}$$

Since we want $h^s_v = f_v$, the corresponding system of equations $\mathcal{S}$ is given by,

$$\sum_{u \in N_h^-(v)} a_{uv}x_u + (a_{uv}+1)f(x)_u = f(x)_v \tag{6.2}$$

By using the definitions from Chapter 4, the left-hand side of the previous equation can be equivalently written as,

$$\begin{aligned} h_v^s(x) &= \sum_{u \in V} M(h)_{vu}[a_{uv}x_u + (a_{uv}+1)f_u(x)] \\ &= \sum_{u \in V} M(h)_{vu}a_{uv}x_u + \sum_{u \in V} M(h)_{vu}(a_{uv}+1)f_u(x) \\ &= \sum_{u \in V} M(h)_{vu}a_{uv}x_u + \sum_{u \in V} M(h)_{vu}(a_{uv}+1)\sum_{w \in V} M(f)_{uw}x_w \end{aligned}$$

Then, since $M(f) = M(h^s)$, we obtain:

$$M(f)_{vz} = M(h)_{vz}a_{zv} + \sum_{u \in V} M(f)_{uz}M(h)_{vu}(a_{uv}+1)$$

Equivalently,

$$M(h)_{vz}a_{zv} + \sum_{u \in V} M(f)_{uz}M(h)_{vu}a_{uv} = M(f)_{zv} + \sum_{u \in V} M(f)_{uz}M(h)_{vu} \tag{6.3}$$

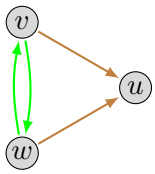
If, in addition, we impose the constraint that $\forall (u,u) \in A(h)$, $a_{uu} = 1$, we obtain a system of linear equations with variables a_{uv}.

From the way in which this system of equations is constructed, it follows that if the L-DENh problem has a solution, then the system of equations must also have a solution, given that the solutions of L-DENh are included in the equation system.

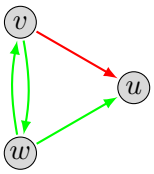
If none of the labelings in this solution space induces an update digraph, then there does not exist an update schedule s such that $h^s = f$; that is, the L-DENh problem has no solution.

Now, observe that for each vertex the system of equations can be written in the form $Az = b$, where $A \in \mathcal{M}_{n\times n}(\mathbb{F}_2)$ and $b, z \in \mathcal{M}_{n\times 1}(\mathbb{F}_2)$. Consequently, the size of the solution space for each vertex is of order $\mathcal{O}(2^{n-\text{rank}(A|b)}) \subseteq \mathcal{O}(2^n)$. Therefore, the total solution space is of order $\mathcal{O}(2^m)$, where $m = |A(h)|$.

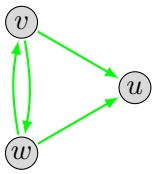
Remark 6.1. Note that additional equations can be added to the system in order to avoid forbidden cycles. That is, let be a digraph $G(h)$ and let $v, w \in V(h)$, where v and w are part of a strongly connected component with positive arcs (for instance, consider the Figure 6.1). Observe that if $\text{lab}(v,u) = \ominus \wedge \text{lab}(w,u) = \oplus$ (Figure 6.1b) or if $\text{lab}(v,u) = \oplus \wedge \text{lab}(w,u) = \ominus$, then the forbidden cycle u, v, w, u is formed. Therefore, it is necessary that the arcs (v,u) and (w,u) have the same label, which is enforced by adding the equation $a_{vu} + a_{wu} = 0$ to the system of equations (Figure 6.1c). From this, multiple equations can be incorporated into the system for every pair of vertices v and w that belong to a positive strongly connected component and such that both have arcs toward u. Besides, adding these equations to the system allows the solution to be found more quickly, since the additional constraints reduce the search space for a labeled digraphs that is update.



(a) $G(h)$, after solving the system of equations



(b) $G(h)$, with $\text{lab}(v,u) = \ominus$ and $\text{lab}(w,u) = \oplus$



(c) $G(h)$, after forcing system with $a_{vu} + a_{wu} = 0$

Figure 6.1: A condition based on forbidden cycle for the equation system.

Given the high computational cost of exploring the entire solution space, and considering that the free variables correspond to unlabeled arcs, it is proposed the algorithm `TotalForce`. This algorithm is based on `Force` (Algorithm A.1), introduced by Palma et al. (2015), which identifies arcs that must be labeled in such a way that the digraph remains an update digraph (i.e., there does not exist a forbidden cycle).

By applying `TotalForce`, free variables of the system of equations can be fixed, that is, variables that are not constrained by the system itself. Imposing the update digraph conditions derived from `TotalForce`, together with those presented in Remark 6.1, help the system of equations to be solved more efficiently and it helps to find a solution to the L-DENh problem.

Moreover, according to Theorem 3.1 (Montalva (2011)), if the partial labeling obtained after solving the system of equations is update, then there exists a complete labeling under which the digraph remains update. Therefore, if such a complete labeling also satisfies the system of equations, it is obtained a solution to the L-DENh problem.

Algorithm 6.1: `TotalForce`($\mathcal{S}$)

Input: A system of linear equations $\mathcal{S}$
Output: A system of linear equations $\mathcal{S}$

```
repeat
    S' ← S;
    Construct the labeled digraph G(h, lab) induced by S;
    Apply Force to G(h, lab);
    S ← Solve S using the newly labeled arcs returned by Force;
until S' = S;
return S
```

The Algorithm 6.1 initially operates within a `repeat` loop, where it assigns labels to arcs corresponding to free variables of the system of equations in such a way that no forbidden cycle is created, that is, the resulting labeled digraph remains an update digraph. After these labels are forced, the system of equations is solved again using the newly fixed labels.

Note that, according to what is stated in the work of Palma et al. (2015), from The-

orem 3.1 it is possible to deduce that a partially labeled digraph that does not contain a forbidden cycle can be completed in at least one way such that the resulting labeled digraph is an update digraph. Consequently, the use of `Force` allows the construction of an update schedule in such a way that, at each iteration of the loop, a digraph that remains update is constructed.

If the system of equations remains unchanged with respect to the iteration of the `repeat` loop, then no additional arcs can be forced. In this case, Algorithm 6.1 terminates and returns the updated system of equations.

To analyze the time complexity of `TotalForce`, let us define $m = |A(h)|$ and $n = |V(f)|$, and note that in Algorithm 6.1, the `repeat` loop is executed at most $\mathcal{O}(m)$ times, and that applying `Force` to the digraph $G(h, \text{lab})$ requires $\mathcal{O}(n^3)$ time[1]. Updating and solving the system $\mathcal{S}$ also requires $\mathcal{O}(n^3)$ time [2]. Therefore, the overall time complexity of Algorithm 6.1 is $\mathcal{O}(n^3 m)$.

In this way, if only it is considered an algorithm that solves the system of equations and subsequently applies `TotalForce`, three possible types of labelings may arise:

- A complete labeling;
- A partial labeling;
- A labeling that does not yield a solution (either because it is not update or because it does not satisfy the system of equations).

To illustrate this, the following examples presents each of the possible types of labelings obtained when considering only the system of equations together with `TotalForce`.

Example 6.1. Consider two linear Boolean networks $f, h : \{0,1\}^n \to \{0,1\}^n$ defined by

i	f	h
1	$x_3 + x_4 + x_5 + x_6$	$x_3 + x_4 + x_5 + x_6$
2	$x_2 + x_3 + x_4 + x_5$	$x_2 + x_3 + x_4 + x_5$
3	$x_2 + x_4 + x_5$	$x_1 + x_2 + x_3 + x_6$
4	$x_3 + x_4 + x_5 + x_6$	$x_3 + x_4 + x_5 + x_6$
5	x_3	x_3
6	$x_3 + x_5$	$x_3 + x_5$

From the above networks and considering the definition of the system of equations given by (6.3), we can deduce the following for each vertex.

[1] As shown in the `Force` algorithm (see Appendix and references), this procedure is based on the construction of the reverse-paths matrix M_{cr}, which is computed using the `ReversePaths` algorithm (see references and appendix) and runs in $\mathcal{O}(n^3)$ time. Consequently, the `Force` procedure has time complexity $\mathcal{O}(n^3)$.

[2] Solving a system of linear equations over $\mathbb{F}_2$ using Gaussian elimination takes $\mathcal{O}(n^2)$ time. Nevertheless, Wiedemann (1986) presents a random method of solving a nondegenerate linear system in n equations and n unknowns over a finite field in $\mathcal{O}(nw)$ field operations.

f_1:

	a_{31}	a_{41}	a_{51}	a_{61}	
1	0	0	0	0	$= 0$
2	1	0	0	0	$= 0+1$
3	1	1	1	1	$= 1+1+1+1$
4	1	$1+1$	0	0	$= 1+1+1$
5	1	1	1	1	$= 1+1+1+1$
6	0	1	0	1	$= 1+1$

	a_{31}	a_{41}	a_{51}	a_{61}	
2	1	0	0	0	$= 1$
3	1	1	1	1	$= 0$
4	1	0	0	0	$= 1$
5	1	1	1	1	$= 0$
6	0	1	0	1	$= 0$

Applying Gauss elimination:

	a_{31}	a_{41}	a_{51}	a_{61}	
2	1	0	0	0	$= 1$
3	0	1	1	1	$= 1$
4	0	0	0	0	$= 0$
5	0	1	1	1	$= 1$
6	0	1	0	1	$= 0$

	a_{31}	a_{41}	a_{51}	a_{61}	
2	1	0	0	0	$= 1$
3	0	1	1	1	$= 1$
4	0	0	0	0	$= 0$
5	0	0	0	0	$= 0$
6	0	0	1	0	$= 1$

It is obtained: $a_{31} = 1$,$a_{51} = 1$ and $a_{41} = a_{61}$.

f_2:

	a_{32}	a_{42}	a_{52}	
1	0	0	0	$= 0$
2	1	0	0	$= 1+1+1$
3	1	1	1	$= 1+1+1$
4	1	$1+1$	0	$= 1+1+1$
5	1	1	1	$= 1+1+1$
6	0	1	0	$= 0+1$

	a_{32}	a_{42}	a_{52}	
2	1	0	0	$= 1$
3	1	1	1	$= 1$
4	1	0	0	$= 1$
5	1	1	1	$= 1$
6	0	1	0	$= 1$

It is obtained: $a_{32} = a_{41} = a_{52} = 1$.

f_3:

	a_{13}	a_{23}	a_{63}	
1	1	0	0	$= 0$
2	$1+1$	0	0	$= 1+1$
3	1	1	1	$= 0+1+1+1+1$
4	1	1	0	$= 1+1+1$
5	1	1	1	$= 1+1+1+1$
6	1	0	1	$= 0+1$

	a_{13}	a_{23}	a_{63}	
1	1	0	0	$= 0$
2	0	0	0	$= 0$
3	1	1	1	$= 0$
4	1	1	0	$= 1$
5	1	1	1	$= 0$
6	1	0	1	$= 1$

It is obtained: $a_{13} = 0$, $a_{23} = 1$ and $a_{63} = 1$

f_4:

	a_{34}	a_{54}	a_{64}	
1	0	0	0	$= 0$
2	1	0	0	$= 0+1$
3	1	1	1	$= 1+1+1$
4	1	0	0	$= 1+1+1$
5	1	1	1	$= 1+1+1$
6	0	0	1	$= 1$

	a_{34}	a_{54}	a_{64}	
2	1	0	0	$= 1$
3	1	1	1	$= 1$
4	1	0	0	$= 1$
5	1	1	1	$= 1$
6	0	0	1	$= 1$

It is obtained: $a_{34} = a_{54} = a_{64} = 1$.

f_5: It follows immediately that: $a_{35} = 1$

f_6:

	a_{36}	a_{56}	
1	0	0	$= 0$
2	1	0	$= 0+1$
3	1	1	$= 1+1$
4	1	0	$= 0+1$
5	1	1	$= 1+1$
6	0	0	$= 0$

	a_{36}	a_{56}	
2	1	0	$= 1$
3	1	1	$= 0$
4	1	0	$= 1$
5	1	1	$= 0$

It is obtained: $a_{36} = a_{56} = 1$.

The result of this system of equations is presented in Figure 6.2

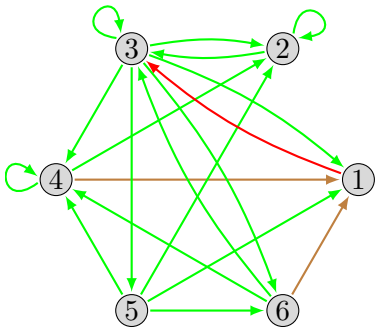


Figure 6.2: Example of solution of system of linear equations.

Next, Algorithm 6.1 is applied, which in turn, within the `repeat` loop, begins by applying Algorithm A.1. In this step, it is obtained that the arc $(6, 1)$ must be labeled positive, since if it were labeled negative, the forbidden cycle $6, 3, 1, 6$ would be formed; consequently, the arc $(4, 1)$ must also be labeled positive, by the solution of the system of equations. Thus, the unique solution is the digraph $G(h, \text{lab})$ presented in Figure 6.3.

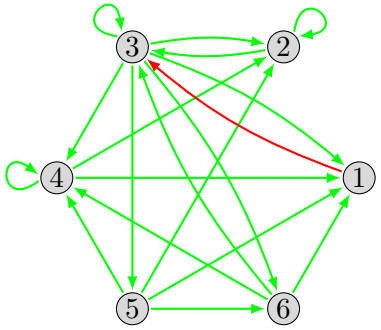


Figure 6.3: Example of Algorithm 6.2.

Note that the analysis of Algorithm A.1 also implies that the arc $(4,1)$ must be labeled positive, since if it were labeled negative, the forbidden cycle $4,2,3,5,1,4$ would be formed.

Example 6.2. For this example consider two linear Boolean networks $f, h : \{0,1\}^4 \to \{0,1\}^4$ defined by (see Figure 6.7a):

i	f	h
1	x_1	$x_2 + x_3 + x_4$
2	$x_1 + x_3 + x_4$	$x_1 + x_3 + x_4$
3	$x_1 + x_2 + x_4$	$x_1 + x_2 + x_4$
4	$x_1 + x_2 + x_3$	$x_1 + x_2 + x_3$

Then the system of equations using (6.3) is as follows.

f_1:

	a_{21}	a_{31}	a_{41}	
1	1	1	1	$= 0$
2	1	1	1	$= 0$
3	1	1	1	$= 0$
4	1	1	1	$= 0$

	a_{21}	a_{31}	a_{41}	
1	1	1	1	$= 0$
2	0	0	0	$= 0$
3	0	0	0	$= 0$
4	0	0	0	$= 0$

It is obtained $a_{21} + a_{31} + a_{41} = 0$.

f_2:

	a_{12}	a_{32}	a_{42}	
1	0	1	1	$= 0$
2	0	1	1	$= 0$
3	0	1	1	$= 0$
4	0	1	1	$= 0$

	a_{12}	a_{32}	a_{42}	
1	0	1	1	$= 0$
2	0	0	0	$= 0$
3	0	0	0	$= 0$
4	0	0	0	$= 0$

It is obtained: $a_{32} = a_{42}$ and a_{12} is free.

f_3:

	a_{13}	a_{23}	a_{43}	
1	0	1	1	$= 0$
2	0	1	1	$= 0$
3	0	1	1	$= 0$
4	0	1	1	$= 0$

	a_{13}	a_{23}	a_{43}	
1	0	1	1	$= 0$
2	0	0	0	$= 0$
3	0	0	0	$= 0$
4	0	0	0	$= 0$

It is obtained: $a_{23} = a_{43}$ and a_{13} is free.

f_4:

	a_{14}	a_{24}	a_{34}	
1	0	1	1	$= 0$
2	0	1	1	$= 0$
3	0	1	1	$= 0$
4	0	1	1	$= 0$

	a_{14}	a_{24}	a_{34}	
1	0	1	1	$= 0$
2	0	0	0	$= 0$
3	0	0	0	$= 0$
4	0	0	0	$= 0$

It is obtained: $a_{24} = a_{34}$ and a_{14} is free.

Note that the only variable fixed in the system of equations is the one corresponding to the loop at vertex 1, which has a positive label, while the remaining arcs retain free labels; the labeling resulting from this system of equations $\mathcal{S}$ is represented by the digraph in Figure 6.4.

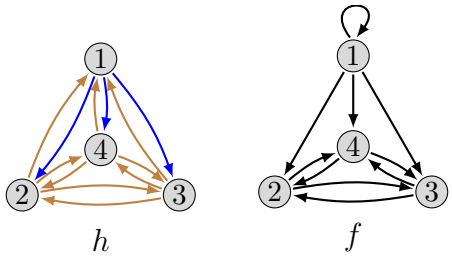


Figure 6.4: Labeling of h and f

Next, the algorithm `TotalForce` (Algorithm 6.1) is applied, which it does not force any arc label in the digraph, nor does it fix any additional label through the system of equations; thus, the digraph remains update.

In this example, the system of equations does not impose any restriction on the variables a_{12}, a_{13}, and a_{14} (whose corresponding arcs are labeled in blue in Figure 6.4). Later, in Example 6.4, it will be shown that the only labeling that yields a solution requires that the three arcs directed toward vertex 1 be negative, which implies that $\text{lab}(1,2) = \text{lab}(1,3) = \text{lab}(1,4) = \oplus$; otherwise, a forbidden cycle is formed.

Example 6.3. Now, we show an example when the system of equations does prove that there does not exist a solution.

Let be two linear Boolean networks $f, h : \{0,1\}^n \to \{0,1\}^n$ (see Figure 6.5):

i	f	h
1	x_3	x_2
2	x_1	x_1
3	0	$x_1 + x_2 + x_3$

(a) $G(f)$ **(b)** $G(h)$

Figure 6.5: Example of an inconsistent equation system.

From the equation corresponding to x_2, we obtain $a_{23} = 0$, which implies from the equation of x_1 that $a_{13} + 0 + 1 = 0$, and therefore $a_{13} = 1$. Substituting this value into the equation of x_3, we obtain $1 + 1 + 1 = 0$, which renders the equation system as inconsistent. This cam also be seen from the system of equations planted in (6.3):

	a_{13}	a_{23}	a_{33}		
1	1	1	0	=	1
2	0	1	0	=	0
3	1	0	0	=	0

	a_{13}	a_{23}	a_{33}		
1	1	0	0	=	0
2	0	1	0	=	0
3	0	0	0	=	1

Since exploring the solution space of the system of equations is of exponential order and the partial labeling obtained from the system may remain partial even after applying `TotalForce`, two strategies are proposed in order to obtain a complete labeling that solves the L-DENh problem:

Solve-and-Force. This approach combines the system of equations with `TotalForce`, but additionally assigns labels according to specific criteria, after which `TotalForce` is applied iteratively. The detailed description of this algorithm is provided in Section 6.1.

Block-Iteration. This algorithm follows the spirit of the method proposed by Cabrera (2024) for solving the problem in the disjunctive case, which is based on the use of the definition of a k-valid schedule (Definition 6.1). At each iteration k, the algorithm seeks to construct the largest possible $(k+1)$-block schedule. From this construction, criteria are introduced to drop vertices from block $k+1$ to block $k+2$, one of which incorporates the use of `TotalForce`. A detailed description of this algorithm is presented in Section 6.2.

It is worth noting that the main difference between these two algorithms lies in their respective constructions: the first algorithm builds labels on the digraph, whereas the second constructs the blocks of the update schedule directly.

6.1 Solve-and-Force algorithm

In the present section, the first strategy is defined. This strategy operates recursively and complements the system of equations and the algorithm `TotalForce` by assigning a label to an arc.

Algorithm 6.2: $\mathtt{s}(f, h)$

Input: Two linear Boolean networks f and h
Output: `True` if no free labels remain and the labeled digraph is update; `False` otherwise

1 **if** $f = h$ **then return** `True` ;
2 $\mathcal{S} \leftarrow$ solution of the system of linear equations induced by f and h
3 **if** $\mathcal{S}$ *does not have a solution* **then return** `False` ;
4 **if** $\forall v \in V, N_h^-(v) \neq N_f^-(v)$ **then return** `False` ;
5 **return** $\mathtt{solve}(f, h, \mathcal{S})$

Algorithm 6.2 takes as input two linear Boolean networks f and h. If $f = h$ (line 1), then h can be updated using the parallel update schedule s_p, yielding $h^{s_p} = f$. Hence, a solution exists and the algorithm returns `True`.

Then, in line 2, the system of equations is constructed and solved according to eq. (6.2).

If the associated system of equations has no solution (line 3), that is, if it is inconsistent, the algorithm returns `False`. Moreover, if there does not exist a vertex $v \in V(h)$ such that $N_h^-(v) = N_f^-(v)$ (line 4), then it does no exists a vertex belonging to the first block of the update schedule. Consequently, there does not exist an update schedule and the algorithm again returns `False`.

If none of the above conditions holds, the algorithm proceeds by executing Algorithm 6.3.

In order to identify which arcs can be fixed to avoid the existence of a forbidden cycle, Algorithm 6.3 begins by applying the procedure `TotalForce` (Algorithm 6.1) to the system of equations (line 1).

Then, if all labels are forced, that is, if no free labels remain, `Verify` (Algorithm A.3) is applied in order to see if the labeled digraph induced by the system of equations is update, this algorithm was presented by Palma et al. (2015). Then, if the resulting labeled digraph is an update digraph, a solution exists and the algorithm returns `True`; otherwise, it returns `False`.

On the other hand, if at least one free label remains, the digraph that has been forced and whose remaining labels have been fixed by the system of equations is checked to determine whether it is an update digraph. If it is not update, then there does not exist a solution, and `False` is returned; otherwise, an arc $e = (u, v) \in A(h)$ is selected and labeled positive. After fixing $a_{uv} \leftarrow 1$ ($\mathrm{lab}(e) \leftarrow \oplus$), Algorithm 6.3 is applied recursively to the resulting labeling. If there does not exist a solution under the choice $a_{uv} \leftarrow 1$, then the variable (label) is changed to $a_{uv} \leftarrow 0$ ($\mathrm{lab}(e) \leftarrow \ominus$), and Algorithm 6.3 is applied again.

Algorithm 6.3: `solve`$(f, h, \mathcal{S})$

Input: Two linear Boolean networks f and h, and system of linear equations $\mathcal{S}$
Output: `True` if no free labels remain and the labeled digraph is update; `False` otherwise

```
S ← TotalForce(S)
if No free labels remain then
    return Verify(S)
else
    if Verify(S) then
        Let an arc e ← (u, v) ∈ A(h) unlabeled
        S' ← Solve S considering a_uv ← 1
        if solve(f, h, S') then
            return True
        else
            S' ← Solve S considering a_uv ← 0
            return solve(f, h, S')
        end if
    else
        return False
    end if
end if
```

Remark 6.2. It is important to emphasize that, although this algorithm only returns `True` when a solution exists and `False` otherwise, it is in fact capable of constructing the solution. That is, it produces a labeled digraph that is update, from which the update schedule can be constructed by applying Algorithm B.1 (Aracena et al. (2011)).

Note that the algorithm `solve` selects an arc at random to be labeled as positive. This choice can be refined according to the following alternatives:

1. Choosing the label of the arc;
2. Choosing the arc to be labeled.

Among these options, the one that has the greatest impact on the solution of the L-DENh problem is the choice of the label. Indeed, if more arcs are assigned negative labels, the induced update schedule tends to contain a larger number of blocks, and the probability of forming a forbidden cycle increases. Conversely, if positive labels are preferred, the blocks tend to incorporate a larger number of vertices. By grouping more vertices within the same block, the approach follows the idea proposed by Cabrera (2024), where the first blocks are constructed to be as large as possible. In Section 6.2, an algorithm is proposed that follows this same principle.

On the other hand, regarding the second point, a variant is proposed for selecting the arc to be labeled as positive. This variant is based on the idea introduced by Cabrera (2024) of incorporating as many vertices as possible into the first blocks using the definition of a k-valid schedule (Definition 6.1, which will be introduced later). The idea of incorporating these vertices into the first blocks can be reflected by assigning positive labels to arcs that are not yet labeled and whose induced update of the network h produces the same dependencies as the original network f. In this way, selecting arcs that satisfy this condition increases the probability that the dependencies of the updated network coincide with those of the original network. Based on this idea, the following algorithm is proposed.

Algorithm 6.4: `BuildAndAssignPossitive`$(f, h, \mathcal{S})$

Input: Two linear Boolean networks f and h, and a system of linear equations $\mathcal{S}$

Output: A labeled arc

Label with $\oplus$ all unlabeled arcs of $G(h)$, denoted by $G(h, \text{lab}^{\oplus})$.
if $G(h, \text{lab}^{\oplus})$ *is an update digraph* **then**
  Let s' be induced by labels
  $T \leftarrow \{(z,u) \in A(h) : N^-_{h^{s'}}(u) = N^-_f(u) \wedge \text{lab}(z,u) = \bigcirc\}$
  if $T \neq \emptyset$ **then return** $e \in T$ *with label* $\oplus$;
  else return *Any partially labeled arc with label* $\oplus$;
else
  return *Any unlabeled arc with label* $\oplus$
end if

The main purpose of Algorithm 6.4 is to return an arc labeled as positive. To select such arc, the algorithm first temporarily assigns a positive label to all arcs of the digraph that are initially unlabeled, resulting in a labeling that may or may not induce an update digraph. If the resulting labeled digraph is an update digraph, then an update schedule s' can be constructed [3].

Next, the set T is defined as the set of arcs (z, u) such that the vertex u has equals dependencies in $h^{s'}$ and f, and whose labels were initially free; that is, there exists a vertex $z \in N^-_h(u)$ such that $\text{lab}(z, u) = \bigcirc$. If $T \neq \emptyset$, an arc from T can be selected and assigned a positive label.

If $T = \emptyset$, then an arbitrary arc with a free label is selected and assigned a positive label.

[3] An algorithm that constructs an update schedule from a given labeled digraph is presented by Aracena et al. (2011) (Algorithm B.1)

6.1.1 Complexity, confidence and examples

In this section, we analyze the time complexity of the Algorithm 6.2, discuss simulation results that guarantee its stability, and present illustrative examples.

Algoritm complexity and ratio confidence

Let L denote the number of free labels at the beginning of Algorithm 6.3. Since at each iteration of Algorithm 6.3 the algorithm branches by assigning either a positive or a negative label, the total number of possible label combinations is $2^L \in \mathcal{O}(2^m)$. Moreover, as the algorithm starts by applying Algorithm 6.1, which requires $\mathcal{O}(n^3m)$ time, the overall worst-case time complexity of the algorithm is $\mathcal{O}(2^m n^3 m)$.

Observe that if a solution is found by consistently assigning the label $\oplus$, then the best-case running time of Algorithm 6.3 is $\mathcal{O}(n^3m^2)$. As will be shown later, in a large proportion of the simulations a solution is indeed found by always choosing the label $\oplus$.

Finally, the time complexity of Algorithm 6.2 is $\mathcal{O}(2^m n^3 m)$, since:

- Checking whether two Boolean networks are equal takes $\mathcal{O}(n^2)$ time
- Constructing and solving the system $\mathcal{S}$ takes $\mathcal{O}(n^3)$ time
- Verifying that there exists no vertex with identical dependencies in h and f requires $\mathcal{O}(n^2)$ time
- Algorithm 6.3 runs in $\mathcal{O}(2^m n^3 m)$ time

To analyze the time required by Algorithm 6.2 to find a solution, simulations were performed on linear Boolean networks with sizes ranging from 3 to 40 vertices, considering 1000 randomly generated networks for each size. We define the *First branch* as the execution path in which the algorithm initially assigns a positive label at each iteration and reverts to a negative assignment only when the positive choice does not yield a solution. Otherwise, we define the *No-first branch*.

Across the iterations for each network size, it was obtained that the algorithm finds a solution in the first branch in 100% of the cases. However, atypical cases were identified under which the algorithm does not find a solution in the first branch. These cases are presented in Examples 6.4 to 6.6. For the cases in which a solution is found in the first branch, Figure 6.6 presents the average execution time for each network size.

From Figure 6.6, it is observed stability in the average time required to find a solution in the first branch with respect to the network size, with these times remaining below one second. Moreover, under this number of iterations, the algorithm finds a solution in the first branch in 100% of the cases.

It should also be noted that simulations were performed using the variant of Algorithm 6.3 that considers Algorithm 6.4 instead of select an arc at random. This resulted in a marginal increase in the time required to search for a solution, while still

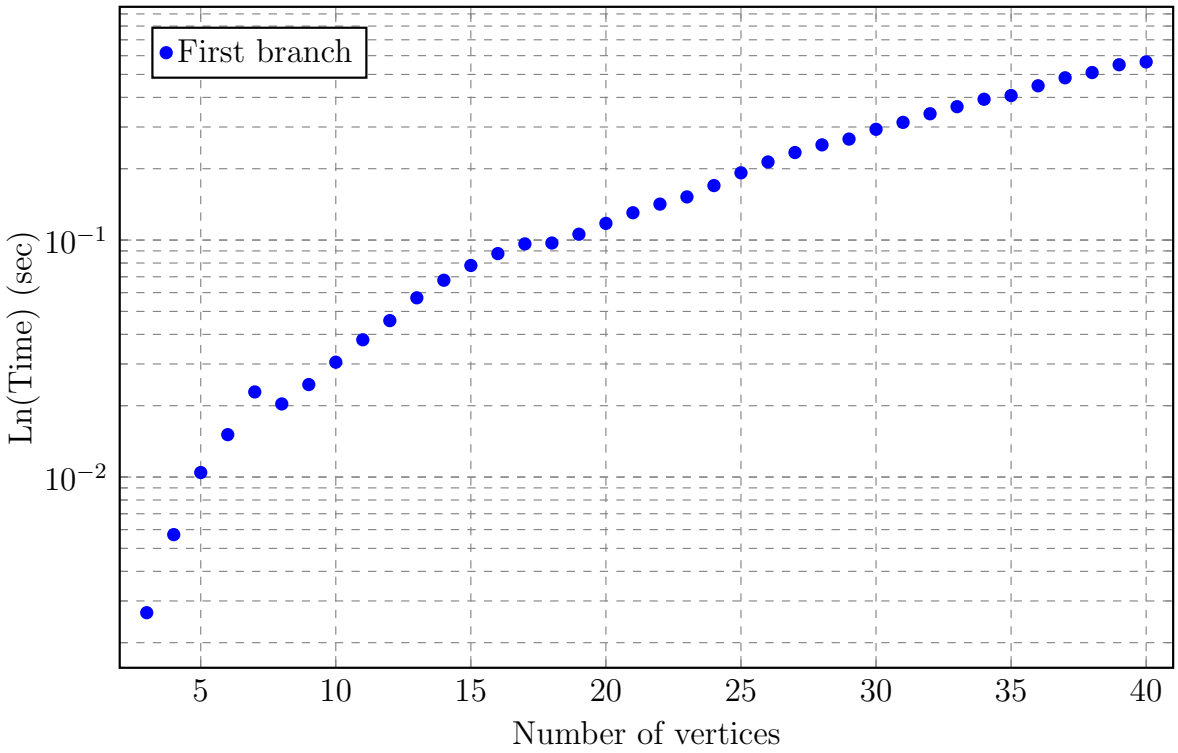


Figure 6.6: Execution time of Solve-and-Force.

achieving the same proportion of cases in which a solution is found in the first branch (100%). This behavior is due to the fact that, when solving the system of equations, the algorithm determines a large portion of what would otherwise be defined by T (line 4).

Note that, from selecting an randomly arc to be positive labeled, the following variants can be defined:

- Choosing the label of the arc e to be always negative (probability of being positive equal to zero);
- Choosing the label to be positive or negative with probability 0.5.

Considering the above variants and the number of simulations, it gets no difference on the total number of cases in which the variants find a solution in the first branch.

Examples

In this section, examples are presented considering the algorithm Algorithm 6.2 in its original version and its variants.

The following example illustrates the execution of Algorithm 6.2 when a solution is not found in the first branch, and it will be shown that, had the variant using Algorithm 6.4 been employed, a solution would have been found in the first branch.

Example 6.4. Consider two linear Boolean networks defined in Example 6.2. Now, following the execution of Algorithm 6.2, on both Boolean networks, it is observed that $f \neq h$ and that, for $v \in \{2,3,4\}$, $N_f^-(v) = N_h^-(v)$; consequently, the system of equations is constructed as Example 6.2. Then, `solve` is executed, starting with `TotalForce` which as Example 6.2 has shown, the algorithm does not force any arc label in the digraph, nor it fix any additional label through the system of equations. Given this, the digraph remains update, since only the loop at vertex 1 is labeled. In this way, the assignment of labels to the arcs then proceeds.

- $\text{lab}(2,1) = \oplus$. This assignment is represented by the digraph in Figure 6.7c. When solving the system of equations based on this label (line 7), applying `Force` (line 4), and solving the system of equations again (line 5), no additional label is fixed.

- $\text{lab}(3,1) = \oplus$. By fixing $\text{lab}(3,1) = \oplus$ and solving the system of equations, the label $\text{lab}(4,1) = \ominus$ is fixed, which is reflected in the digraph in Figure 6.7d. Then, by applying `Force` (Algorithm A.1 at line 4), the following labels are fixed: $\text{lab}(4,2) = \text{lab}(4,3) = \ominus$ and $\text{lab}(1,4) = \text{lab}(2,4) = \text{lab}(3,4) = \oplus$ (Figure 6.7e); if any of these arcs had a different label, a forbidden cycle would be formed. Thus, the system of equations for vertex 2 based on these assignments yields $a_{42} = 0$ and $a_{32} + a_{42} = 0$, which has solution $a_{32} = 0$. Analogously, we obtain $a_{23} = 0$, resulting in the labeling $\text{lab}(a_{32}) = \text{lab}(a_{23}) = \ominus$ (Figure 6.7f). Since $2, 3, 2$ is a forbidden cycle, the partially labeled digraph is not update.

- $\text{lab}(3,1) = \ominus$. Note that by assigning $\text{lab}(3,1) = \ominus$, the system yields $\text{lab}(4,1) = \oplus$. Then, `Force` yields $\text{lab}(3,2) = \text{lab}(3,4) = \ominus$ and $\text{lab}(1,3) = \text{lab}(2,3) = \text{lab}(4,3) = \oplus$ (note that, for instance, if $\text{lab}(4,3) = \ominus$, the forbidden cycle $1, 3, 4, 1$ would be formed). Next, solving the system of equations based on these new labels results in $\text{lab}(4,2) = \text{lab}(2,4) = \ominus$, which clearly does not correspond to an update digraph, since $2, 4, 2$ is a forbidden cycle. This sequence of operations is reflected in the digraph shown in Figure 6.7g.

- $\text{lab}(2,1) = \ominus$. With this assignment, the algorithm proceeds to a new branch, in which only the label $\text{lab}(1,2) = \oplus$ is forced, resulting in the system fixing no additional variables. The outcome of this step is shown in Figure 6.7h.

- $\text{lab}(3,1) = \oplus$. This assignment is shown in Figure 6.7i, here the system fixes the label of arc $(4,1)$, that is, $\text{lab}(4,1) = \oplus$ (line 7). Then, `Force` is applied (line 4), yielding the following labels: $\text{lab}(3,2) = \text{lab}(4,2) = \oplus$ and $\text{lab}(2,3) = \text{lab}(2,4) = \ominus$. Solving the system of equations based on these new variables yields $\text{lab}(4,3) = \text{lab}(3,4) = \ominus$. From this, the forbidden cycle $3, 4, 3$ is obtained, and thus the digraph is not update.

- $\text{lab}(3,1) = \ominus$. This assignment is reflected in Figure 6.7j. The system of equations fixes the label $\text{lab}(4,1) = \ominus$. Then, by applying `Force`, we obtain

$\text{lab}(1,3) = \text{lab}(1,4) = \oplus$; subsequently, the system of equations (line 5) in `TotalForce` (Algorithm 6.1) does not fix any additional labels.

- $\text{lab}(3,2) = \oplus$. Under this assignment, the system of equations fixes the label $\text{lab}(4,2) = \oplus$. With this, no additional labels are fixed either by `Force` or by the system of equations. Thus, $\text{lab}(2,3) = \oplus$ is assigned, which by the system of equations fixes $\text{lab}(4,3) = \oplus$. Then, neither `Forxe` nor the system fixes any further labels. The sequence of these steps is illustrated in Figure 6.7k.

- $\text{lab}(2,4) = \oplus$. Under this assignment, the system fixes the label $\text{lab}(3,4) = \oplus$. As a result, the digraph obtained at this step labels the last free arc, yielding a fully labeled digraph, which is update. The result of this final step is shown in Figure 6.7l.

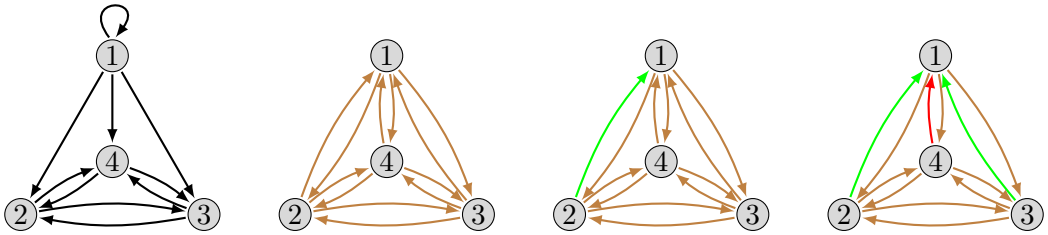
From this example, it becomes evident that the arcs corresponding to the variables a_{12}, a_{13}, and a_{14}, which are free in the system of equations, cannot receive a negative label. Indeed, any arc directed toward vertex 1 in the network h must be labeled negative; otherwise, either a forbidden cycle is created or there does not exist a solution.

Remark 6.3. Note that, in Example 6.4, if any arc not directed toward vertex 1 had been chosen as positive, the algorithm would have found a solution in the first branch. This can be achieved by using Algorithm 6.4. For instance, if all remaining arcs are provisionally labeled as positive, it follows that for $u \in \{2,3,4\}$, $N^-_{h^{sp}}(u) = N^-_f(u)$. Consequently, an in-arc of u can be selected and assigned a positive label.

Moreover, this example can be generalized to n vertices: let $f, h : \{0,1\}^n \to \{0,1\}^n$ be defined as in Figure 6.8. Note that the resulting system of equations takes the following form:

- For $u = 1$,

$$\begin{aligned} h^s(x)_1 &= \sum_{v=2}^{n} a_{v1}x_v + (a_{v1}+1)f(x)_v \\ &= \sum_{v=2}^{n} a_{v1}x_v + (a_{v1}+1)\left(\sum_{w\neq v} x_w\right) \\ &= f(x)_1 \\ &= x_1 \end{aligned}$$



(a) $G(f)$ **(b)** $G(h)$ labeled under $\mathcal{S}$ solution **(c)** $G(h)$ with $\mathrm{lab}(2,1) = \oplus$ **(d)** $G(h)$ with $\mathrm{lab}(2,1) = \oplus$, $\mathrm{lab}(3,1) = \oplus$ after `Force`

(e) $G(h)$ with $\mathrm{lab}(2,1) = \oplus$, $\mathrm{lab}(3,1) = \oplus$ after `Force` **(f)** $G(h)$ with $\mathrm{lab}(2,1) = \oplus$, $\mathrm{lab}(3,1) = \oplus$ after `Force` and solve $\mathcal{S}$ **(g)** $G(h)$ with $\mathrm{lab}(2,1) = \oplus$, $\mathrm{lab}(3,1) = \ominus$ after `Force` and solve $\mathcal{S}$ **(h)** $G(h)$ with $\mathrm{lab}(2,1) = \ominus$, $\mathrm{lab}(1,2) = \oplus$ and `Force`

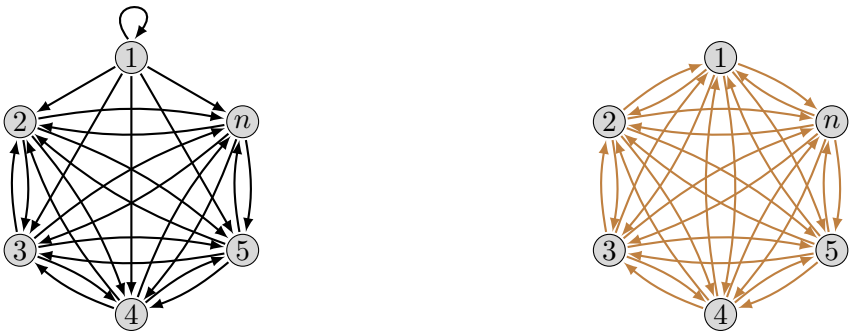
(i) $G(h)$ with $\mathrm{lab}(2,1) = \ominus$, $\mathrm{lab}(1,2) = \oplus$, $\mathrm{lab}(3,1) = \oplus$ after `Force` and solve $\mathcal{S}$ **(j)** $G(h)$ with $\mathrm{lab}(2,1) = \ominus$, $\mathrm{lab}(1,2) = \oplus$, $\mathrm{lab}(3,1) = \ominus$ after solve $\mathcal{S}$ and `Force` **(k)** $G(h)$ with $\mathrm{lab}(2,1) = \ominus$, $\mathrm{lab}(1,2) = \oplus$, $\mathrm{lab}(3,1) = \ominus$, $\mathrm{lab}(3,2) = \oplus$, $\mathrm{lab}(2,3) = \oplus$ **(l)** $G(h)$ fully labeled

Figure 6.7: Worst example of Solve-and-Force assigning positive labels first

- For $u \in \{2, \ldots, n\}$,

$$\begin{aligned} h^s(x)_u &= \sum_{v \neq u} a_{vu} x_{vu} + (a_{vu} + 1) f(x)_v \\ &= \sum_{v \neq u} a_{vu} x_{vu} + (a_{vu} + 1) \left(\sum_{w \neq v} x_w \right) \\ &= f(x)_v \\ &= \sum_{z \neq v} x_z \end{aligned}$$

Note that, when applying `TotalForce`, no label is forced in order to avoid forbidden cycles. In this way, if the variant based on `AssignLabelRandomPositive` is followed, the algorithm will iterate over all possible positive assignments until reaching the one in which the label assigned to vertex 1 is negative.



(a) $G(f)$ **(b)** $G(h, s)$

Figure 6.8: Worst example of Solve-and-Force assigning positive labels first with n vertices.

For the following example, it is considered the variant of Algorithm 6.2 that initially assigns a negative label.

Example 6.5. For this example, it is considered a network that iterated over more than one branch under Algorithm 6.2, using the variant that initially assigns a negative label.

Consider two linear Boolean networks $f, h : \{0,1\}^5 \to \{0,1\}^5$ defined by:

i	f	h
1	$x_3 + x_4$	$x_3 + x_4$
2	$x_2 + x_3 + x_4 + x_5$	$x_2 + x_3 + x_4 + x_5$
3	$x_1 + x_4$	$x_1 + x_4$
4	$x_1 + x_3$	$x_1 + x_3$
5	$x_1 + x_2 + x_3 + x_4 + x_5$	$x_2 + x_3 + x_4$

Next, when applying Algorithm 6.2 with both Boolean networks as input, it is observed that $f \neq h$ and that, for $v \in \{1,2,3,4\}$, $N_f^-(v) = N_h^-(v)$; thus, the system of equations is constructed according to (6.3).

f_1:

	a_{31}	a_{41}		
1	1	1	=	0
2	0	0	=	0
3	1	1	=	0
4	1	1	=	0
5	0	0	=	0

	a_{31}	a_{41}		
1	1	1	=	0
2	0	0	=	0
3	0	0	=	0
4	0	0	=	0
5	0	0	=	0

It is obtained $a_{31} = a_{41}$

f_2:

	a_{22}	a_{32}	a_{42}	a_{52}		
1	0	1	1	1	=	1
2	0	0	0	1	=	1
3	1	1	1	1	=	0
4	1	1	1	1	=	0
5	1	0	0	0	=	1

	a_{22}	a_{32}	a_{42}	a_{52}		
1	1	0	0	0	=	1
2	0	1	1	0	=	0
3	0	0	0	1	=	1
4	0	0	0	0	=	0
5	0	0	0	0	=	0

It is obtained $a_{22} = a_{52} = 1$ and $a_{32} = a_{42}$.

f_3:

	a_{13}	a_{43}		
1	1	1	=	0
2	0	0	=	0
3	1	1	=	0
4	1	1	=	0
5	0	0	=	0

	a_{13}	a_{43}		
1	1	1	=	0
2	0	0	=	0
3	0	0	=	0
4	0	0	=	0
5	0	0	=	0

It is obtained $a_{13} = a_{43}$.

f_4:

	a_{14}	a_{34}		
1	1	1	=	0
2	0	0	=	0
3	1	1	=	0
4	1	1	=	0
5	0	0	=	0

	a_{14}	a_{34}		
1	1	1	=	0
2	0	0	=	0
3	0	0	=	0
4	0	0	=	0
5	0	0	=	0

It is obtained $a_{14} = a_{34}$.

f_5:

	a_{25}	a_{35}	a_{45}		
1	0	1	1	=	1
2	0	0	0	=	0
3	1	1	1	=	1
4	1	1	1	=	1
5	1	0	0	=	0

	a_{25}	a_{35}	a_{45}		
1	1	0	0	=	0
2	0	1	1	=	1
3	0	0	0	=	0
4	0	0	0	=	0
5	0	0	0	=	0

It is obtained $a_{25} = 1$ and $a_{35} = a_{45}$.

The result of this system of equations $\mathcal{S}$ is shown in Figure 6.9b. Subsequently, the sequence of assigned labels and the corresponding resulting digraphs are illustrated in Figure 6.9.

For the following example, the case in which the algorithm using the variant Algorithm 6.4 does not find a solution in the first branch is presented.

Example 6.6. Consider two linear Boolean networks $f, h : \{0,1\}^5 \to \{0,1\}^5$ (Figure 6.10) defined by:

i	f	h
1	x_2	$x_1 + x_2 + x_3 + x_4 + x_5$
2	$x_4 + x_5$	$x_1 + x_2 + x_3 + x_4 + x_5$
3	$x_1 + x_4 + x_5$	$x_1 + x_4 + x_5$
4	$x_2 + x_5$	$x_2 + x_5$
5	$x_1 + x_4$	$x_3 + x_5$

Next, when applying Algorithm 6.2 with Algorithm 6.4 on both Boolean networks as input, it is observed that $f \neq h$ and that, for $v \in \{3,4\}$, $N_f^-(v) = N_h^-(v)$; thus, the system of equations is constructed according to (6.3).

f_1:

	a_{11}	a_{21}	a_{31}	a_{41}	a_{51}		
1	1	0	1	0	1	=	0
2	1	1	0	1	0	=	1
3	0	0	1	0	0	=	0
4	0	1	1	1	1	=	1
5	0	1	1	1	1	=	1

	a_{11}	a_{21}	a_{31}	a_{41}	a_{51}		
1	1	0	0	0	0	=	1
2	0	1	0	1	0	=	0
3	0	0	1	0	0	=	0
4	0	0	0	0	1	=	1
5	0	0	0	0	0	=	0

f_2:

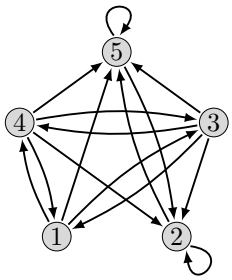
	a_{12}	a_{22}	a_{32}	a_{42}	a_{52}		
1	1	0	1	0	1	=	0
2	1	1	0	1	0	=	0
3	0	0	1	0	0	=	0
4	0	1	1	1	1	=	0
5	0	1	1	1	1	=	0

	a_{12}	a_{22}	a_{32}	a_{42}	a_{52}		
1	1	0	0	0	1	=	0
2	0	1	0	0	0	=	1
3	0	0	1	0	0	=	0
4	0	0	0	1	1	=	1
5	0	0	0	0	0	=	0

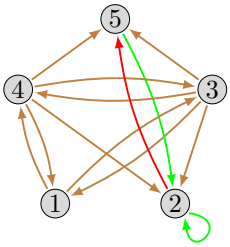
f_3:

	a_{13}	a_{43}	a_{53}		
1	1	0	1	=	0
2	1	1	0	=	0
3	0	0	0	=	0
4	0	1	1	=	0
5	0	1	1	=	0

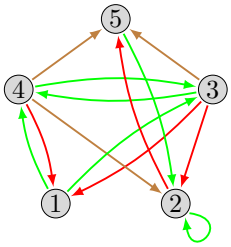
	a_{13}	a_{43}	a_{53}		
1	1	0	1	=	0
2	0	1	1	=	0
3	0	0	0	=	0
4	0	0	0	=	0
5	0	0	0	=	0



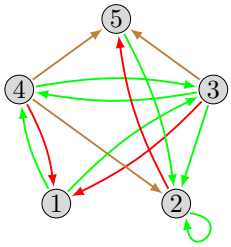
(a) $G(f)$



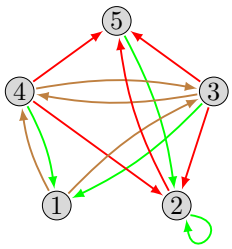
(b) $G(h)$ labeled under $\mathcal{S}$ solution



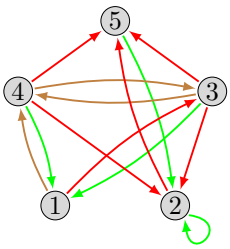
(c) $G(h)$ assigning $\text{lab}(3,1) = \ominus$ and $\text{lab}(3,2) = \ominus$, $\mathcal{S}$ does not have solution



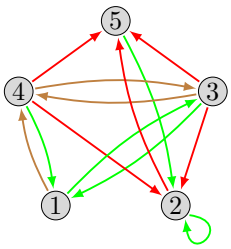
(d) $G(h)$ assigning $\text{lab}(3,1) = \ominus$ and $\text{lab}(3,2) = \oplus$, $\mathcal{S}$ does not have solution



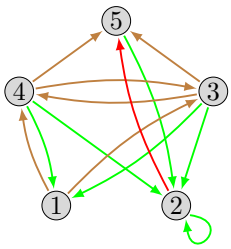
(e) $G(h)$ assigning $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \ominus$ considering `Force` and solve $\mathcal{S}$



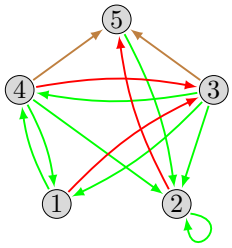
(f) $G(h)$ assigning $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \ominus$, $\text{lab}(1,3) = \ominus$, $\mathcal{S}$ does not have solution



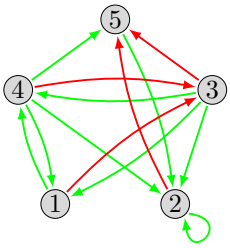
(g) $G(h)$ assigning $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \ominus$, $\text{lab}(1,3) = \oplus$, $\mathcal{S}$ does not have solution



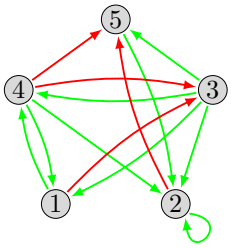
(h) $G(h)$ assigning $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \oplus$, and after `ForceTotal`



(i) $G(h)$ assigning $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \oplus$, $\text{lab}(1,3) = \ominus$



(j) $G(h)$ with $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \oplus$, $\text{lab}(1,3) = \ominus$, $\text{lab}(3,5) = \ominus$, forbidden cycle $3,4,5,3$



(k) $G(h)$ with $\text{lab}(3,1) = \oplus$, $\text{lab}(3,2) = \oplus$, $\text{lab}(1,3) = \ominus$, $\text{lab}(3,5) = \oplus$

Figure 6.9: Worst example of Solve-and-Force assigning negative labels first

f_4:

	a_{24}	a_{54}		
1	0	1	=	1
2	1	0	=	1
3	0	0	=	0
4	1	1	=	0
5	1	1	=	0

	a_{24}	a_{54}		
1	1	0	=	1
2	0	1	=	1
3	0	0	=	0
4	0	0	=	0
5	0	0	=	0

f_5:

	a_{35}	a_{55}		
1	1	1	=	1
2	0	0	=	0
3	1	0	=	0
4	1	1	=	1
5	1	1	=	1
	0	1	=	1

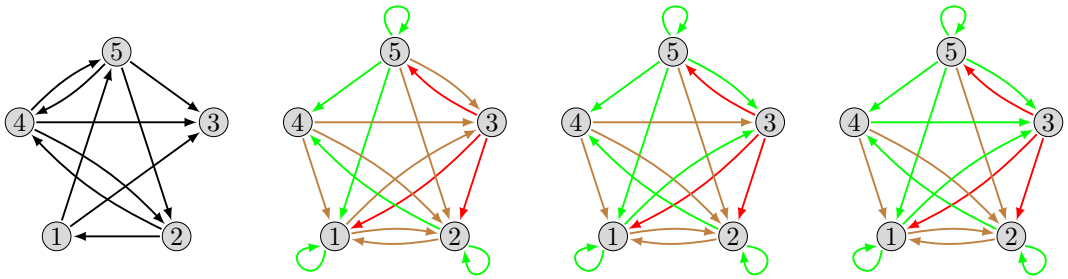
	a_{35}	a_{55}		
1	1	0	=	0
2	0	1	=	1
3	0	0	=	0
4	0	0	=	0
5	0	0	=	0

The resulting system $\mathcal{S}$ is presented in Figure 6.10b.

Next, by applying `TotalForce` (Algorithm 6.1) and immediately executing `Force`, it follows that $\text{lab}(5,3) = \text{lab}(1,3) = \oplus$ (Figure 6.10c), since if either of these arcs were assigned the opposite label, both would ultimately result in a forbidden cycle. Then, by solving the system of equations, the label $\text{lab}(4,3) = \oplus$ is fixed. Since the digraph remains update and there are still labels to be assigned, the algorithm Algorithm 6.4 is applied. By provisionally assigning the remaining labels as positive (Figure 6.10e) and executing Algorithm B.1 to construct s, the resulting update schedule is $s = \{3\}\{1,2,4,5\}$. This yields $N^-_{h^s}(1) = N^-_f(1)$, which indicates that either of the arcs $(4,1)$ or $(2,1)$ can be assigned a positive label. Consequently, the arc $(4,1)$ is selected and labeled $\oplus$ (Figure 6.10f).

Thus, by solving the system of equations under the assignment $\text{lab}(4,1) = \oplus$, $\text{lab}(2,1) = \oplus$ is fixed, this result is shown in Figure 6.10g. Then, `TotalForce` is executed, which returns the same digraph since no additional labels are forced. Since the digraph remains update, Algorithm 6.4 is applied again. As a result, when labeling all remaining arcs positively, there is no vertex under the resulting update schedule whose update dependencies of h^s yields the same dependencies of the network f. Therefore, one of the remaining arcs is randomly assigned a positive label; in this case, $\text{lab}(1,2) = \oplus$ is chosen (Figure 6.10h). However, after solving the system of equations, the resulting digraph is not update, since the forbidden cycle $1,2,4,1$ appears in Figure 6.10h. On the other hand, if $\text{lab}(1,2) = \ominus$, solving the system of equations produces the forbidden cycle $2,5,4,2$. Thus, the assignment $\text{lab}(4,1) = \oplus$ leads to a digraph that is not update (Figure 6.10i).

Next, by considering the assignment $\text{lab}(4,1) = \ominus$ (Figure 6.10j), `TotalForce` is applied, starting with the execution of `Force`, which labels the arcs $(5,2)$ and $(1,2)$ with $\oplus$ (Figure 6.10k). Then, by solving the system of equations, the remaining arcs are labeled as follows: $(2,1)$ with $\oplus$ and $(4,2)$ with $\ominus$, resulting in an update digraph (Figure 6.10l), which is a solution.

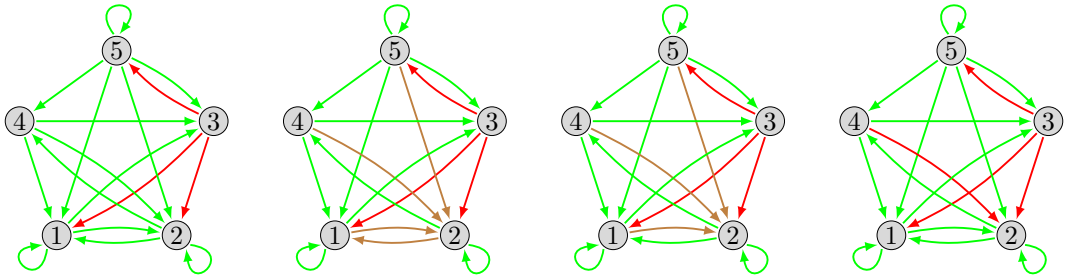


(a) $G(f)$

(b) $G(h)$ considering $\mathcal{S}$ solution

(c) $G(h)$ after `Force`

(d) $G(h)$ resolving $\mathcal{S}$ after `Force`

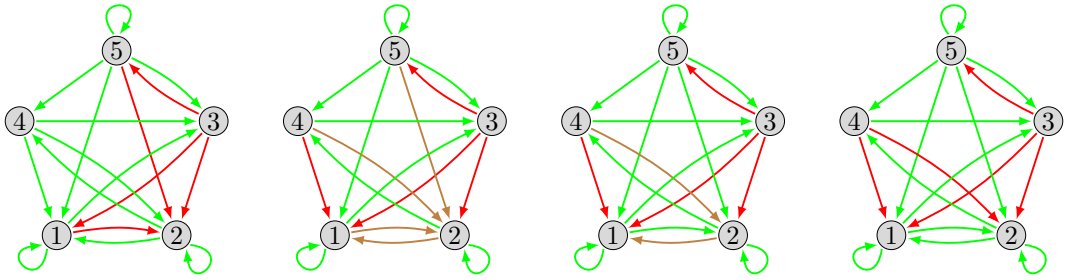


(e) $G(h)$ assigning the unlabeled arcs as $\oplus$, $s = \{3\}\{1,2,4,5\}$

(f) $G(h)$ assigning $\text{lab}(4,1) = \oplus$ given that $N^-_{h^s}(1) = N^-_f(1)$

(g) $G(h)$, $\text{lab}(4,1) = \oplus$ after solve $\mathcal{S}$

(h) $G(h)$, assigning $\text{lab}(4,1) = \oplus$ and $\text{lab}(1,2) = \oplus$ under random criteria, solve $\mathcal{S}$ results in no-update digraph



(i) $G(h)$, assigning $\text{lab}(4,1) = \oplus$ and $\text{lab}(1,2) = \ominus$, solve $\mathcal{S}$ results in no-update digraph

(j) $G(h)$, $\text{lab}(4,1) = \ominus$

(k) $G(h)$, $\text{lab}(4,1) = \ominus$, after `Force`

(l) $G(h)$, $\text{lab}(4,1) = \ominus$, after solve $\mathcal{S}$ given results of `Force`

Figure 6.10: Worst case with heuristic

6.2 Block-Iteration algorithm

The algorithm `Block-Iteration` is based on the approach developed in the work of Cabrera (2024). Specifically, it seeks vertices whose incoming arcs are more likely to receive positive labels, in order to incorporate them into the first blocks, which are constructed to be as large as possible. In this algorithm several arcs are labeled at the same time. This idea relies on the following definition introduced by Cabrera (2024), which was later extended to linear Boolean networks.

Definition 6.1. Let $f, h : \{0,1\}^n \to \{0,1\}^n$ be two linear Boolean networks, we say that an update schedule s is a k-valid update schedule if,

$$\forall u \in \{1,\ldots,n\},\ s(u) \leq k \Longrightarrow N_f^-(u) = N_{h^s}^-(u)$$

This definition is used under three main components:

1. It is used to generate blocks of an update schedule for an partial labeled digraph.
2. A recursive function that generates new schemes based on different combinations of vertices that define the block B_{k+1}.
3. A set of criteria that establish the conditions under which it is necessary to remove vertices from block B_{k+1} of a k-valid scheme.

The algorithm begins by proposing the parallel schedule as the initial solution if $h = f$(line 1). Otherwise, we begin by solving the system of linear equations to (6.2). Next, the algorithm executes `TotalForce` (Algorithm 6.1 in line 3) in order to force the labeled digraph to avoid forbidden cycles. Since in `TotalForce` after each execution of `Force` (Palma et al. (2015)) the system of equations is updated, there exists the possibility that no solution exists. Therefore, after `TotalForce`, it is checked whether the labeled digraph induced by this system of equations admits a solution (line 5). Then, it is constructed the sets A^+ and A^-, which contain the positive and negative labels induced by $\mathcal{S}$. Finally, it is executed the recursive function Algorithm 6.6 (line 11).

Algorithm 6.5: $\mathtt{s}(f, h)$

Input: Two linear Boolean networks f and h
Output: True if there exists an update schedule s such that $h^s = f$; False otherwise

```
1  if f = h then return True ;
2  Solve the equation system S induced by f and h
3  S ← TotalForce(S)
4  if S does not have a solution then
5  |  return False
6  else
7  |  if the G(h, lab) induced by S is update and no free labels exist then
8  |  |  return True
9  |  else
10 |  |  Let A+, A− the sets of positve and negative arcs, respectively, induced
   |  |   by S
11 |  |  return SubSets(f, h, 0, s_p, A+, A−)
12 |  end if
13 end if
```

Then, the Algorithm 6.6 is defined, which works recursively by considering each subset $B \subseteq B_{k+1}$ as block $k+1$ in order to construct a solution.

The Algorithm 6.6, in the first step, applies the Algorithm 6.7 in line 1, that determines which vertices must be removed from block B_{k+1} under certain necessary conditions (to be defined later) in order to build a valid solution to the problem.

If Algorithm 6.7 removes all vertices from B_{k+1}, then $B_{k+1} = \emptyset$, in which case no solution exists for the problem (line 2), and the rest of the function is not executed.

Subsequently, the algorithm iterates over each subset $B \neq \emptyset$ of B_{k+1}, from the largest (the entire set B_{k+1}) to the smallest (singleton), in order to define block $k+1$; in this way, a solution is obtained that ensures that block B_{k+1} is as large as possible.

It should be noted that, in the case where there exists $u \in B_{k+2}$ and $v \in B_{k+1} \cap (N_f^-(u) \,\Delta\, N_{h^s}^-(u))$ such that $B_{k+2} \cap N_f^+(v) \cap N_h^-(u) = \emptyset$, then it is necessary to remove some vertex from block $k+1$, which implies that $B \subsetneq B_{k+1}$. On the other hand, if there exists a vertex $z \in B_{k+2} \cap N_f^+(v) \cap N_h^-(u)$ (Figure 6.11), it is still possible that, in some later iteration after k, it holds that $v \in N_f^-(u) \cap N_{h^s}^-(u)$. Therefore, this condition is sufficient to conclude that $B \subsetneq B_{k+1}$.

In this way, given $B \subseteq B_{k+1}$, one can define $B'_{k+1} = B$, and the new scheme is given by $s' = B_1, \ldots, B_k, B'_{k+1}, V \setminus B'^*_{k+1}$, with $B'_{k+1} = B$. Then, if s' is a solution, the algorithm returns s' (line 6). On the other hand, if s' is not a solution, the Algorithm 6.6 is executed again, but now iterating over block B_{k+2} of s' (line 7).

Algorithm 6.6: `SubSets`(f, h, k, s, A^+, A^-)

Input: Two linear Boolean networks f and h, an integer k, an update schedule s, and two sets of arcs A^+ and A^-

Output: `True` if there exists an update schedule s such that $h^s = f$; `False` otherwise

$s, A^+, A^- \leftarrow$ `NextBlock`(h, f, s, k, A^+, A^-)
if $B_{k+1} \neq \emptyset$ **then**
  for $B \subseteq B_{k+1}$ ***such that*** $B \neq \emptyset$ **do**
    $B'_{k+1} \leftarrow B$
    $s' \leftarrow B_1, \ldots, B_k, B'_{k+1}, V \setminus B'^*_{k+1}$
    if $h^{s'} = f$ **then return** `True` ;
    else `SubSets`$(f, h, k+1, s', A^+, A^-)$;
  end for
end if
return `False`

The concept behind Algorithm 6.7 considers the definition of k-valid update schedule extracted from Cabrera (2024) extended to linear Boolean networks. In this way, Algorithm 6.7 first proposes a block $k+1$ and then defines conditions to remove vertices from this block.

Algorithm 6.7: `NextBlock`(f, h, k, s, A^+, A^-)

Input: Two linear Boolean networks f and h, an integer k, an update schedule s, and two sets of arcs A^+ and A^-

Output: A $(k+1)$-valid update schedule and two sets of arcs A^+ and A^-

repeat
  $s' \leftarrow s$
  $B_{k+2} \leftarrow \{v \in B_{k+1} : N^-_{h^s}(v) \neq N^-_f(v)\}$
  $B_{k+1} \leftarrow B_{k+1} \setminus B_{k+2}$
  $A^+ \leftarrow \{(u,v) \in A(h) : v \in B^*_k \ \wedge u \in B_{k+1} \cup B_{k+2}\} \cup A^+$
  $A^- \leftarrow \{(u,v) \in A(h) : u \in B^*_k \ \wedge v \in B_{k+1} \cup B_{k+2}\} \cup A^-$
  Update $\mathcal{S}$ with A^+ and A^-
  $\mathcal{S} \leftarrow$ `TotalForce`$(\mathcal{S})$
  $T_0 \leftarrow \{v \in B_{k+1} : \exists u \in B_{k+2}, (v,u) \in A^+\} \cup \{v \in B_{k+1} : \exists u \in B_{k+2}, (u,v) \in A^-\}$
  $T_1 \leftarrow \{v \in B_{k+1} : \exists u \in B_{k+2}, v \in N^-_{h^s}(u) \,\Delta\, N^-_f(u) \wedge N^-_h(u) \cap N^+_f(v) \cap (B_{k+1} \cup B_{k+2}) = \emptyset\}$
  $T_2 \leftarrow \{v \in B_{k+1} : \exists u \in B_{k+2}, \ \exists w \in B^*_k \cap (N^-_{h^s}(u) \,\Delta\, N^-_f(u)), \ N^+_f(w) \cap N^-_h(u) \cap B_{k+1} = \{v\} \ \wedge \ N^+_f(w) \cap N^-_h(u) \cap B_{k+2} = \emptyset\}$
  $B_{k+1} \leftarrow B_{k+1} \setminus (T_0 \cup T_1 \cup T_2)$
  $B_{k+2} \leftarrow B_{k+2} \cup T_0 \cup T_1 \cup T_2$
until $B_{k+1} = B'_{k+1}$;
return s

The previous algorithm begins by iterating over a loop that assigns $s' \leftarrow s$ at each iteration. If, at the end of the process, $B'_{k+1} = B_{k+1}$ (where B'_{k+1} denotes the block of the schedule s'), then the algorithm has not removed any vertex from B_{k+1}, and therefore the loop terminates.

Inside of each loop, the algorithm begins by defining the vertices that must belong to block B_{k+1} in order for s to be a $(k+1)$-valid scheme, that is, such that every $v \in B_{k+1}$ satisfies $N_f^-(v) = N_{h^s}^-(v)$ (lines 3 to 4).

Thus, in lines 5 to 6, the sets A^+ and A^- are updated based on the new blocks B_{k+1} and B_{k+2}[4]. Next, the system of equations $\mathcal{S}$ is updated with the new labels fixed in A^+ and A^-. The algorithm then proceeds to execute `TotalForce` (Algorithm 6.1) on this updated system of equations, after which the sets A^+ and A^- are updated again.

Then, in line 10, T_1 is defined as the set of vertices $v \in B_{k+1}$ such that there exists some $u \in B_{k+1}$ where v disagrees between the in-neighborhoods of u in the networks h^s and f, that is, $v \in N_f^-(u) \,\Delta\, N_{h^s}^-(u)$, and moreover all the vertices that create dependencies between v and u belong to B_k^*, that is, $N_h^-(u) \cap N_f^+(v) \subseteq B_k^*$. Equivalently, this means that there does not exist a $w \in N_h^-(u) \cap N_f^+(v) \cap (B_{k+1} \cup B_{k+2})$. Otherwise, from Figure 6.11 it can be observed that, if there exists some $w \in B_{k+1}$ (or $z \in B_{k+2}$), then in a later iteration it would still be possible to build the dependency of u with respect to v. Therefore, if it is required to create or remove such a dependency in h^s, and it also holds that $N_h^-(u) \cap N_f^+(v) \subseteq B_k^*$, the only option is to assign v to block B_{k+2}.

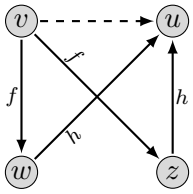


Figure 6.11: Example for T_1 definition, with $\{u, w\} \subseteq B_{k+1}$ and $\{u, z\} \subseteq B_{k+2}$.

Similarly, in line 11, T_2 is defined as the set consisting of those vertices $v \in B_{k+1}$ for which there exists $u \in B_{k+2}$ and $w \in B_k^*$ such that w disagrees between the in-neighborhoods of u in the networks h^s and f, that is, $w \in B_k^* \cap (N_f^-(u) \,\Delta\, N_{h^s}^-(u))$. Moreover, it is required that such w has exactly one vertex v that creates or eliminates dependencies between u and w ($N_f^+(w) \cap N_h^-(u) \cap B_{k+1} = \{v\}$), and that there is no vertex $z \in B_{k+2}$ such that $z \in N_f^+(w) \cap N_h^-(u)$ ($N_f^+(w) \cap N_h^-(u) \cap B_{k+2} = \emptyset$). In this case, since w has already been fixed from iteration $k-1$, that is, it can no longer be assigned to a later block, the only option is to assign vertex v to block $k+2$, as illustrated in Figure 6.12.

In this way, the necessary conditions for building the dependencies after iteration k are established, on which T_0, T_1 and T_2 are defined.

[4]Note that the labels between the sets $B_{k+1} \cup B_{k+2}$ and B_k^* remain constant regardless of the distribution of vertices between B_{k+1} and B_{k+2}.

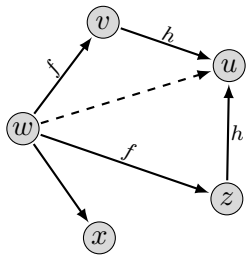


Figure 6.12: Example for T_2 definition, with $\{w\} \subseteq B_k^*$, $\{v, x\} \subseteq B_{k+1}$ and $\{u, z\} \subseteq B_{k+2}$.

Let us note that, as in `Solve-and-Force`, this algorithm not only determines whether a solution exists, but also constructs an update schedule that is a solution to the L-DENh problem.

6.2.1 Complexity and examples

In this section, the complexity of Algorithm 6.5 will be presented from both theoretical and practical perspectives. Subsequently, it will be analyzed the proportion of cases in which the solution can be computed in polynomial time, compared to the total number of simulated cases. Based on this, Example 6.7 is presented which illustrates examples of linear Boolean networks where the algorithm does not find a solution in the first branch; Examples 6.8 and 6.9 exhibits specific cases cases where $T_1 \neq \emptyset$ and $T_2 \neq \emptyset$.

Algorithm complexity

To determine the time complexity of Algorithm 6.5, note that Algorithm 6.7 involves set operations and updates of linear Boolean networks, which can be performed in polynomial time, $\mathcal{O}(n)$ and $\mathcal{O}(n^3)$, respectively; line 7 is done in $\mathcal{O}(n^3)$ given that includes to solve a system of linear equations, the execution time of `TotalForce` in the worst case is $\mathcal{O}(n^3 m)$. Therefore, given that the loops is repeated depending on the block B_{k+1} which has order on the number of vertices, the time complexity of Algorithm 6.7 is $\mathcal{O}(n^4 m)$.

With this, the complexity of Algorithm 6.6 is $\mathcal{O}(n^4 2^n m)$, since it starts executing `NextBlock` and then in the `for` loop in line 3 iterates over every subset of B_{k+1}, which accounts for $\mathcal{O}(2^{|B_{k+1}|}) \subseteq \mathcal{O}(2^n)$ combinations. Moreover, within this loop, the comparison $h^s = f$ is performed in line 6, which has a complexity of $\mathcal{O}(n^3)$. Therefore, as Algorithm 6.5 involves both the comparison $h = f$ and the call to Algorithm 6.6, its overall time complexity is $\mathcal{O}(n^4 2^n m)$ in the worst case.

Note that if the algorithm finds a solution by considering the first subset $B \subseteq B_{k+1}$ in each iteration $k \in \{1, \ldots, n\}$, then we are in the best-case scenario, which requires polynomial time, since it would depend only on conditional operations and on the execution of Algorithm 6.7 for each $B \subseteq B_{k+1}$.

Let us consider that, for each $k \in 1, \ldots, n$, the algorithm finds a solution when evaluating the first subset $B \subseteq B_{k+1}$. This situation is referred to as the algorithm finding a solution in the first branch, since the process can be represented as a tree where each node corresponds to a subset $B \subseteq B_{k+1}$ and each level of the tree is associated with a value of $k \in 1, \ldots, n$.

In Figure 6.16a, the average time required by the algorithm to find a solution is presented, considering network sizes ranging from 3 to 40 vertices and 1000 randomly generated networks for each size. The figure shows the time required to find a solution in the first branch and in the second branch. It is observed that the curve corresponding to solutions that are not found in the first branch exhibits a considerable increase in time, which reinforces the computed time complexity of $\mathcal{O}(n^4 2^n m)$ for the worst case. With respect to the curve labeled "1st branch", an almost constant growth is observed, which reinforces the fact that a solution is found in polynomial time.

Algorithm confidence

In Figure 6.16b we present the proportion of cases in which the algorithm finds a solution in the first branch compared to the total cases for each simulation, we sample 500 random linear Boolean networks for each network size from 1 to 30.

From the previous graph, it can be observed that the proportion of cases in which a solution is found by considering the first branch, relative to the total number of cases, exceeds 99%. This provides an initial indication that the algorithm is stable and that, in most cases, it finds a solution in polynomial time.

Example 6.7. Let $f, h : \{0,1\}^3 \to \{0,1\}^3$ be two linear Boolean networks defined in the next table (Figure 6.13a).

i	f	h
1	x_3	x_3
2	$x_1 + x_2 + x_3$	$x_1 + x_2 + x_3$
3	x_2	$x_1 + x_2 + x_3$

The algorithm begins by checking whether the parallel schedule is a solution to the problem. Since $f \neq h$, it proceeds to construct a system of linear equations according to (6.3):

f_1:

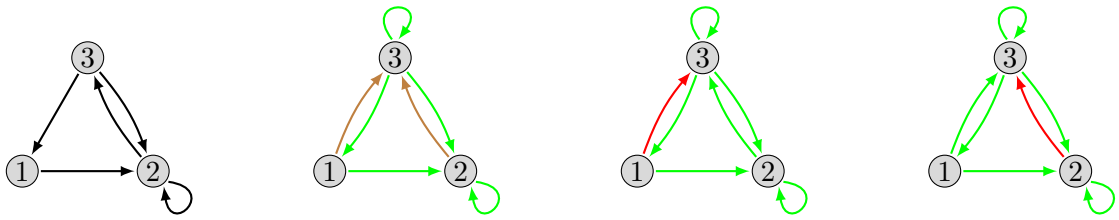
	a_{31}		
1	0	=	0
2	1	=	1
3	1	=	1

	a_{31}		
1	1	=	1
2	0	=	0
3	0	=	0

f_2:

	a_{12}	a_{22}	a_{32}		
1	1	1	0	=	0
2	0	0	1	=	1
3	1	1	1	=	1

	a_{12}	a_{22}	a_{32}		
1	1	0	0	=	1
2	0	1	0	=	1
3	0	0	1	=	1



(a) $G(f)$ **(b)** $G(h)$ after solve $\mathcal{S}$ **(c)** $G(h)$ **(d)** $G(h)$

Figure 6.13: Third example of Block-Iteration

f_3:

	a_{13}	a_{23}	a_{33}		
1	1	1	0	=	1
2	0	0	1	=	1
3	1	1	1	=	0

	a_{13}	a_{23}	a_{33}		
1	1	1	0	=	1
2	0	0	1	=	1
3	0	0	0	=	0

The solution of this system is shown in Figure 6.13b.

Then, upon applying `TotalForce`, no additional arc is labeled, neither by `Force` (Algorithm A.1) nor by the system of equations. Thus, since the resulting system is not indeterminate, the algorithm proceeds to execute `SubSets` (Algorithm 6.6) with $k = 0$, which starts by executing `NextBlock` (Algorithm 6.7). This algorithm first sets $s \leftarrow s_p$ and then constructs the block B_{k+2}. Since $N^-_{h^s}(3) \neq N^-_f(3)$, it follows that $B_2 \leftarrow \{3\}$.

Next, since there is no certainty that vertices 1 or 2 will remain in the first block, the system of equations is not updated with the labels $\text{lab}(1,3)$ and $\text{lab}(2,3)$, that is, $(1,3),(2,3) \notin A^+ \cup A^-$. Consequently, the system is not updated and $T_0 = T_1 = T_2 = \emptyset$. In this way, the output of `NextBlock` is the schedule $s = \{1,2\}\{3\}$.

Since $h^s \neq f$ still holds, the algorithm begins iterating over each subset of $B_1 = \{1,2\}$, starting with $B = \{1\}$. This choice yields the schedule $s = \{1\}\{2,3\}$, from which the execution of `SubSets` with $k = 2$ is continued, again beginning with `NextBlock`. Upon executing `NextBlock`, it is obtained that $N^-_{h^s}(2) \neq N^-_f(2)$. At this iteration, since B_2 is being constructed and B_1 is fixed, the label of arc $(1,3)$ is also fixed as $\text{lab}(1,3) = \ominus$, that is, $(1,3) \in A^-$. Consequently, solving the system of equations also fixes the label of arc $(2,3)$ as $\text{lab}(2,3) = \oplus$, which results in the forbidden cycle $3,1,2,3$ (Figure 6.13c).

Proceeding with the next subset $B = \{2\}$, the schedule $s = \{2\}\{1,3\}$ is obtained. In this case, verifying that $h^s = f$ holds, it follows that s is indeed a solution (Figure 6.13d).

Example 6.8. In the present example, it is detailed only the computation of the set T_1 in the algorithm `NextBlock` .

Let $f, h : \{0,1\}^6 \to \{0,1\}^6$ be two linear Boolean networks defined according to Figure 6.14. Consider the schedule $s' = \{1,2,3,4,5\}$ and $B_2 = \{4,6\}$. Then, according to the definition of T_1, we obtain,

$T_1 \leftarrow \{v \in \{1,2,3,5\} : \exists u \in \{4,6\},\ v \in N^-_{h^{s'}}(u) \,\Delta\, N^-_f(u) \wedge N^-_h(u) \cap N^+_f(v) \cap (B_1 \cup B_2) = \emptyset\}$

From this, we have:

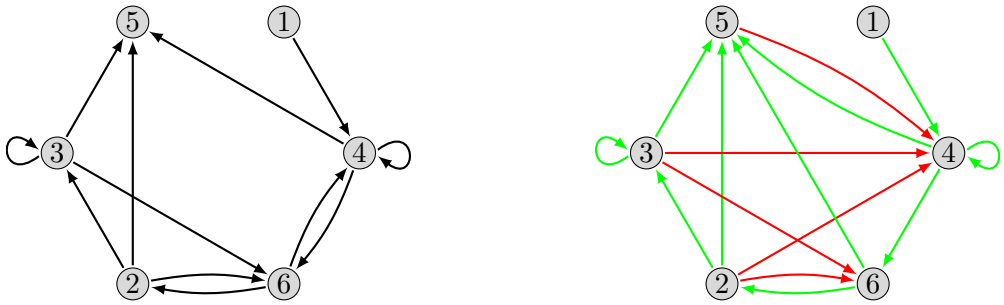
$$N^-_{h^{s'}}(3) \,\Delta\, N^-_f(3) = \{4,6\} \,\Delta \{2,4,6\} = \{1\},$$
$$N^-_{h^{s'}}(5) \,\Delta\, N^-_f(5) = \{2,3,4\} \,\Delta \{2,3,4\} = \emptyset.$$

Then, setting $v = 1$ and $u = 3$, we have $N^-_h(3) \cap N^+_f(1) \cap (\{1,2,3,5\} \cup \{4,6\}) = \emptyset$, which implies that $T_1 = \{1\}$.

Similarly, it can be shown that $T_2 = \emptyset$ (an example of its computation is provided in the next subsection).

Therefore, $B_1 \leftarrow B_1 \cup \{1\} = \{1,2,3,5\}$, and the resulting update schedule is given by $s = \{2,3,5\}\{1,4,6\}$.

Finally, since $h^s = f$, the algorithm terminates by returning s.



(a) f **(b)** (h, s)

Figure 6.14: An example of T_1 of Algorithm 6.7

Example 6.9. In the present example, it is presented only the computation of the set T_2 in the algorithm B_{k+2}. To this end, consider two linear Boolean networks $f, h : \{0,1\}^5 \to \{0,1\}^5$ defined according to Figure 6.15. Additionally, consider the schedule $s = \{1,3\}\{2,4\}\{5\}$.

This yields $T_1 = \emptyset$, and for T_2 we set $u = 5$, which gives $N^-_{h^s}(u) = \{1,3,4\}$. Notice that $B^*_1 \cap (N^-_{h^s}(u) \,\Delta\, N^-_f(u)) = \{3\}$. With this, we set $w = 3$, and compute

$$N^+_f(w) \cap N^-_h(u) \cap B_2 = \{1,2,4\} \cap \{1,2,3,4,5\} \cap \{2,4\} = \{2\},$$

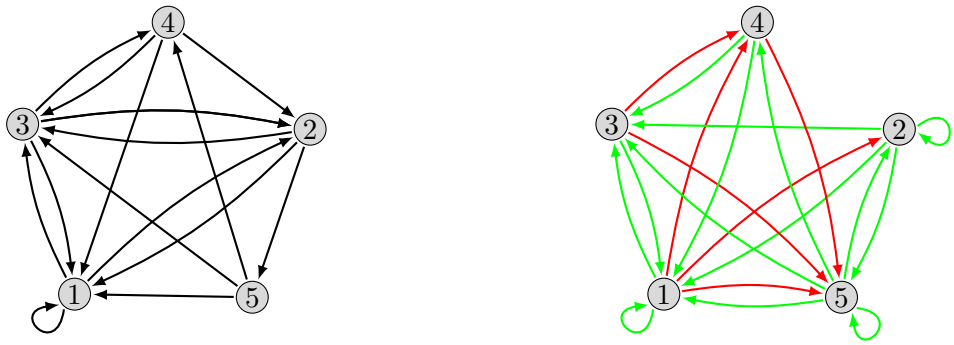
and

$$N^+_f(w) \cap N^-_h(u) \cap B_3 = \{1,2,4\} \cap \{1,2,3,4,5\} \cap \{5\} = \emptyset.$$

Therefore, $T_2 = \{1\}$, and the resulting update schedule is

$$s = \{1,3\}\{2\}\{4,5\},$$

which satisfies $h^s = f$, and the algorithm terminates.



(a) f **(b)** (h, s)

Figure 6.15: An example of T_2 of Algorithm 6.7

6.3 Comparison between Block Iteration and Solve-and-Force Algorithms

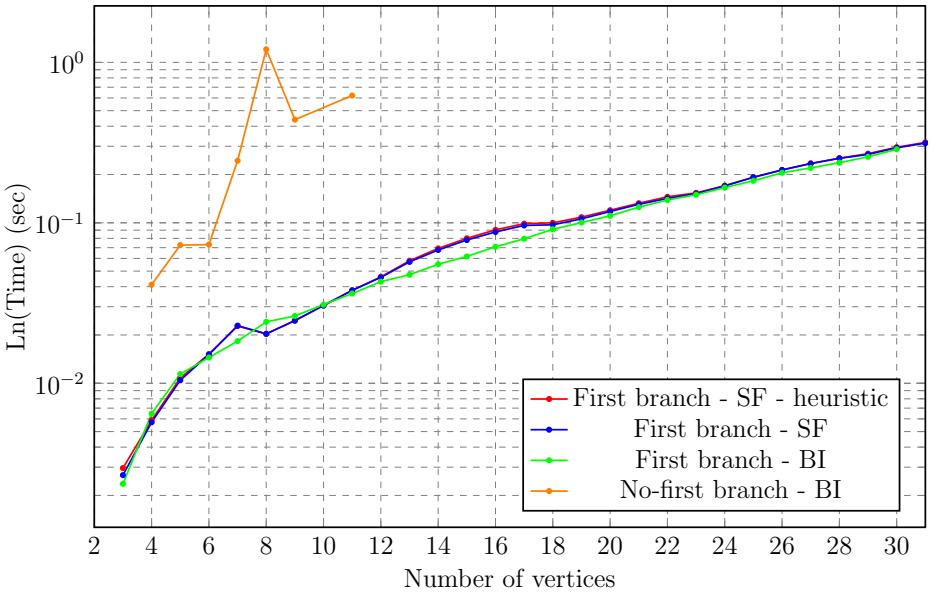
Having already presented both algorithms that find a solution to the L-DENs problem, in this section it is compared the execution time and the number of cases in which a solution is found in the first branch between both algorithms.

Next, in Figure 6.16a, it is presented the average execution time required to find a solution for Solve-and-Force (SF), Solve-and-Force with `AssignLabelPositiveArcs` and Block-Iteration (BI), considering network sizes ranging from 3 to 30 vertices. A total of 1000 randomly generated networks are considered for each network size in the case of each algorithm. Similarly, Figure 6.16b is presented, which contains the percentage per vertex for which both algorithms, Solve-and-Force and Block-Iteration, find a solution in the first branch.

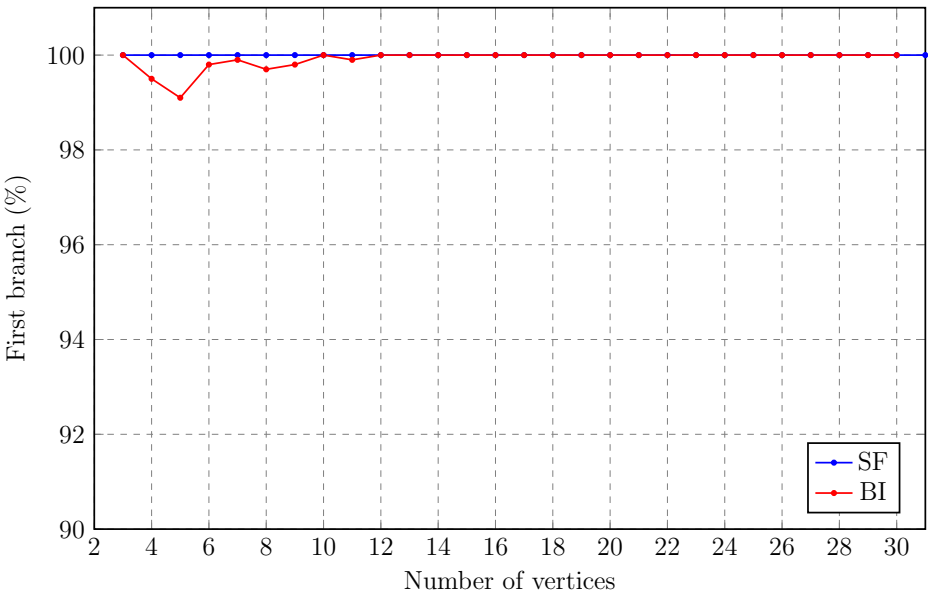
From the Figures 6.16a and 6.16b, the following observations can be made.

- Stability is observed in the execution times at which both SF and BI find a solution in the first branch
- More time to find a solution in an alternative branch to the first branch for Block-Iteration given that the worst case takes $\mathcal{O}(n^4 2^n m)$ time
- A difference is observed in the proportion of cases in which a solution is found in the first branch between both algorithms on the first vertices, being 100% for Solve-and-Force and approximately 99% on average for Block-Iteration (considering worst ratio across network size)
- `Solve-and-Force` with `AssignLabelPositiveArcs` shows slightly higher execution times than when arcs are selected randomly for labeling

In Figure 6.17, the average execution times for the cases in which the Block-Iteration and Solve-and-Force algorithms do not find a solution are presented. It is observed that,



(a) Execution time of finding a solution for Force-and-Solve and Block-Iteration



(b) Ratio of first branch of Force-and-Solve and Block-Iteration

Figure 6.16: Performance comparison of Force-and-Solve vs. Block-Iteration.

- the execution times of both algorithms exhibit stability with respect to network size
- Solve-and-Force requires less time than Block-Iteration to determine that no solution exists, with average execution times below one second in the conducted simulations

The difference in execution times between the two algorithms when determining that no solution exists is mainly due to the fact that Solve-and-Force operates as a branch-and-bound procedure. That is, a branch is no longer explored if a previously assigned label leads to a not valid solution. In contrast, Block-Iteration iterates over every possible subset combination of B_{k+1} in order to determine whether a solution exists.

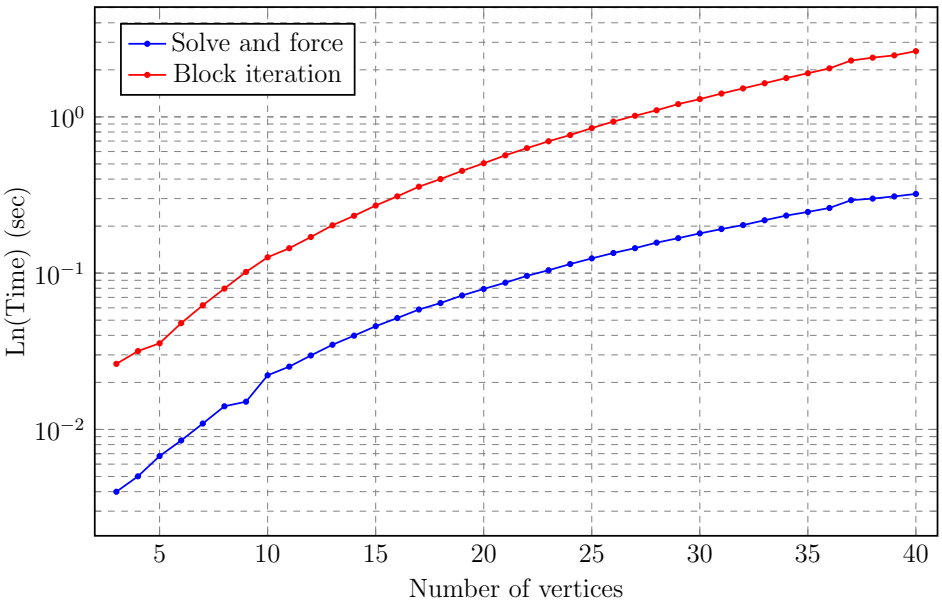


Figure 6.17: Execution time to decide if there is no solution of Force-and-Solve and Block-Iteration

Although in the simulations the Solve-and-Force algorithm finds a solution in the first branch in 100% of the cases, there exist atypical networks for which the algorithm, under any variant, does not find a solution in the first branch. We have identified that these cases are characterized by the presence of free variables when solving the system of equations. Nevertheless, the existence of a heuristic is conjectured under which it is possible to choose both the arc and its label in such a way that a solution is always found in the first branch. If such a heuristic exists, it would follow that the L-DENh problem lies in **P**. Therefore, the following conjecture is stated.

Conjeture 6.1. *L-DENh is in* **P**.

Chapter 7

Conclusion

In this work, it was studied the problem of, given a linear Boolean network f, find an update schedule s and another network h of the same family such that, when updated under this scheme, it is non-trivially dynamically equivalent to the original network (L-DEN). It was shown that solving this problem can be done in polynomial time.

Moreover, two subproblems of the L-DEN problem were studied. The first one, the L-DENs problem, consists of fixing the update schedule s and determining whether there exists a linear Boolean network h such that (h, s) is non-trivially dynamically equivalent to the original network f. It was shown that the problem can be solved by formulating a system of equations whose unknowns correspond to the dependencies of the network h.

The second subproblem corresponds to the L-DENh problem. In this case, both linear Boolean networks f and h are fixed, and the objective is to find an update schedule s such that (h, s) is non-trivially dynamically equivalent to the original network f. For this problem, it is presented two algorithms (Algorithms 6.2 and 6.5) that solve the problem. It was shown that, the solution can be found in the most of the cases in the first branch of these algorithms, which runs in polynomial time. However, there exists some atypical cases in which neither algorithm finds the solution in the first branch. Given this, it is conjectured the existence of a heuristic such that one can select an arc or a label that guarantees finding a solution in the first branch of the algorithms Solve-and-Force or Block-Iteration. Such a heuristic would directly imply that the problem lies in **P**.

The study of finding a solution to the L-DEN problem was approached by considering different architectures of the interaction graph. The first case considered is when the graph is non-trivially acyclic, where results from Cabrera (2024) were important, as some of them could be analogously applied to linear networks, leading to results that were useful for constructing solutions where the network contains only one arc with a negative label.

The second case studied considers the interaction graph of the original network as strongly connected but different from a simple cycle. In this case, the discovery that arcs with positive labels can be used to our advantage to make other arcs in the parallel

digraph not effective is crucial. This is due to the parity among the variables within the local activation functions, which are naturally part of the families of functions with addition modulo 2. This is not the case for disjunctive networks, where every local activation function is defined by the OR type, meaning that all arcs with positive labels are always effective in the parallel digraph. Therefore, a linear network with an interaction graph with these characteristics will always have a solution, which can be constructed based on a simple cycle that has an outgoing arc to another vertex (which always exist), and this can be done in polynomial time, as finding simple cycles in a strongly connected component and operating on the input neighborhoods of each vertex with symmetric difference has the same complexity.

Additionally, there arise instances in which the L-DEN problem has trivial solutions. These instances are linear networks with an interaction graph as a simple cycle, disjoint simple cycles, and simple cycles that are sinks of the same vertex. In order to demonstrate the non-existence of a solution for these cases, it is useful to note that if a simple cycle has no outgoing arcs to another vertex, this cycle must be in only one of the two blocks that define the schedule in the solution.

Thus, considering the above cases, a polynomial-time algorithm is proposed, which constructs the solution by focusing on the input neighborhoods of each vertex belonging to either a strongly connected component or an acyclic component. Additionally, a polynomial decision algorithm is proposed to determine whether or not a solution exists for the L-DEN problem, as it is sufficient to recognize the architectures of the interaction graph under which the only solution is the trivial one.

Now, let us consider the DEN problem restricted to families of disjunctive Boolean networks (D-DEN problem) and the solution proposed by Cabrera (2024). Although both solutions can be found in polynomial time, and there are cases where this solution is the same as the one proposed in this document for L-DEN (when the input neighborhoods in the interaction graph are identical), it is important to highlight that in linear Boolean networks, there are more situations where a solution can indeed be found. This is achieved by directly observing the interaction graph of the original network. For example, cases where the input neighborhoods are non-comparable (such as in complete graphs, double cycles, and double chains with loops at the extremes), as well as some particular cases like the directed path are considered.

If we now consider the L-DEN subproblem, L-DENs, its solution is based on the use of matrices defined over the finite field $\mathbb{F}_2$ to model the dynamics of the networks. In particular, for the original linear Boolean network f, the dynamics are represented as $f(x) = M(f)x$. Given a fixed update schedule s, it is then straightforward to formulate the corresponding system of equations by considering $M(h)$ as a matrix whose unknown coefficients represent the dependencies of h. It is worth emphasizing that the procedure to solve this problem differs substantially from the approach proposed for the disjunctive case (Cabrera (2024)), where the network h is constructed primarily using Lemma 3.1. As shown in Example 5.1, this property does not hold for linear Boolean networks.

Regarding the L-DENh problem, two algorithms were proposed to determine whether there exists an update schedule that provides a solution, both running in polynomial

time whenever the solution is found in the first branch. The first algorithm, Solve-and-Force, consists of formulating a system of equations over $\mathbb{F}_2$ for each vertex, whose unknowns correspond to the labels of the arcs of the network h. In this context, the results of Palma et al. (2015) and Montalva (2011) are crucial. The former introduces the algorithm `Force` (Algorithm A.1), which labels arcs in such a way that no forbidden cycle is created. The latter guarantees the existence of a labeling under which the digraph remains update (Theorem 3.1).

The second algorithm, Block-Iteration, also relies on a system of equations whose unknowns are the labels of the network h, but follows an approach closer in spirit to Cabrera (2024), based on the iterative construction of the blocks of a k-valid update schedule (Definition 6.1). That is, at each iteration k, the block B_{k+1} of the schedule s is constructed, initially considering all vertices $u \in V(h)$ such that $N^-_{h^s}(u) = N^-_f(u)$, and then transferring some of these vertices to the block B_{k+2} according to criteria that follow the ideas developed by Cabrera (2024).

Both Solve-and-Force and Block-Iteration always find a solution whenever one exists, and in most cases they do so in the first branch. However, it has not yet been proven that either algorithm always finds a solution in the first branch. It is conjectured that there exists a heuristic capable of handling the atypical cases in such a way that a solution is always found in the first branch. If such a heuristic exists, this would imply that the L-DENh problem can be solved in polynomial time.

For future work, it is interesting to study the L-DEN problem by considering update schedules with more than two blocks, as well as the problem of enumerating all solutions for a given instance of L-DEN. This direction can build upon the results presented in Section 5.1.3. One possible approach is to follow Proposition 5.6, which establishes conditions under which it is possible to refine the second block of a solution to a single vertex, or to use Proposition 5.8, which provides a framework toward the enumeration of solutions. It is also of interest to continue investigating the L-DENh problem in light of the proposed conjecture, particularly by characterizing the atypical cases in order to prove that the algorithm always finds a solution in the first branch.

Appendix A

Force

The following definitions are taken from Palma et al. (2015) for use in 5.3.

The notion of a labeled digraph can be extended to a partial labeling; thus, the labeling function can be redefined as $\text{lab} : A(f) \to \{\oplus, \ominus, \bigcirc\}$. Moreover, $A^{\oplus}_{(G,\text{lab})} = \{(u,v) \in A(f) : \text{lab}(u,v) = \oplus\}$ is defined; analogously, $A^{\ominus}_{(G,\text{lab})}$ and $A^{\bigcirc}_{(G,\text{lab})}$ are defined. With this, the support of lab is also defined as $Sup(\text{lab}) = A^{\oplus}_{(G,\text{lab})} \cup A^{\ominus}_{(G,\text{lab})}$. The arcs in the support are called labeled arcs, and the remaining arcs are called unlabeled arcs. Note that if $Sup(\text{lab}) = A(f)$, then the digraph is fully labeled.

We say that the function $\widetilde{\text{lab}} : A(f) \to \{\oplus, \ominus, \bigcirc\}$ is an extension of $\text{lab} : A(f) \to \{\oplus, \ominus, \bigcirc\}$ if, for every $a \in Sup(\text{lab})$, $\widetilde{\text{lab}}(a) = \text{lab}(a)$. If $Sup(\widetilde{\text{lab}}) = A(f)$, then $\widetilde{\text{lab}}$ is said to be a complete extension.

Given a digraph $G(f)$ and a partial labeling function lab. If $\text{lab}(a) = \bigcirc$, a simple extension $\text{lab}^{a=\oplus}$ is defined as a function such that $\forall e \in A(f) \setminus \{a\}$, $\text{lab}^{a=\oplus}(e) = \text{lab}(e)$ and $\text{lab}^{a=\oplus}(a) = \oplus$. Analogously, a simple extension $\text{lab}^{a=\ominus}$ is defined.

Algorithm A.1: `Force`$(G(f, \text{lab}))$

Input: An update digraph $G(f, \text{lab})$
Output: A maximal extension of lab

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

Algorithm A.2: `ReversePaths`(G, lab)

Input: A labeled digraph (G, lab)
Output: A matrix M encoding the existence of reverse and negative reverse paths of (G, lab)

Let M be a $|V| \times |V|$ matrix;
foreach $(u, v) \in V \times V$ **do**
  if $(v, u) \in A_{\ominus}(G, \text{lab})$ **then**
    $M(u, v) \leftarrow -1$;
  else
    if $(u, v) \in A_{\oplus}(G, \text{lab})$ **then**
      $M(u, v) \leftarrow 1$;
    else
      $M(u, v) \leftarrow \infty$;
    end if
  end if
end foreach
for $k \leftarrow 1$ **to** $|V|$ **do**
  for $i \leftarrow 1$ **to** $|V|$ **do**
    for $j \leftarrow 1$ **to** $|V|$ **do**
      if $M(i, k) \odot M(k, j) < M(i, j)$ **then**
        $M(i, j) \leftarrow M(i, k) \odot M(k, j)$;
      end if
    end for
  end for
end for
return M;

The next algorithm is introduced by Palma et al. (2015) which is adapted to input a system of equation (constructed using (6.2) which induces an labeled digraph).

Algorithm A.3: `Verify`$(\mathcal{S})$

Input: A system of equations $\mathcal{S}$ which induces a labeled digraph.
Output: True if $\mathcal{S}$ which induced a labeled digraph is update; False otherwise

Let $G(f, \text{lab})$ the graph induced by $\mathcal{S}$
foreach $u \in V(f)$ **do**
  if *There exists a reverse path from* u *to* u **then**
    return *False*
  end if
end foreach
return *True*

Appendix B

Construct schedule

Algorithm extracted from Aracena et al. (2011).

Algorithm B.1: `UpdateSchedule`(G, lab)

Input: A reduced labeled digraph $(G = (V, A), \text{lab})$ such that the labeled reoriented digraph (G_R, lab_R) has no forbidden cycle

Output: An update schedule s such that $(G, \text{lab}) = (G, \text{lab}^s)$

$H \leftarrow G_R$;
$n \leftarrow |V|$;
`ValMax` $\leftarrow$ table of size $|V(G_R)|$ in which are stored the maximal possible values of $s(v),\ v \in V(G_R)$.
foreach $v \in V$ **do**
  `ValMax`$[v] \leftarrow n$;
end foreach
while $\exists v \in V, N_H^-(v) = \emptyset$ **do**
  $s(v) \leftarrow$ `ValMax`$[v]$;
  foreach $(v, w) \in A(H)$ **do**
    if $(w, v) \in A(H)$ *is a negatiev arc* **then**
      `ValMax`$[u] \leftarrow \min\{$`ValMax`$[w],\ s(v) - 1\}$;
    else
      `ValMax`$[u] \leftarrow \min\{$`ValMax`$[w],\ s(v)\}$;
    end if
    delete arc (v, w) from H;
  end foreach
end while
$s_{\min} \leftarrow \min\{s(v) : v \in V\}$;
foreach $v \in V$ **do**
  $s(v) \leftarrow s(v) - s_{\min}+1$;
end foreach
return s;

Bibliography

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

Aracena, J., Fanchon, E., Montalva, M., and Noual, M. (2011). Combinatorics on update digraphs in boolean networks. *Discrete Applied Mathematics*, 159(6):401–409.

Aracena, J., Goles, E., Moreira, A., and Salinas, L. (2009). On the robustness of update schedules in boolean networks. *Biosystems*, 97(1):1–8.

Cabrera, L. (2024). *Study of the block-sequential operator on Boolean networks. Application to discrete network analysis*. Phd thesis, Universidad de Concepción.

Chandrasekhar, K., Kadelka, C., Laubenbacher, R., and Murrugarra, D. (2023). Stability of linear boolean networks. *Physica D: Nonlinear Phenomena*, 451:133775.

Elspas, B. (1959). The theory of autonomous linear sequential networks. *IRE Transactions on Circuit Theory*, 6(1):45–60.

Gadouleau, M., Richard, A., and Fanchon, E. (2014). Reduction and fixed points of boolean networks and linear network coding solvability.

Goles, E. and Noual, M. (2012). Disjunctive networks and update schedules. *Advances in Applied Mathematics*, 48(5):646–662.

Ho, J.-L. (2009). Global convergence for the XOR Boolean networks. *Taiwanese Journal of Mathematics*, 13(4):1271 – 1282.

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

Li, S.-Y. R., Sun, Q. T., and Shao, Z. (2011). Linear network coding: Theory and algorithms. *Proceedings of the IEEE*, 99(3):372–387.

Montalva, M. (2011). *Feedback set problems and dynamical behavior in regulatory networks*. Phd thesis, Universidad de Concepción.

Noual, M., Regnault, D., and Sené, S. (2012). Boolean networks synchronism sensitivity and xor circulant networks convergence time. *Electronic Proceedings in Theoretical Computer Science*, 90:37–52.

Palma, E., Salinas, L., and Aracena, J. (2015). Enumeration and extension of non-equivalent deterministic update schedules in boolean networks. *Bioinformatics*, 32(5):722–729.

Qi, H. and Cheng, D. (2009). Analysis and control of boolean networks: A semi-tensor product approach. In *2009 7th Asian Control Conference*, pages 1352–1356.

Robert, F. (1995). *Les systemes dynamiques discrets*, volume 19. Springer Science & Business Media.

Wiedemann, D. (1986). Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62.