Optimising Attractor Computation in Boolean Automata Networks

Kévin Perrot, Pacôme Perrotin, Sylvain Sené

Optimising attractor computation in Boolean automata networks

Kévin Perrot[2], Pacôme Perrotin[1], and Sylvain Sené[2]

[1] Aix Marseille Univ., Univ. de Toulon, CNRS, LIS, France
[2] Université Publique, Marseille, France

Abstract. This paper details a method for optimising the size of Boolean automata networks in order to compute their attractors under the parallel update schedule. This method relies on the formalism of modules introduced recently that allows for (de)composing such networks. We discuss the practicality of this method by exploring examples. We also propose results that nail the complexity of most parts of the process, while the complexity of one part of the problem is left open.

Keywords: Boolean automata networks · modularity · optimisation

1 Introduction

Boolean automata networks (BANs) are studied for their capacity to succintly expose the complexity that comes with the composition of simple entities into a network. They belong to a wide family of systems which include cellular automata and neural networks, and can be described as cellular automata with arbitrary functions and on arbitrary graph structures.

Understanding and predicting the dynamics of computing with BANs has been a focus of the scientific community which studies them, in particular since their applications include the modelling of gene regulatory networks [13,22,15] [5,6]. In those applications, fixed points of a BAN are often viewed as cellular types and limit cycles as biological rhythms [13,22]. It follows that most biological studies relying on BANs require the complete computation of their dynamics to propose conclusions. The complete computation of the dynamics of BANs is an exponentially costly process. Indeed, for n the size of a BAN, the size of its dynamics is precisely 2^n. The dynamics of a BAN is usually partitionned in two sorts of configurations: the recurring ones that are parts of attractors and either belong to a limit cycle or are fixed points; the others that evolve towards these attractors and belong to their attraction basins. The questions of characterising, computing or counting those attractors from a simple description of the network have been explored [8,1,10,7,16,2], and have been shown to be difficult problems [8,18,3,4,17].

In this paper, we propose a new method for computing the attractors of a BAN under the parallel update schedule. For any input network, this method generates another network which is possibly smaller and which is guaranteed

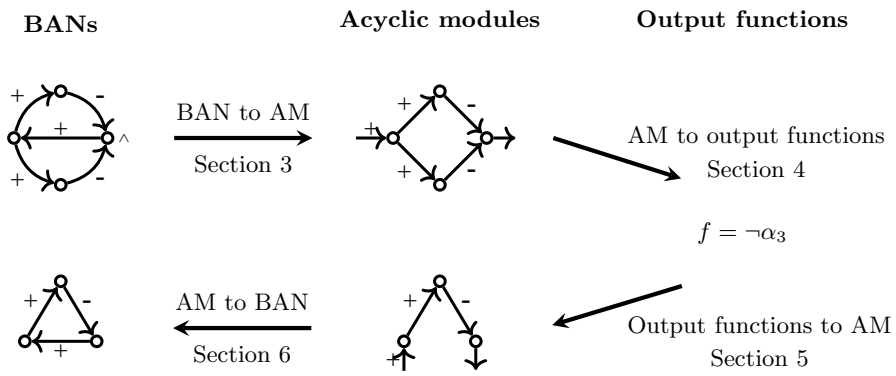

Fig. 1. Illustration of the optimisation pipeline explored in this paper. Each arrow corresponds to a part of the pipeline, and a section in this article.

to possess attractors isomorphic to those of the input network. Computing the dynamics of this smaller network therefore takes as much time as needed to compute the dynamics of the input networks, divided by some power of two.

This method uses tools and results developed in previous works by the authors [19,20]. These works involve adding inputs to BANs, in a generalisation called modules. In some cases, the entire computation of a module can be understood as functions of the states of its inputs, disregarding the network itself. In particular, a result (Theorem 16) states that two networks that have equivalent such computations share isomorphic attractors.

Section 2 starts by exposing all the definitions needed to read this paper. Section 3 explores the question of obtaining an acyclic module (AM) from a BAN. Section 4 explains how to extract so called output functions from a module. Section 5 details how to generate a minimal module from a set of output functions. Finally Section 6 shows the final step of the method, which implies constructing a BAN out of an AM and computing its dynamics. Each section explores complexity results of the different parts of the process, and details examples along the way. An illustrative outline of the paper can be found in Figure 1.

2 Definitions

2.1 Boolean functions

In this paper, we consider a Boolean function as any function $f : \mathbb{B}^A \to \mathbb{B}$, for A a finite set. An affectation x of f is a vector in $\mathbb{B}^A$. When considered as the input or output of a complexity problem, we encode Boolean functions as Boolean circuits. A *Boolean circuit* of f is an acyclic digraph in which nodes without incoming edges are labelled by an element in A, and every other node by a Boolean gate in $\{\wedge, \vee, \neg\}$, with a special node marked as the output of the

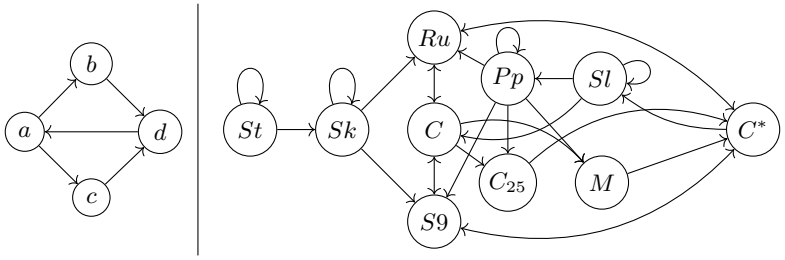

Fig. 2. On the left, the interaction digraph of F_A, as described in Example 1. On the right, the interaction digraph of F_B, as described in Example 2.

circuit. The evaluation $f(x)$ is computed by mapping x to the input nodes of the circuit, and propagating the evaluation along the circuit using the gates until the output node is reached.

2.2 Boolean automata networks and acyclic modules

Boolean automata networks BANs are composed of a set S of automata. Each automaton in S, or node, is at any time in a state in $\mathbb{B}$. Gathering those isolated states into a vector of dimension $|S|$ provides us with a configuration of the network. More formally, a *configuration* of S over $\mathbb{B}$ is a vector in $\mathbb{B}^S$. The state of every automaton is bound to evolve as a function of the configuration of the entire network. Each node has a unique function, called a local function, that is predefined and does not change over time. A *local function* is thus a function f defined as $f : \mathbb{B}^S \to \mathbb{B}$. Formally, a BAN F is a set that assigns a local function f_s over S for every $s \in S$.

BANs are usually represented by the influence that automata hold on each other. As such the visual representation of a BAN is a digraph, called an *interaction digraph*, whose nodes are the automata of the network, and arcs are the influences that link the different automata. Formally, s *influences* s' if and only if there exist two configurations x, x' such that $f_{s'}(x) \neq f_{s'}(x')$ and for all r in S, $r \neq s$ if and only if $x_r = x'_r$.

Example 1. Let $S_A = \{a, b, c, d\}$. Let F_A be the BAN defined by $f_a(x) = x_d$, $f_b(x) = f_c(x) = x_a$, and $f_d(x) = \neg x_b \vee \neg x_c$. The interaction digraph of this BAN is depicted in Figure 2 (left panel).

Example 2. Let $S_B = \{St, Sl, Sk, Pp, Ru, S9, C, C25, M, C^*\}$. Let F_B be the BAN defined by $f_{St}(x) = \neg x_{St}$, $f_{Sl}(x) = \neg x_{Sl} \vee x_{C^*}$, $f_{Sk}(x) = x_{St} \vee \neg x_{Sk}$, $f_{Pp}(x) = x_{Sl} \vee \neg x_{Pp}$, $f_{Ru}(x) = f_{S9}(x) = \neg x_{Sk} \vee x_{Pp} \vee \neg x_C \vee \neg x_{C^*}$, $f_C(x) = \neg x_{Ru} \vee \neg x_{S9} \vee \neg x_{Sl}$, $f_{C25}(x) = \neg x_{Pp} \vee x_C$, $f_M(x) = x_{Pp} \vee \neg x_C$, and $f_{C^*}(x) = \neg x_{Ru} \vee \neg x_{S9} \vee x_{C25} \vee \neg x_M$. The interaction digraph of this BAN is depicted in Figure 2 (right panel).

In the scope of this paper, BANs (and modules) are udpated according to the parallel update schedule. Formally, for F a BAN and x a configuration of F, the update of x under F is denoted by configuration $F(x)$, and defined as for all s in S, $F(x)_s = f_s(x)$.

Example 3. Consider F_A of Example 1, and $x \in \mathbb{B}^{S_A}$ such that $x = 1001$. We observe that $F_A(x) = 1111$. Configurations 1000 and 0111 are recurring and form a limit cycle of size 2, as well as configurations 0000, 0001, 1001, 1111, 1110 and 0110 that form a limit cycle of size 6.

Dynamics and attractors We define the *dynamics* of a BAN F as the digraph with $\mathbb{B}^S$ as its set of vertices. There exists an edge from x to y if and only if $F(x) = y$. An *attractor* of F is a strongly connected component of its dynamics. Computing the dynamics of a BAN from the description of its local function is an exponential process. See [21] for a more throughout introduction to BANs and related subjects.

Modules Modules were first introduced in [19]. A module M is a BAN with added inputs. It is defined on two sets: S a set of automata, and I a set of inputs, with $S \cap I = \varnothing$. Similarly to standard BANs, we can define configurations as vectors in $\mathbb{B}^S$, and we define input configurations as vectors in $\mathbb{B}^I$. A local function of a module updates itself based on a configuration x and an input configuration i, concatenated into one configuration. Formally, a local function is defined from $\mathbb{B}^{S \cup I}$ to $\mathbb{B}$. The module M defines a local function for every node s in S.

Example 4. Let M_e be the module defined on $S_e = \{p, q, r\}$ and $I = \{\alpha, \beta\}$, such that $f_p(x) = x_\alpha$, $f_q(x) = \neg x_p$, and $f_r(x) = x_q \vee \neg x_\beta$.

We represent modules with an interaction digraph, in the same way as for BANs. The interaction digraph of a module has added arrows that represent the influence of the inputs over the nodes; for every node s and every input α, the node s of the interaction digraph has an ingoing arrow labelled α if and only if α influences s, that is, there exists two input configurations i, i' such that for all β in I, $\beta \neq \alpha$ if and only if $i_\beta = {i'}_\beta$, and x a configuration such that $f_s(x \cdot i) \neq f_s(x \cdot i')$, where $\cdot$ denotes the concatenation operator.

A module is *acyclic* if and only if its interaction digraph is cycle-free.

Recursive wirings A recursive wiring over a module M is defined by a partial function $\omega : I \nrightarrow S$. The result of such a wiring is denoted $\circlearrowleft_\omega M$, a module defined over sets S and $I \setminus \mathrm{dom}(\omega)$, in which the local function of node s is denoted f'_s and defined as

$$\forall x \in \mathbb{B}^{S \cup I \setminus \mathrm{dom}(\omega)},\ f'_s(x) = f_s(x \circ \hat{\omega}),\ \text{with } \hat{\omega}(i) = \begin{cases} \omega(i) & \text{if } i \in \mathrm{dom}(\omega) \\ i & \text{if } i \in I \setminus \mathrm{dom}(\omega) \end{cases}.$$

Output functions Output functions were first introduced in [20] and present another way of computing the evolution of an acyclic module. In the Boolean case, those functions are defined on $\mathbb{B}^{I\times\{1,\dots,D\}} \to \mathbb{B}$, for I the input set of the module, and D some integer. We interpret an input in $\mathbb{B}^{I\times\{1,\dots,D\}}$ as an evaluation over $\mathbb{B}$ of a set of variables $I \times \{1,\dots,D\}$, and for $\alpha \in I$ and $d \leq D$, we denote this variable by α_d. In the context of an acyclic module M, α_d is refering to the evaluation of the input α on the dth update of the module. A vector $j \in \mathbb{B}^{I\times\{1,\dots,D\}}$ simply describes an evaluation of all the inputs of the network over D iterations. With such a vector, and $x \in \mathbb{B}^S$, it is easy to see that the acyclic module M can be updated k times in a row, for any $k \leq D$. The result of this update is denoted by $M(x, j_{[1,\dots,k]})$. The *delay* of an output function O is the maximal value in the set of all the $d \in \mathbb{N}$ for which there exists $\alpha \in I$ such that variable α_d has an influence on the computation of O. That is, there exists a couple of vectors $x, x' \in \mathbb{B}^{I\times\{1,\dots,D\}}$ which are equal except for $x_{(\alpha,d)} \neq x'_{(\alpha,d)}$, and $O(x) \neq O(x')$. Finally, for M an acyclic module defined on the sets S and I, for D a large enough integer, for $x \in \mathbb{B}^S$ and $j \in \mathbb{B}^{I\times\{1,\dots,D\}}$ some vectors, and for s a node in S, we define the output function of s, denoted O_s, as the output function with minimal delay d such that $O_s(j) = M(x, j_{[1,\dots,k]})_s$. Such a function always exists, and since it has minimal delay it is always unique.

2.3 Promise problems and classes of function problems

In this paper, we make the hypothesis that every module that is part of an instance of a complexity problem follows the property that each of its local functions has only *essential* variables. That is, a variable is included as input of the circuit encoding the function if and only if the automaton or input represented by that variable has an influence on said function. This hypothesis will be implemented throughout this paper by the use of promise problems [9], which include a decision method which can dismiss instances of the problem without that method's complexity cost being included in the complexity of the problem.

This approach is motivated by the fact that obfuscating the relation between automatons by building redundant variables in a circuit increases the complexity of most considered problems. We justify our decision in two points: first, the approach of this paper is that of providing and studying an applicable method in a context where misleading inputs in local functions are unlikely. Second, despite the inclusion of these promises, high complexity issues arise in our pipeline. As such, we consider that they help understanding the precise issues that prevent our method from being efficient.

Additionnaly, we consider the FP and FNP classes as defined in [14].

3 From BANs to AMs

The first step of our process is to unfold a BAN into an AM. This simply requires the removal of any cycle in the interaction digraph of the BAN, and their replacement by inputs. In the scope of this paper, the number of inputs generated

is required to be minimal. This is justified by the fact that the complexity of most of the problems addressed in the pipeline highly depends on the number of inputs of the considered AM.

▸ Acyclic Unfolding Functional Problem
Input: A Boolean automata network F, an integer k.
Promise: The encoding of the local functions of F only has essential variables.
Output: An acyclic module M with at most k inputs and a recursive wiring ω such that $\circlearrowleft_\omega M = F$.

Theorem 5. *The Acyclic Unfolding Functional Problem is in FNP.*

Proof. The promise of this problem allows us to compute the interaction digraph of F in polynomial time.

Consider the following simple non-deterministic algorithm: first guess a module M and a wiring ω; then check that the number of inputs in M is no more than k and that $\circlearrowleft_\omega M$ syntactically equals F.

This algorithm operates in polynomial non-deterministic time since the recursive wiring is a simple substitution of variables, and thanks to the fact that one only needs to compare $\circlearrowleft_\omega M$ and F at a syntactical level. Indeed, if any solution exists, then a solution exists with the same number of nodes, the same inputs, the same wirings, and such that the substitution operated by ω on M leads to a syntaxical copy of the local functions of F. □

Theorem 6. *The Acyclic Unfolding Functional Problem is NP-hard.*

Sketch of proof. There is a straightforward reduction from the Feedback Vertex Set problem: given G, k we construct a BAN F with OR local functions whose interaction digraph is isomorphic to G. Then the inputs of a solution M to F, k correspond to a feedback vertex set (which is given by the codomain of ω). □

Example 7. Consider S_A and F_A of Example 1. Let us define $I_A = \{\alpha\}$. Let M_A be the acyclic module that defines $f'_a(x) = x_\alpha$, $f'_b(x) = f'_c(x) = x_a$, and $f'_d(x) = \neg x_b \vee \neg x_c$. The module M_A is a valid answer to the instance $F_A, k = 1$ of the Acyclic Unfolding Functional Problem. The interaction digraph of this module is represented in Figure 3 (left panel).

Example 8. Consider S_B and F_B of Example 2. Let us define $I_B = \{\alpha_{St}, \alpha_{Sl}, \alpha_{Sk}, \alpha_{Pp}, \alpha_C, \alpha_{C^*}\}$. Let M_B be the acyclic module that defines $f'_{St}(x) = \neg x_{\alpha_{St}}$, $f'_{Sl}(x) = \neg x_{\alpha_{Sl}} \vee x_{\alpha_{C^*}}$, $f'_{Sk}(x) = x_{\alpha_{St}} \vee \neg x_{\alpha_{Sk}}$, $f'_{Pp}(x) = x_{\alpha_{Sl}} \vee \neg x_{\alpha_{Pp}}$, $f'_{Ru}(x) = f_{S9}(x) = \neg x_{\alpha_{Sk}} \vee x_{\alpha_{Pp}} \vee \neg x_{\alpha_C} \vee \neg x_{\alpha_{C^*}}$, $f'_C(x) = \neg x_{Ru} \vee \neg x_{S9} \vee \neg x_{\alpha_{Sl}}$, $f'_{C25}(x) = \neg x_{\alpha_{Pp}} \vee x_{\alpha_C}$, $f'_M(x) = x_{\alpha_{Pp}} \vee \neg x_{\alpha_C}$, and $f'_{C^*}(x) = \neg x_{Ru} \vee \neg x_{S9} \vee x_{C25} \vee \neg x_M$. The module M_B is a valid answer to the instance $F_B, k = 6$ of the Acyclic Unfolding Functional Problem. The interaction digraph of this module is represented in Figure 3 (right panel).

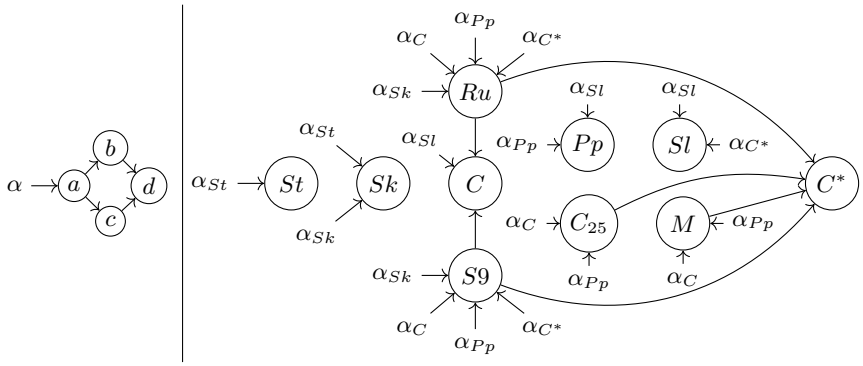

Fig. 3. On the left, the interaction digraph of M_A, as described in Example 7. On the right, the interaction digraph of M_B, as described in Example 8.

4 Output functions

Output functions were first introduced in [20]. They are a way to characterise the asymptotic behaviour of an AM as a set of Boolean functions that are computed from the local functions of the AM. Computing the output functions of an AM is a crucial step in the pipeline proposed in this work.

▸ Output Circuit Computation Problem
Input: An acyclic module M, and $X \subseteq S$ a set of output nodes.
Promise: The encoding of the local functions of M only has essential variables.
Output: An output function for each node in X, encoded as a Boolean circuit.

Theorem 9. *The Output Circuit Computation Problem is in FP.*

Sketch of proof. To build the circuit that encodes some output function of the network, we first construct a list of every output function at different delays that are required to build it, and prove that this list can be constructed in polynomial time. We then replace every entry on that list by the circuit that encodes the corresponding local function, and merge them together to obtain the circuit encoding the result. □

Example 10. Consider M_A of Example 7. Let $X_A = \{d\}$ be an instance of the Output Circuit Computation Problem. The circuit $O_d = \neg\alpha_3$ is a valid answer to that instance.

Example 11. Consider M_B of Example 8. Let $X_B = \{St, Sk, Sl, Pp, C, C^*\}$ be an instance of the Output Circuit Computation Problem. The circuits $O_{St} = \neg\alpha_{St,1}$, $O_{Sl} = \neg\alpha_{Sl,1} \vee \alpha_{C^*,1}$, $O_{Sk} = \alpha_{St,1} \vee \neg\alpha_{Sk,1}$, $O_{Pp} = \alpha_{Sl,1} \vee \neg\alpha_{Pp,1}$, $O_C = (\alpha_{Sk,2} \wedge \neg\alpha_{Pp,2} \wedge \alpha_{C,2} \wedge \alpha_{C^*,2}) \vee \neg\alpha_{Sl,1}$ and $O_{C^*} = \alpha_{C,2} \vee \neg\alpha_{Pp,2}$ taken altogether are a valid answer to that instance.

5 Optimal acyclic module synthesis

This part of the process takes as input a set of output functions and generates a module that realizes these functions with an hopefully minimal number of nodes. In this part the actual optimisation of the pipeline, if any, can be directly observed. It is also the part of the pipeline which bears most of the computational cost.

▶ Module Synthesis Problem
Input: A set I of input labels, a finite set of output functions O, encoded as Boolean circuits, defined on those labels, and k an integer.
Output: An acyclic module M with at most k nodes such that every function in O is the output function of at least one node in M.

Theorem 12. *The Module Synthesis Problem is coNP-hard.*

Proof. Consider an instance f of the Tautology problem, with I the set of propositional variables contained in f. We define f' as the output function defined on the labels I such that f' is obtained from f by substituting all variables $\alpha \in I$ by their equivalent of delay 1, α_1. Let us also define f_1 as the constant output function of delay 0 which value is always 1. We compose an instance of the Module Synthesis Problem with I the set of input labels, $O = \{f', f_1\}$ and $k = 1$. This instance has a solution if and only there exists an acyclic module with only one node such that the output function of this node is equivalent to all the output functions in O. This implies that, if the problem has a solution, f' is equivalent to f_1, which proves that f' and f are tautologies. Therefore computing the output of the Module Synthesis Problem requires solving a coNP-hard decision problem. □

Theorem 13. *The Module Synthesis Problem is in* FNP^{coNP}.

Proof. Consider the following algorithm. First, guess an acyclic module M, with size k. Compute every output function of the network, which is in FP. Then simply check that every function in O is equivalent to at least one output function in M, which requires at most $|M| \times |O|$ calls to a coNP oracle. □

It is unclear whether the synthesis problem can be proven to be in FNP or to be $\mathrm{NP}^{\mathrm{coNP}}$-hard. An attempt has been made to prove the former by using a greedy algorithm which would fuse nodes in an acyclic module, starting from a trivially large enough module. However this method seems to require a singular fusion operation which does not seem to be computable in polynomial time. This leads us to believe that a greedy algorithm would not prove the Optimal Module Synthesis Problem to be in FNP. Similarly, it is interesting to consider the open question of whether or not the Module Synthesis Problem can be proven $\mathrm{NP}^{\mathrm{coNP}}$-hard. This implies to prove, between other things, that the problem is NP-hard. This is, to us, another open problem as the Module Synthesis Problem does not seem equiped to compute the satisfaction of a Boolean formula or circuit.

This open question bears strong ressemblance to another open problem that concerns Boolean circuits. The Circuit Minimisation Problem is known to be in NP but it is not known whether the problem is in P or NP-hard, as both possibilities have deep consequences on famous open questions in theoretical computer sciences [12]. The same problem has been found to be NP-complete in both restricted (DNFs) and generalised (unrestricted Boolean circuits) variations of the Boolean circuit model [11].

There are strong similarities between acyclic modules and Boolean circuits. Both are defined on acyclic digraphs, have inputs and outputs, and compute Boolean functions. It is important to note that this analogy is misleading when talking about the optimisation of their size. Optimising a Boolean circuit requires the optimisation of a Boolean function in terms of the number of gates that computes it. Optimising an acyclic module, however, requires the optimisation of a network of functions with respect to a notion of delay of the inputs, whereas in this case one node may contain an arbitrary Boolean function. As such these problems seem too independent to provide any reduction between them.

Example 14. Consider the output function O_d defined in Example 10. Let us define M'_A as the module defined on $S'_A = \{a, b, d\}$ and $I_A = \{\alpha\}$, such that $f''_a = x_\alpha$, $f''_b = x_a$ and $f_d = \neg x_b$. The module M'_A is a valid answer to the instance $I_A, \{O_d\}, k = 3$ of the Module Synthesis Problem. The interaction digraph of this module is depicted in Figure 4 (left panel).

Example 15. Consider the output functions $O_B = \{O_{St}, O_{Sl}, O_{Sk}, O_{Pp}, O_C, O_{C^*}\}$ defined in Example 11. Let us define M'_B as the module defined on $S'_B = \{St, Sl, Sk, Pp, Ru, C25, C, C^*\}$ and $I_B = \{\alpha_{St}, \alpha_{Sl}, \alpha_{Sk}, \alpha_{Pp}, \alpha_C, \alpha_{C^*}\}$, such that $f''_{St}(x) = \neg x_{\alpha_{St}}$, $f''_{Sl}(x) = \neg x_{\alpha_{Sl}} \vee x_{\alpha_{C^*}}$, $f''_{Sk}(x) = x_{\alpha_{St}} \vee \neg x_{\alpha_{Sk}}$, $f''_{Pp}(x) = x_{\alpha_{Sl}} \vee \neg x_{\alpha_{Pp}}$, $f''_{Ru}(x) = \neg x_{\alpha_{Sk}} \vee x_{\alpha_{Pp}} \vee \neg x_{\alpha_C} \vee \neg x_{\alpha_{C^*}}$, $f''_C(x) = \neg x_{Ru} \vee \neg x_{\alpha_{Sl}}$, $f''_{C25}(x) = \neg x_{\alpha_{Pp}} \vee x_{\alpha_C}$, and $f'_{C^*}(x) = x_{C25}$. The module M'_B is a valid answer to the instance $I_B, O_B, k = 8$ of the Module Synthesis Problem. The interaction digraph of this module is depicted in Figure 4 (right panel).

6 Final wiring and analysis

The final step in the pipeline is simply to wire the module obtained in Section 5 so that the obtained networks hold isomorphic attractors to the input network. This is ensured by application of the following result.

Theorem 16 ([20]). *Let M and M' be two acyclic modules, with T and T' subsets of their nodes such that $|T| = |T'|$. If there exists a bijection g from I to I' and a bijection h from T to T' such that for every $s \in T$, O_s and $O'_{h(s)}$ have same delay, and for every input sequence j with length the delay of O_s,*

$$O_s(j) = O'_{h(s)}(j \circ g^{-1})$$

then for any function $\omega : I \to T$, the networks $\circlearrowleft_\omega M$ and $\circlearrowleft_{h \circ \omega \circ g^{-1}} M'$ have isomorphic attractors (up to the renaming of automata given by h).

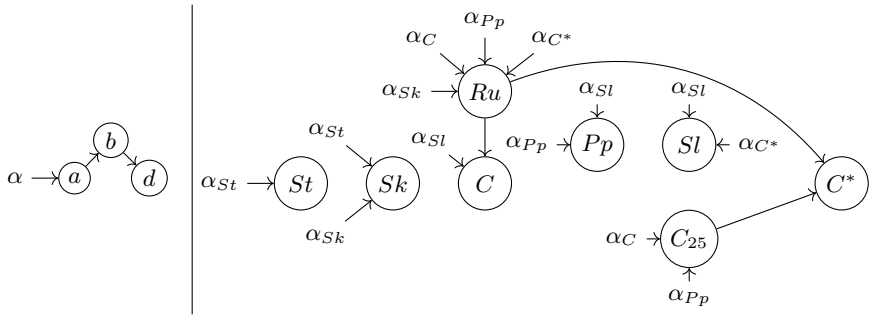

Fig. 4. On the left, the interaction digraph of M'_A, as described in Example 14. On the right, the interaction digraph of M'_B, as described in Example 15.

Applying this theorem to the current problem is simple: the module M is the module obtained in Section 3, and the module M' is the module obtained in Section 5. The set T is the set of nodes which are substituted by new inputs in the process described in Section 3. The set T' is the set of nodes in M' which are considered as the output of the module, for example when the module M' is obtained as the result of the application of the functional problem defined in Section 5.

As modules M and M' are defined over the same set of inputs, the bijection g is the identity. The bijection h is directly constructed so that for all $s \in T$, $h(s)$ in M' has an equivalent output function as s in M, which is always possible thanks to the careful structure of our pipeline. It follows quite clearly that for any $s \in T$, and for any input sequence j, $O_s(j) = O'_{h(s)}(j \circ g^{-1})$ holds, and the theorem applies.

Example 17. Consider M'_A of Example 14. Let $\omega_A(\alpha) = d$. The AN $\circlearrowleft_{\omega_A} M'_A$ is defined over $S'_A = \{a, b, d\}$ such that $f'''_a(x) = x_d$, $f'''_b(x) = x_a$, $f'''_d(x) = \neg x_b$. The interaction digraph of this module is depicted in Figure 5 (left panel).

Example 18. Consider M'_B of Example 15. Let $\omega_B(\alpha_s) = s$, for all $s \in X_B$. The AN $\circlearrowleft_{\omega_B} M'_B$ is defined over $S'_B = \{St, Sl, Sk, Pp, Ru, C25, C^*\}$ such that $f'''_{St}(x) = \neg x_{St}$, $f'''_{Sl}(x) = \neg x_{Sl} \vee x_{C^*}$, $f'''_{Sk}(x) = x_{St} \vee \neg x_{Sk}$, $f'''_{Pp}(x) = x_{Sl} \vee \neg x_{Pp}$, $f'''_{Ru}(x) = \neg x_{Sk} \vee x_{Pp} \vee \neg x_C \vee \neg x_{C^*}$, $f'''_C(x) = \neg x_{Ru} \vee \neg x_{Sl}$, $f'''_{C25}(x) = \neg x_{Pp} \vee x_C$, and $f'_{C^*}(x) = x_{C25}$. The interaction digraph of this module is depicted in Figure 5 (right panel).

This allows us to compute the attractors of any BAN by computing the dynamics of another BAN with possibly less nodes, thus dividing the number of computed configurations by some power of two. Examples throughout this paper showcase the application of the pipeline over two initial examples.

Examples 1, 7, 10, 14 and 17 show the optimisation of a simple four nodes network into a three nodes equivalent network. The optimisation proceeds here by 'compacting' two trivially equivalent nodes, b and c, into one. The resulting

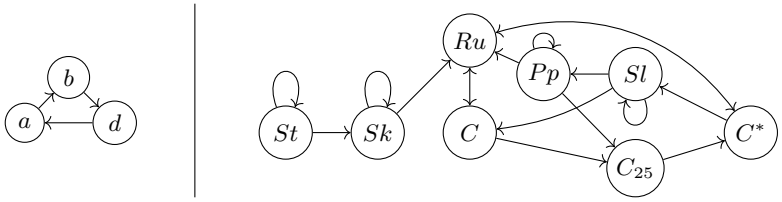

Fig. 5. On the left, the interaction digraph of F'_A, as described in Example 17. On the right, the interaction digraph of F'_B, as described in Example 18.

BAN has dynamics 2^1 times smaller than the initial network, with isomorphic attractors. Examples 2, 8, 11, 15 and 17 show the optimisation of a larger, more intricate network which is drawn from a model predicting the cell cycle sequence of fission yeast [5]. This practical example, processed through our pipeline, reduces from 10 nodes to 8. This implies a reduction in dynamics size of 2^2, while keeping isomorphic attractors. Both sets of examples are illustrated throughout the paper in Figures 2, 3, 4 and 5.

7 Conclusion

The present paper showcases an innovative way of reducing the cost of computing the attractors of Boolean automata networks. The method provides better optimisation on networks showing structural redundancies, which are removed by the pipeline. The limitations of this method are still significant; it requires solving a problem that is at least coNP-hard, and believed to be $\mathrm{FNP}^{\mathrm{coNP}}$-complete. As it presently stands, this method is not as much a convincing practical tool as it is a good argument in favor of the powerfulness of acyclic modules, their output functions, and the approaches they allow together towards the computation of BAN dynamics.

Other future perspectives include finding better complexity bounds to the Module Synthesis Problem, finding efficient heuristical or approximate implementations of the pipeline, and generalising the formalism of output functions and the optimisation pipeline to different update schedules distinct from parallel.

Acknowledgements The works of Kévin Perrot and Sylvain Sené were funded mainly by their salaries as French State agents, affiliated to Aix-Marseille Univ., Univ. de Toulon, CNRS, LIS, UMR 7020, Marseille, France (both) and to Univ. Côte d'Azur, CNRS, I3S, UMR 7271, Sophia Antipolis, France (KP), and secondarily by ANR-18-CE40-0002 FANs project, ECOS-Sud C19E02 project, STIC AmSud CoDANet 19-STIC-03 (Campus France 43478PD) project.

References

1. Aracena, J.: Maximum number of fixed points in regulatory Boolean networks. Bull. Math. Biol. **70**, 1398–1409 (2008)

2. Aracena, J., Richard, A., Salinas, L.: Number of fixed points and disjoint cycles in monotone Boolean networks. SIAM J. Discr. Math. **31**, 1702–1725 (2017)
3. Bridoux, F., Durbec, N., Perrot, K., Richard, A.: Complexity of maximum fixed point problem in Boolean Networks. In: Proc. of CiE'19. LNCS, vol. 11558, pp. 132–143. Springer (2019)
4. Bridoux, F., Gaze-Maillot, C., Perrot, K., Sené, S.: Complexity of limit-cycle problems in Boolean networks (2020), submitted, arXiv:2001.07391
5. Davidich, M.I., Bornholdt, S.: Boolean network model predicts cell cycle sequence of fission yeast. PLoS One **3**, e1672 (2008)
6. Demongeot, J., Goles, E., Morvan, M., Noual, M., Sené, S.: Attraction basins as gauges of robustness against boundary conditions in biological complex systems. PLoS One **5**, e11793 (2010)
7. Demongeot, J., Noual, M., Sené, S.: Combinatorics of Boolean automata circuits dynamics. Discr. Appl. Math. **160**, 398–415 (2012)
8. Floreen, P., Orponen, P.: Counting stable states and sizes of attraction domains in Hopfield nets is hard. In: Proc. of IJCNN'89. pp. 395–399. IEEE (1989)
9. Goldreich, O.: On promise problems: A survey. In: Theoretical computer science, pp. 254–290. Springer (2006)
10. Goles, E., Salinas, L.: Comparison between parallel and serial dynamics of Boolean networks. Theor. Comput. Sci. **396**, 247–253 (2008)
11. Ilango, R., Loff, B., Oliveira, I.C.: NP-hardness of circuit minimization for multi-output functions. In: Proc. of ECCC'20. pp. TR20–021 (2020)
12. Kabanets, V., Cai, J.: Circuit minimization problem. In: Proc. of STOC'00. pp. 73–79. ACM (2000)
13. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. J. Theor. Biol. **22**, 437–467 (1969)
14. Meggido, N., Papadimitriou, C.: A note on total functions, existence theorems, and computational complexity. Tech. report, IBM, Tech. Rep. (1989)
15. Mendoza, L., Alvarez-Buylla, E.R.: Dynamics of the genetic regulatory network for *Arabidopsis thaliana* flower morphogenesis. J. Theor. Biol. **193**, 307–319 (1998)
16. Noual, M.: Dynamics of circuits and intersecting circuits. In: Proc. of LATA'12. LNCS, vol. 7183, pp. 433–444. Springer (2012)
17. Noûs, C., Perrot, K., Sené, S., Venturini, L.: #P-completeness of counting update digraphs, cacti, and a series-parallel decomposition method. In: Proc. of CiE'20 (2020), accepted, arXiv:2004.02129
18. Orponen, P.: Neural networks and complexity theory. In: Proc. of MFCS'92. LNCS, vol. 629, pp. 50–61. Springer (1992)
19. Perrot, K., Perrotin, P., Sené, S.: A framework for (de)composing with Boolean automata networks. In: Proc. of MCU'18. LNCS, vol. 10881, pp. 121–136. Springer (2018)
20. Perrot, K., Perrotin, P., Sené, S.: On the complexity of acyclic modules in automata networks. In: Proc. of TAMC'20 (2020), accepted, arXiv:1910.07299
21. Robert, F.: Discrete Iterations: A Metric Study. Springer (1986)
22. Thomas, R.: Boolean formalization of genetic control circuits. J. Theor. Biol. **42**, 563–585 (1973)