Computing Hierarchical Transition Graphs of Asynchronous Genetic Regulatory Networks

Marco Pedicini[1](✉), Maria Concetta Palumbo[2], and Filippo Castiglione[2]

[1] Department of Mathematics and Physics, Roma Tre University, Rome, Italy
marco.pedicini@uniroma3.it

[2] CNR - Institute for Applied Computing "M. Picone", Rome, Italy

Abstract. In the field of theoretical biology the study of the dynamics of the so-called gene regulatory networks is useful to follow the relationship between the expression of a gene and its dynamic regulatory effect on the cell fate. To date, most of the models developed for this purpose, applies the synchronous update schedule while reality is far from being so. On the other hand, the more realistic asynchronous update requires to compute all possible updates at each single instant, thus bearing a much greater computational load.

In the present work, we describe a novel method that addresses the problem of efficiently exploring the dynamics of a gene regulatory network with the asynchronous update.

Keywords: SAT solver · Discrete dynamical systems
Tarjan's algorithm · Gene regulatory networks
Strongly connected components

1 Introduction

A *gene regulatory network* (GRN) can be regarded as a discrete dynamical system with a transition function $T : S \to S$ which is determined by the activation/inhibition dependencies between a given number of genes, transcription factors or RNA molecules, where S is the set of all activation profiles of the n involved elements, that hereinafter we refer to as *the genes*. In its simplest form with the number of activation levels $q = 2$ representing genes that are either *activated* or *silent*, this function T is represented as a vector of Boolean expressions. In this and other more complicated cases (*i.e.*, $q > 2$) components of T come in special forms as polynomials and the system can be considered a Polynomial Dynamical System on Finite Fields $\mathbb{F}_q$ [VCL12].

Given the transition function T, a dynamical system can be described as the graph consisting in the set of the ordered pairs $(s, T(s))$ for any $s \in S$. This graph is called the *state transition graph* (STG) of the dynamics T. The transition function T can be applied according to a synchronous schedule, meaning that all genes are updated at the same time. A more general situation occurs when we want to study the evolution in the case of asynchronous dynamics: in this case, the gene to be updated is chosen at random among all genes. In both cases, the size N of the set S of the vertices of the state transition graph of a dynamics T is exponential in the number of genes n ($N = q^n$) and any exhaustive method for the investigation of its structure rapidly becomes intractable even for small n. In the asynchronous case things are more difficult in the sense that the number of arcs in the state transition graph is much larger.

In what follows we refer to the *state transition graph structure* as the set of attractors and their properties. An *attractor* is a set A of states which coincides with the set $\bigcup_k T^k(A)$ of all its successors. In particular, in the synchronous dynamics only two forms of attractors are possible: *limit cycles* (also referred to as *cyclic attractors*) and *steady states* (*point attractors*).

Whereas any steady state of the synchronous dynamics is also a steady state in the asynchronous case, the same cannot be affirmed for synchronous dynamics limit cycles. In fact, a cycle could be or not a limit cyclic in the asynchronous dynamics. Intuitively, the reason for this behaviour is that the update function is non-deterministic and therefore it allows the dynamics to exit from cycles.

A *strongly connected component* (SCC) of a state transition graph is a set of states S, such that for any pair of states $s, s' \in S$, a directed path from s to s' exists. Among all SCCs of a state transition graph we are interested in the *maximal* ones with respect to the classical subset relationship. Let's call $\mathcal{M}(T)$ the set of all maximal SCCs in the state transition graph of the dynamics T. A strongly connected component is called *terminal* if it has no outgoing edges. Note that given a state transition graph there exists at least one terminal SCC. Given two maximal components $A, B \in \mathcal{M}(T)$ we say that A *precedes* B (indicated $A \rightarrow B$) whenever a directed path from A to B exists.

Terminal SCCs are *minimal* with respect to this ordering relationship because they are the attractors of the dynamics and by definition they are their own successors. In the literature of asynchronous networks, in order to emphasize that a limit cycle cannot coincide with an attractor but rather with a part of it, attractors which are not point-attractors are sometimes referred to as *loose attractors* [HB97].

An informative representation of the state transition graph structure is then provided by a graph in which the set of nodes/vertices is the set of maximal SCCs $\mathcal{M}(T)$ and edges/arcs are the pairs (A, B) such that A precedes B ($A \rightarrow B$). In a similar way to [BCM+13], we call *hierarchical transition graph* (HTG) of the asynchronous network this compact version of the asynchronous state transition graph structure.

In this work, we are concerned with the method to determine the hierarchical transition graph under certain hypothesis. Moreover, the algorithm is based on the possibility of generating transition paths in the state transition graph that avoid to cross previously determined cycles. Despite the number of nodes of the state transition graph is exponential in the number of genes, the algorithm we propose here is efficient if the graph satisfies few necessary hypotheses (described below) in order to visit only a small fraction of the whole graph.

The best algorithm for finding SCC in a directed graph is the algorithm from Tarjan [Tar72]. Its time complexity is $o(N + M)$, *i.e.,* linear in the number of nodes N of the resulting state transition graph, and in the number of transitions between states M [Tar72]. We achieve a similar bound in the size of the hierarchical transition graph: the linear bound is therefore obtained with respect to (i) the number of maximal SCCs, (ii) the number of nodes in any SCC and (iii) the length of transient paths connecting pairs of SCCs. This result can be attained because thanks to the use of logical Boolean expression of the dynamics we do not need to explicitly compute the whole state transition graph. Instead, by following the approach by Dubrova and Teslenko [DT11], we generate transition paths as solutions of the Boolean satisfiability problem, which can be computed by using highly specialised and optimised software called *SAT solvers*. In particular, logical expressions can be adapted during the computation, in such a way that their paths avoid nodes belonging to previously-discovered cycles.

In the following, we use the term *SAT-complexity* meaning that we count any call to the SAT solver at unitary cost; this is the same situation assumed in the case of synchronous dynamics in [DT11], where the main-loop iterates on the number of cycles of the relative state transition graph. Nevertheless, from the complexity viewpoint, any call to the SAT solver could impact on runtime with an exponential cost. Therefore, strictly speaking, the procedure cannot be said to have polynomial complexity but we can measure the complexity of the procedure in terms of calls to the SAT solver and also in terms of the fraction of the state transition graph visited in order to determine the hierarchical transition graph of the asynchronous network. If the above mentioned three conditions on the structure of SCCs are respected, the proposed variant of Tarjan's algorithm has a polynomial SAT-complexity bound on the number of genes n.

The main result of our work is the presentation of a new algorithm to determine the hierarchical transition graph of Boolean networks with several dozens of genes in the asynchronous dynamics. To this aim, we recall in Sect. 2 some definitions and formalisms on Boolean regulatory networks. In Sect. 3, we summarise the algorithm for finding limit cycles in the case of synchronous Boolean networks. Our algorithm is obtained by merging the (optimal) Tarjan approach to the determination of SCCs and the algorithm that generates paths by the SAT solver. The combination of these two approaches, described in Sect. 4, is the main result presented in this paper since it allows to determine the hierarchical transition graph of SCCs without exploring the entire state transition graph.

2 Boolean Networks as Dynamical Systems

Since Kauffman's studies, steady states and limit cycles in gene regulatory networks are regarded as set up of cellular genetic programs. Therefore there is some interest in studying the dynamics of groups of genes in the context of biological functions they are supposed to be involved in. For instance, cell differentiation is one of these functions in which the activation of the genetic transcription program brings the cell into a novel phenotypic state [Kau93, DJ02].

Definition 1. *A* gene regulatory network *of a* set of genes V *($|V| = n$) assuming values in a* scalar domain $\mathbb{K}$ *is described by components (*i.e., *by dependency functions)*

$$f_v : \mathbb{K}^n \to \mathbb{K} \text{ for any gene } v \in V.$$

Each of these functions is one component of the synchronous global updating function of the state (that for the sake of simplicity we call the dynamics*),*

$$T_{\mathsf{sync}} : \mathbb{K}^n \to \mathbb{K}^n$$

where for any state $s = (k_1, \ldots, k_n) \in \mathbb{K}^n$ we obtain a new state of the system by

$$T_{\mathsf{sync}}(s) = (f_{v_1}(s), \ldots, f_{v_n}(s)).$$

In the case of Definition 1, the dynamics is termed *synchronous* to put in evidence the fact that at each step all the components of the state are updated at once. A more flexible situation occurs when we want to study the evolution given the *asynchronous* dynamics. In this case, each component is updated independently from the others with no specific order in the sequence of updates. The resulting behaviour can be though as a non-deterministic dynamical system where starting from a configuration it is possible to reach different configurations as a consequence of the choice of the component v_i (with i chosen at random) to update.

Definition 2. *The asynchronous updating function is*

$$T_{\mathsf{async}} : \mathbb{K}^n \to \mathbb{K}^n$$

where for any state $s = (k_1, \ldots, k_n) \in \mathbb{K}^n$ we obtain a new state of the system by

$$T_{\mathsf{async}}(s) = (k_1, \ldots, f_{v_i}(s), \ldots, k_n) \qquad \text{for any choice of } i \in \{1, \ldots, n\}.$$

Definition 3. *The state transition graph of the dynamics T (either T_{sync} or T_{async}) is a graph $G(T)$ with nodes in $\mathbb{K}^n$ and edges $(s, s') \in \mathbb{K}^n \times \mathbb{K}^n$ if and only if $s' = T(s)$.*

Note that the number of nodes N in the state transition graph corresponds to the number of possible states which is the exponential in the size $|\mathbb{K}|$ to the power n, the number of genes, namely

$$N = |G(T)| = |\mathbb{K}|^n$$

regardless of the chosen dynamics (synchronous or asynchronous). The most commonly used scalar field consists of the binary one $\mathbb{K} = \mathbb{F}_2$, in which case each component of the dynamics T can be expressed as a Boolean expression, and the corresponding genetic regulatory network is simply referred to as a Boolean network (BN).

Example 1. Let us consider a genetic regulatory network on $\mathbb{F}_2$, with $|V| = 3$ and the components given by the following Boolean expressions:

$$\begin{aligned} f_1(s_1, s_2, s_3) &= \neg s_3 \wedge (s_1 \vee s_2) \\ f_2(s_1, s_2, s_3) &= s_1 \wedge s_3 \\ f_3(s_1, s_2, s_3) &= \neg s_3 \vee (s_1 \wedge s_2) \end{aligned} \tag{1}$$

For what concerns the corresponding state transition graph, we have that $N = q^n = 2^3 = 8$. Figure 1 shows the two graphs corresponding to state transition graph in both synchronous and asynchronous dynamics.

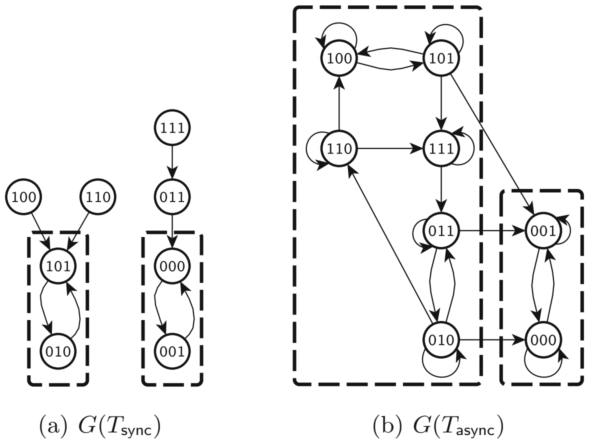
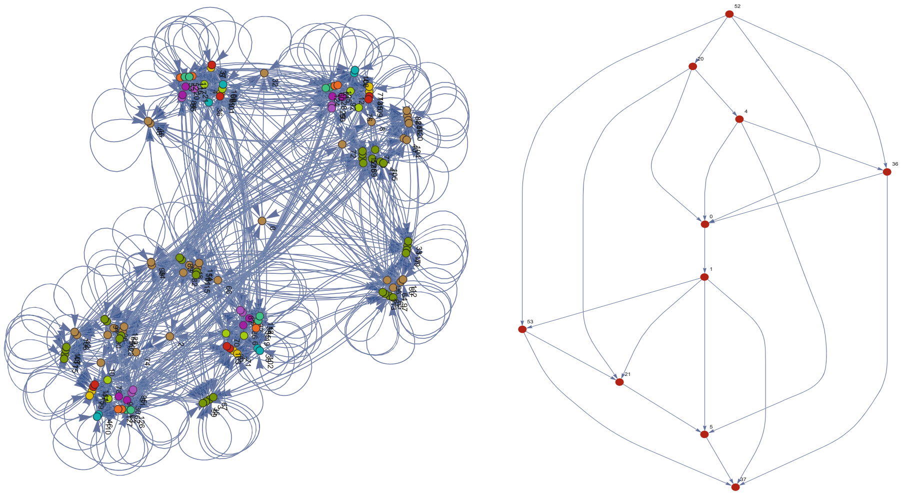


Fig. 1. Synchronous and asynchronous dynamics state transition graphs of the Boolean network specified in Eq. (1).

We are interested in several parameters which describe the state transition graph obtained starting from a given Boolean network with different transition functions and, particularly in biological applications we are concerned with the determination of the strongly connected components of the state transition graph since the biological interpretation of SCCs can be related to the stable functional characterisation of the cell behaviour [Kau93, DJ02]. Synchronous updates are rough but reasonable models of (early) response in signal networks. In Fig. 1, we compare the two state transition graphs $G(T_{\mathsf{sync}})$ and $G(T_{\mathsf{async}})$. In the first case, we see that any path terminates in a cycle, whilst in the latter case we have many self loops and two SCCs, one of which is terminal (*i.e.,* an attractor). When the state transition graph is available and feasible to manage, one can describe the relationship between pair of maximal SCCs as a partial order.

A useful definition in [GCBP+13] describes a "state-transition diagram" as a *hierarchical transition graph.* Hierarchical transition graphs are built on the analysis of the paths from initial states to attractors. In this paper, we tackle the problem of determining these SCCs in the $G(T_{\mathsf{async}})$ when the graph is too large to be explicitly computed, although its compressed form (*i.e.*, the hierarchical transition graph) can be effectively computed. See an example in Fig. 2.



(a) State Transition Graph (nodes in the same maximal SCC have identical colours)

(b) Hierarchical Transition Graph (nodes are SCC)

Fig. 2. An asynchronous random network with $n = 7$. The hierarchical transition graph in panel (b) (nodes are the maximal SCCs) is much more compact than the corresponding state transition graph in panel (a). (Color figure online)

Definition 4. *Given the state transition graph* $G(T_{\mathsf{async}})$*,* $\mathcal{M}(T)_{\mathsf{async}}$ *the set of its maximal strongly connected components and* $\rightarrow$ *a partial order relation between elements of* $\mathcal{M}(T)_{\mathsf{async}}$*, we define the* hierarchical transition graph *as the graph*

$$H(T_{\mathsf{async}}) := (\mathcal{M}(T)_{\mathsf{async}}, \rightarrow).$$

Given the specification of the dynamics of a Boolean network T_{async}, the general method for determining all the maximal SCCs of its state transition graph $G(T_{\mathsf{async}})$, consists in exhaustively analysing the graph. As already mentioned the number of nodes in $G(T)$ is $N = 2^n$, so that in order to make effective any procedure to compute $H(T_{\mathsf{async}})$, we need polynomial bounds for:

1. k where $\mathcal{M}(T_{\mathsf{async}}) = \{m_1, \ldots, m_k\}$, that is the number of maximal SCCs,
2. $l = \max_{1 \le i \le k} k_i$ where $m_i = \{s_{i,1}, \ldots, s_{i,k_i}\}$, that is the cardinality of the largest SCC,
3. d that is the length of the longest path between two SCCs; in particular, this measure is bounded by the diameter of the state transition graph $G(T_{\mathsf{async}})$.

With the above notations, we have the following

Theorem 1. *The SAT-complexity of determining $H(T_{\mathsf{async}})$ is bounded by*

$$o((k + \log_2 d)\, l\, d).$$

If the three quantities $k, l, d < n^c$ for some c, the procedure is effective with regards to the number of calls to the SAT solver; i.e., *it is polynomial-time in the number of calls to the SAT solver and we denote this execution time as $o_{SAT}(n^{3c})$.*

In the rest of the paper, we constructively show that there exists an algorithm which effectively finds the $H(T_{\mathsf{async}})$ for those Boolean networks which satisfy complexity bounds in the hypothesis.

We tackle the problem of efficiently determining the hierarchical transition graph of an asynchronous network by visiting only nodes of the state transition graph which belong to the SCCs and paths connecting them. In order to obtain the optimal solution, this work relies on the integration of (i) the best known algorithm to determine SCCs, *i.e.,* the algorithm from Tarjan and (ii) the use of Boolean formulas for expressing any path which belongs to the graph. This condition enables the use of a SAT solver in order to find assignments to the variables which satisfies the path-formula. Moreover, the path-formula can be enriched with extra conditions on the represented states, avoiding that nodes of already discovered loops appear again on the path obtained by the SAT solver. This technique allows to overcome limitations of an exhaustive approach. It has indeed been successfully applied in the determination of the limit cycles of gene regulatory networks using a synchronous update. In the asynchronous case, the results are very limited since the structure of cycles as described by the SCCs of $G(T_{\mathsf{async}})$ is much more complex.

3 Finding Limit Cycles in Synchronous Networks

The first ingredient of our work is the possibility of computing cycles in a state transition graph without exhaustively exploring it in its entirety. Indeed, for the gene regulatory networks of our interest that are composed of several dozens of genes, the corresponding state transition graph is so large that it is not even possible to store it in current digital memories.

To overcome such limitations, several approaches have been suggested, all sharing the idea to treat the updating function in a symbolic manner, like in [DT11,BGS06,ZYL+13]. To this end, there have been various proposals such as Binary Decision Diagrams [GMDC+09], Algebraic Decision Diagrams [BFG+93], Boolean Expressions and Logic Programming [HMMK13].

The approach suggested by Dubrova and Teslenko in [DT11] consisted in starting from the Boolean expressions of the components f_v of the dynamics and in defining a Boolean expression in the variables, representing relations between successive states in a path of the graph $G(T_{\mathsf{sync}})$:

$$\mathsf{StepExpression}[t] := \bigwedge_{i=1}^{n} s_i^t \leftrightarrow f_{v_i}(s_1^{t-1}, \dots s_n^{t-1}). \tag{2}$$

Then, by iterating the expression for a number of steps from $t-k$ to t, the algorithm gets the expression corresponding to a Boolean Expression which any path in $G(T_{\mathsf{sync}})$ has to satisfy:

$$\mathsf{PathExpression}[t-k,t] := \bigwedge_{i=0}^{k-1} \mathsf{StepExpression}[t-i].$$

Note that the number of variables involved in each formula depends on the number of the genes and the length of the path, namely, if we identify the size of the expression with the number of variables, we have

$$|\mathsf{PathExpression}[t-k,t]| = n^k.$$

In Algorithm 1, we present the algorithm proposed by Dubrova. It consists in a while-loop which at each iteration performs a call to a SAT solver in order to test if a certain Boolean expression admits a solution. The Boolean expression is the conjunction of the PathExpression and a condition excluding that new paths have nodes which belong to already discovered cycles. If a solution exists, then the presence of a cycle is easily verified by testing for the presence of repeated states in the solution path provided by the SAT solver (CheckPath function). When the algorithm does not find any cycle in a path, then the PathExpression doubles the path length. This last step is very important since, when the path length becomes longer than the diameter of state transition graph and at the same time the algorithm has already visited any cycle in state transition graph, then the formula F becomes unsatisfiable and the exit condition of the loop is reached. From the viewpoint of the run-time complexity, this method performs a number of iterations which in the worst case is bounded by twice the logarithm of the diameter plus the number of cycles in $G(T_{\mathsf{sync}})$. The number of iterations of the main cycle does not provide the complexity of the algorithm in the usual sense, since at each iteration we call the SAT solver which has an exponential complexity bound (on the size of the formula).

Thanks to this approach it is possible to establish limit cycles of synchronous GRNs consisting of a great number of genes. Our numerical experiments are in line with results reported in [DT11] and it is even possible to find limit cycles of a realistic network with 51 genes [PBC+10] in a matter of seconds, which would be impossible to achieve by using an exhaustive search algorithm.

In Table 1, we give an appreciation of how much better the symbolic approach is with respect to the exhaustive search: note that the number of nodes visited by our method (fourth column), is in the order of hundreds against the huge number of nodes appearing in the state transition graph ($N = 2^n$) even for networks consisting of many nodes (*e.g.,* last rows of the table).

Note that the maximal length of the paths does not need to be known in advance. In fact, although it depends on the network diameter, in practice it is found dynamically: the algorithm ends when, by doubling the length of the paths, this number exceeds the diameter and there is no path which can satisfy the formula F, since the algorithm has already found all the cycles.

Algorithm 1. Dubrova-Teslenko Algorithm to find limit cycles of $G(T_{\mathsf{sync}})$ starting from the Boolean expression of T_{sync}.

```
Require: Boolean expression PathExpression which is satisfied by any path in the
    dynamics $T_{\mathsf{sync}}$; a global stack data structure representing the intermediate state
    of the calculation of the HTG.
 1: function Cycles($T$)
 2:     Initialise
 3:     $path\_length := 1$
 4:     $F :=$ PathExpression($-path\_length, 0$)
 5:     while Satisfiability($F$) do
 6:         $(c_{-path\_length}, \ldots, c_0) :=$ SAT($F$)
 7:         if CheckPath($(c_{-path\_length}, \ldots, c_0)$) then
 8:             $c_j$ minimal state forming the loop
 9:             $Attractors(s_0) := Attractors(s_0) \wedge (s_0 \leftrightarrow c_j)$
10:             $F := F \wedge \neg Attractors(s_0)$
11:         end if
12:         if $attractor\_is\_found$ then
13:             $attractor\_is\_found :=$ false
14:         else
15:             $F :=$ PathExpression($-2path\_length, 0$)
16:             $path\_length := 2\,path\_length$
17:         end if
18:     end while
19: end function
```

Table 1. Statistics of the runs of our implementation of Dubrova Algorithm on several literature GRNs using the synchronous updating dynamics.

n	GRN name	# Limit cycles	Visited nodes	Paths (# SAT calls)	Max path length	Reference	Time
10	Fission yeast	13×1	28	16	8	[GMDC+09]	0.45
10	Mammalian cell	1×7, 1×1	29	6	16	[DT11]	0.25
12	Budding yeast	7×1	52	12	16	[DT11]	1.11
15	Arabidopsis thaliana	10×1	45	14	16	[DT11]	1.74
23	T-helper cell	3×1	42	7	16	[DT11]	0.29
40	T-helper cell receptor	1×6, 8×1	136	14	32	[DT11]	3.08
51	Th1/Th2 Switch	1×3, 3×1	97	10	64	[PBC+10]	6.06
52	Drosophila megalonoster	7×1	172	13	32	[DT11]	6.10
54	MAPK	7×8, 2×7, 4×4, 1×2, 3×1	295	22	32	[GCBP+13]	62.00

4 The Algorithm for the Asynchronous Case

What described in the above section leads us to the conclusion that at least in the synchronous case, even when the number of genes produces a large state transition graph and its size makes any tentative to determine limit cycles unreasonable, we have an effective tool which helps addressing (and solving indeed) the problem of finding limit cycles. In the asynchronous case, we have a different formula describing a single step, that is, instead of Eq. 2 we have to use the following one:

$$\mathsf{StepExpression}[t] := \bigvee_{i=1}^{n} \left(s_i^t \leftrightarrow f_{v_i}(s_1^{t-1}, \ldots s_n^{t-1}) \wedge \bigwedge_{\substack{j=1 \\ j \neq i}}^{n} s_j^t \leftrightarrow s_j^{t-1} \right). \quad (3)$$

As it is evident, this formula has a greater logical complexity with respect to the one corresponding to the synchronous case, that is Eq. 2; a fact which reflects in the more intricate nature of the $G(T_{\mathsf{async}})$.

Algorithm 1 works on the assumption that T_{sync} behaves as a deterministic function and only one possible transition can occur after a state; in this way a cycle is certainly found if the path is long enough. Moreover, once a path reaches a loop, it never leaves it (because of the uniqueness of the successor state).

In the asynchronous case, for any state we possibly have n successors (one for each gene, *i.e.,* component, *i.e.,* updating rule). The case in Fig. 1 is more complicated because *non terminal cycles* exist, making the problem of determining the structure of $G(T_{\mathsf{async}})$ more similar to the identification of SCCs in a directed graph.

The strongly connected components of a directed graph can be found using a variant of the depth-first search (the method was originally devised by R.E. Tarjan in 1972, as stated above). Since it is based on the depth-first search (DFS), it runs in time proportional to $|V| + |E|$. It is worth to mention that before Tarjan, no linear time algorithm (in the number $|E|$) was known for this problem. A straightforward approach to the same problem is to follow a path-based algorithm as initially proposed by Purdom [Pur70] and Munro [Mun71] for strong components, later deeply analysed by Gabow [Gab00]. As a consequence of the structure of Dubrova's algorithm which at each iteration generates one path, the "path oriented" approach to finding maximal SCCs in the state transition graph is the more appropriate choice.

We now describe the steps undertaken to design the algorithm which combines those of Dubrova (Algorithm 1) and Tarjan without affecting their complexity bounds. The structure of Algorithm 1 is unchanged, since the algorithm makes calls to the SAT solver to find an assignment to the variables which appears in the expression specifying the path of length *path_length* in the asynchronous dynamics. Note that we do not change the definition of $\mathsf{PathExpression}[t-k, t]$ but $\mathsf{StepExpression}[t]$ has been replaced with the one given in Eq. 3. In this case, because of the different dynamics, we have to consider that cycles can appear also in non-terminal SCCs thus, by running Algorithm 1, we have to detect cycles as part of SCCs in $G(T_{\mathsf{async}})$. In order to do so, we interleave the execution of Algorithm 1 with the path-oriented Tarjan algorithm to discover SCCs. Instead of calling the function CheckPath which tests the presence of a loop in the path (*i.e.,* line 7 of Algorithm 1) a more subtle implementation of this test is required: a new function CheckPath is given as Algorithm 2, it can be seen as the partial evaluation of the strongly connected components determination algorithm at each iteration and is our main contribution to this work.

Unfortunately, in Tarjan's Algorithm the depth first search visit is performed by exploring the graph in a precise order, that is, by following all outgoing edges of the encountered nodes. In our case, however, we have to follow the sequence of nodes in the path provided by the SAT solver. By imposing the $\neg Attractors$ condition in the formula F, we are sure that the path provided by the SAT solver does not contain any node belonging to previously discovered SCCs, nevertheless a node already encountered in a transient path can appear again in new paths.

Algorithm 2. Tarjan DFS specialised to path processing

```
Require: the global $stack$ data structure defined in Algorithm 1
 1: function CheckPath($v :: path$)
 2:     if $path \neq emptypath$ then
 3:         $w :: path' := path$
 4:         if $w$ is marked then
 5:             ProcessBacklink($v, w$)
 6:         else
 7:             Push($w$)
 8:             Mark($w$)
 9:         end if
10:         CheckPath($w :: path'$)
11:     end if
12: end function
```

A difference with a generic depth first search algorithm is that a backtracking function which closes the visit for a given node is not required since it is not possible to say if a node will not be found again in next path to be analysed, until we reach the end. Therefore, the visit of the sub-graph of paths starting in a given node is never closed. Actually, when a node is absorbed in a SCC, then the visit of the sub-graph is completed and the component is represented by the node in the hierarchical transition graph which corresponds to its SCC.

What remains to be analysed are the functions used in the ProcessBacklink, which is the main novelty of our work. The structure we use is a stack; one stack for each thread, see Fig. 3. The function Push performs the push of a node on the stack. New threads are created when the solution provided from the SAT solver contains a node that has never been encountered before in first position. A node found more than once has multiple outgoing edges in its stack and therefore the stack is a tree. In order to detect loops, we fix an ordering between threads and we add pointers from nodes belonging to threads that are in 'higher' positions to nodes which are in 'lower' positions. When this ordering is broken (*i.e.,* ThreadPrecedes is false), because the edge we would like to add connects a node in a lower thread to a higher one, then a merge operation is issued (MoveSubtree) and the subtree in higher position is moved under the node in lower position. We test for the presence of a loop in the branch of the tree

Algorithm 3. Function ProcessBacklink specialised for the path oriented version of Tarjan.

```
Require: the global stack data structure defined in Algorithm 1
 1: function ProcessBacklink($v, w$)
 2:     if TestLoop($v, w$) then
 3:         CollapsePath($w, v$)
 4:     else
 5:         if ThreadPrecedes($w, v$) then
 6:             PushPointer($v, w$)
 7:         else
 8:             MoveSubtree($w, v$)
 9:         end if
10:     end if
11: end function
```

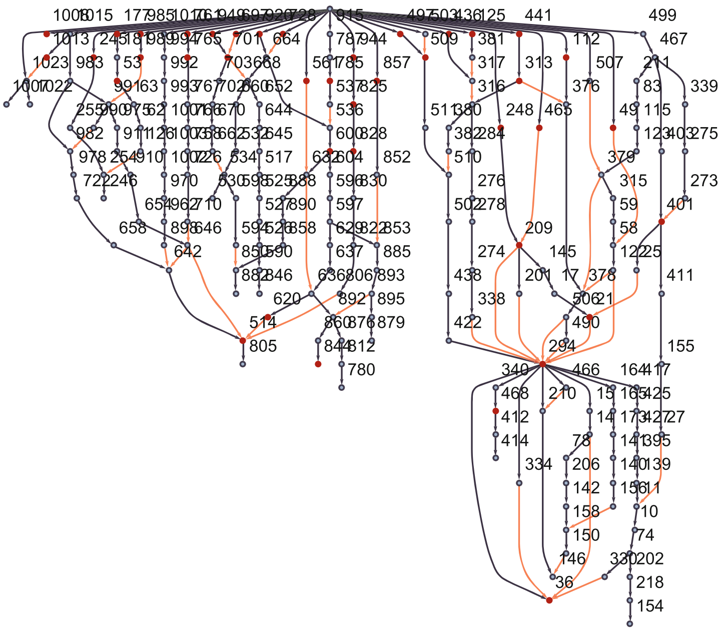


Fig. 3. Example of the data structure of the multithreaded stack (red-links represent backlinks, red nodes represent SCCs collapsed in that node). (Color figure online)

which contains the node w. If a loop is found then we collapse the path to a single *canonical node* representing an entire component by using CollapsePath.

5 Testing on Random Networks

We report about several runs performed on random Boolean networks. These are specified as directed graphs whose links represent either activations or inhibitions

Table 2. Runtime best and worst case out of 20 randomly generated GRNs with both synchronous and asynchronous dynamics. See symbol legend in the paragraph text. Missing data correspond to cases exceeding a bound in the execution time. When runtime exceeds a given time-out (heuristic dependent on the number of genes) they are shown among parentheses.

n		v	$v/2^n$	paths (# SAT calls)	max length of paths	density (K/N)	inh/(act+inh)	time
5	S	8	.25	3	4	0.36	0.11	0.09
		15	.47	6	8	0.32	0.01	0.16
	A	16	.50	6	16	0.48	0.08	1.28
		32	1.00	19	8	0.48	0.08	5.24
7	S	11	.09	4	8	0.31	0.07	0.10
		41	.32	11	8	0.35	0.29	0.30
	A	57	.45	12	16	0.29	0.00	4.39
		128	1.00	62	16	0.14	0.15	189.89
8	S	5	.02	3	4	0.19	0.17	0.10
		31	.12	8	16	0.36	0.30	0.25
	A	144	.56	12	32	0.27	0.18	25.04
		256	1.00	125	8	0.14	0.22	486.15
10	S	17	.17	7	8	0.1	0.2	0.18
		153	.15	20	16	0.17	0.18	3.81
	A	210	.21	10	64	0.2	0.1	34.99
		1024	1.00	485	16	0.23	0.48	7703.87
11	S	14	.01	5	8	0.17	0.01	0.17
		36	.02	10	8	0.17	0.01	0.53
	A	286	.14	17	64	0.19	0.0	100.22
		2048	1.00	572	16	0.15	0.11	27188.90
12	S	14	3×10^{-3}	5	8	0.17	0.01	0.17
		36	.01	10	8	0.17	0.01	0.53
	A	302	.07	13	64	0.23	0.18	56.6134
		3056	.75	300	64	0.17	0.02	49736.80
16	S	16	2×10^{-4}	4	8	0.13	0.01	0.16
		32	5×10^{-4}	7	16	0.13	0.01	0.36
	A	68	1×10^{-3}	7	32	0.14	0.09	14.07
100	S	18	1×10^{-29}	4	8	0.15	0.02	0.60
		367	3×10^{-28}	21	32	0.15	0.02	139.85
	A							
1000	S	32	3×10^{-300}	5	16	0.10	0.01	96.85
		63	6×10^{-300}	7	32	0.15	0.02	931.59
	A							

among genes. Two parameters control the generation of the graph: (i) the *density* parameter δ gives the probability that a gene influences another gene, namely that there is a link between them; (ii) the probability α that such a link is an *inhibition* (respectively $1 - \alpha$ for *activation*). What we report in Table 2 is the best and worst case in terms of performance computed on twenty independent runs for both synchronous and asynchronous updating rules. In the table S stays for synchronous update and A for asynchronous; p is the number of generated paths which coincides with the number of SAT-solver calls; m is the maximal length of paths (it relates to the diameter of the STG); δ is the density of the GRN (fraction of links w.r.t. the fully linked network); α is the fraction of inhibitors over the whole number of links in the GRN. The elapsed time t in seconds correlates to the number of *visited nodes* v; the fraction of the whole STG that was necessary to explore in order to determine the HTG is $v/2^n$ as reported in 4th column.

Moreover whereas in the synchronous case there is a direct correlation between t and m, in the asynchronous case this relationship is inverted. Note that since the networks are drawn at random, the algorithm performs efficiently when the hypotheses of Theorem 1 are satisfied. For instance this is the case of the network with $n = 16$ for which $v/2^n$ equals to $585/2^{16} \simeq 10^{-3}$ which in fact terminates in about 136 s. An opposite case is that of $n = 10$ for which the algorithm explores the totality of nodes of the state transition graph ($937/2^{10} = 90\%$) which runs for more than one hour due to the fact that the state transition graph is made of many relatively small SCCs, a fact that translates to a smaller value for m.

Note that the time values shown in the last column of Table 2 derive from an implementation of the algorithm which does not conform to the criteria of high performance and should therefore be considered as an indication of the relationships among execution time and network characteristics as just discussed. For those cases when the execution time exceeds a heuristic threshold (value dependent on the number of genes), the time spent until that limit is shown among parentheses. In the largest case of $n = 10^3$ the huge size of the search space prevented the algorithm finding a single time for the execution of the asynchronous case.

6 Conclusion

We have presented a method that can efficiently determine the attractors of a gene regulatory network in the case of the asynchronous updating rule by combining the formulation of a dynamical system in terms of satisfiability problems with an efficient algorithm for determining the strongly connected components of a graph; resulting in the possibility to determining the hierarchical transition graph without the need to entirely exploring the state transition graph. The method presented here extends previous algorithms developed for the synchronous dynamics to the asynchronous case which is regarded as being more realistic. The existence of the algorithm is a proof of Theorem 1 which summarises the link between the size of the hierarchical transition graph and the run-time.

A more detailed description of the implementation will be provided in a follow-up manuscript.

References

[BCM+13] Bérenguier, D., Chaouiya, C., Monteiro, P.T., Naldi, A., Remy, E., Thieffry, D., Tichit, L.: Dynamical modeling and analysis of large cellular regulatory networks. Chaos: Interdisc. J. Nonlinear Sci. **23**(2), 025114 (2013)

[BFG+93] Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. In: 1993 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1993, Digest of Technical Papers, pp. 188–191. IEEE (1993)

[BGS06] Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. Formal Methods Syst. Des. **28**(1), 37–56 (2006)

[DJ02] De Jong, H.: Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol. **9**(1), 67–103 (2002)

[DT11] Dubrova, E., Teslenko, M.: A SAT-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Trans. Comput. Biol. Bioinform. **8**(5), 1393–1399 (2011)

[Gab00] Gabow, H.N.: Path-based depth-first search for strong and biconnected components. Inf. Process. Lett. **74**(3–4), 107–114 (2000)

[GCBP+13] Grieco, L., Calzone, L., Bernard-Pierrot, I., Radvanyi, F., Kahn-Perlès, B., Thieffry, D.: Integrative modelling of the influence of mapk network on cancer cell fate decision. PLoS Comput. Biol. **9**(10), e1003286 (2013)

[GMDC+09] Garg, A., Mohanram, K., Di Cara, A., De Micheli, G., Xenarios, I.: Modeling stochasticity and robustness in gene regulatory networks. Bioinformatics **25**(12), i101–i109 (2009)

[HB97] Harvey, I., Bossomaier, T.: Time out of joint: attractors in asynchronous random Boolean networks. In: Proceedings of the Fourth European Conference on Artificial Life, pp. 67–75. MIT Press, Cambridge (1997)

[HMMK13] Hopfensitz, M., Müssel, C., Maucher, M.: HA Kestler: attractors in Boolean networks: a tutorial. Comput. Stat. **28**(1), 19–36 (2013)

[Kau93] Kauffman, S.A.: The Origins of Order: Self-organization and Selection in Evolution. Oxford University Press, Oxford (1993)

[Mun71] Munro, I.: Efficient determination of the transitive closure of a directed graph. Inf. Process. Lett. **1**(2), 56–58 (1971)

[PBC+10] Pedicini, M., Barrenäs, F., Clancy, T., Castiglione, F., Hovig, E., Kanduri, K., Santoni, D., Benson, M.: Combining network modeling and gene expression microarray analysis to explore the dynamics of Th1 and Th2 cell regulation. PLoS Comput. Biol. **6**(12), e1001032 (2010)

[Pur70] Purdom, P.: A transitive closure algorithm. BIT Numer. Math. **10**(1), 76–94 (1970)

[Tar72] Tarjan, R.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)

[VCL12] Veliz-Cuba, A., Laubenbacher, R.: On the computation of fixed points in Boolean networks. J. Appl. Math. Comput. **39**(1–2), 145–153 (2012)

[ZYL+13] Zheng, D., Yang, G., Li, X., Wang, Z., Liu, F., He, L.: An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. PLoS ONE **8**(4), e60593 (2013)