Efficient Computation of Attractor Fields in Coupled Boolean Networks

Luiz C. S. Rozante[1], Carlos R. P. Tovar[1(✉)], David C. Martins-Jr[1], Raphael Y. de Camargo[1], Luciana Arantes[2], and Pierre Sens[2]

[1] UFABC/CMCC, Santo André-SP, Brazil
{luiz.rozante,carlos.reynaldo,david.martins,raphael.camargo}@ufabc.edu.br
[2] Sorbonne Université, CNRS/Inria/LIP6, 4 place Jussieu, Paris 75252, France
{Luciana.Arantes,Pierre.Sens}@lip6.fr
https://www.ufabc.edu.br , https://www.lip6.fr

Abstract. Coupled Boolean Networks (CBN) are a class of discrete-time dynamical systems with a wide range of applications, ranging from distributed change detection in sensor networks to the identification of cellular patterns in tissue structures, for example. An Attractor Field corresponds to a subset of the state space of a CBN that presents certain stability properties. Attractor Fields can be used to model many events in biological systems, in particular those involving stabilization processes between interacting dynamic biological entities. Hence, an important problem consists of computing the Attractor Fields of a given CBN. Due to the exponential nature of the state space of a CBN, computing Attractor Fields is a computationally intensive task, which is currently feasible only for CBNs of unrealistic small size. In this work, we aim to develop a high-performance method for computing Attractor Fields in CBNs. In order to achieve this, we implemented algorithms based on the multi-core architecture that exploit the parallelizable nature of the problem. The experimental results suggest significant performance gains (speedup between 13 and 14) compared to the existing serial method, especially in Step 1. In addition, we achieved almost linear scalability for this Step 1, which allowed us to propose a fork-join model to optimize the method.

Keywords: Coupled Boolean Networks · Computing Attractor Fields · Systems Biology

1 Introduction

Systems consisting of networks of interacting dynamic entities are very common, with applications in several domains of knowledge. For example, phenomena that involve stabilization and/or synchronization processes are important in several areas, such as systems biology, social networks and engineering, among others [1].

Many works involving the modeling of these systems and phenomena have already been developed, including techniques such as coupled maps [7], multi-agent systems [19] and master stability equations [15], among others.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15648, pp. 377–395, 2025.
https://doi.org/10.1007/978-3-031-97000-9_24

Coupled Boolean Networks (CBNs for short) are a class of discrete-time dynamical systems, where each interacting dynamic entity corresponds to a Boolean Network (BN for short). BNs interact with each other according to a topology defined by a directed graph, and the interaction between two BNs is governed by a coupling function. CBNs have a wide range of potential applications in different areas [4]. Many studies involving CBNs focus on issues involving stability [17], synchronization [3,14,20], and control [9,11]. In general, these studies are analytical works where an attempt is made to identify and describe the necessary and sufficient conditions for stability and synchronization, dealing with cases with few BNs [2] or with restrictive inter-BN connection patterns [10].

Attractor Fields (AFs for short) correspond to stable configurations of the state space of a CBN, where the coupling signals are fixed (time invariant) and the BNs remain in the same local attractors. It is a stable configuration in the sense that the coupling signals are constant and the BNs do not change attractors, although they can change states.

AFs have many applications in various areas. For example, in systems biology, AFs are particularly useful when we have a set of dynamic biological entities that interact with each other and whose state depends on events internal to each entity and on the interactions between them. To illustrate, let us take the case of cellular patterning events, where we can model a tissue structure (e.g. a sheet of pluripotent epithelial cells) as a CBN, where each pluripotent cell corresponds to a BN, each cell type corresponds to a basin of attraction and an AF corresponds to a cellular pattern (a stable cellular differentiation pattern). This modeling has a direct application in the neuroblast segregation process in *Drosophila melanogaster* [5,16], where computing AFs is equivalent to predicting the possible cellular patterns that derive from the interaction between several cells where the neurogenic network [12,13] operates.

Tovar *et al.* [18] developed a serial method to compute AFs in CBNs. Tovar's method is general, i.e., it is capable of computing the AFs of CBNs with arbitrary topology (inter-BN connection patterns). However, the exponential nature of the state space of a CBN makes this task a computationally intensive problem, which implies that in practical terms Tovar's method is feasible only for relatively small input sizes (CBNs with about 7 BNs and linear topology). Hence, the investigation and development of high-performance computing solutions is an interesting alternative.

In this work, we present two different parallel implementations of Tovar's method based on the multi-core architecture, assuming scenarios where the user has few hardware resources. In these implementations, we preserve the three steps provided in the serial method, but we execute in parallel (using different strategies) the tasks provided for each step, including intra-step load balancing. We separately analyzed the performance of the three steps of each implementation in order to evaluate which combination of "strategy-step" leads to the best overall performance. We have performed preliminary experiments, and the results suggest that the best way to optimize Tovar's method is to adopt a

fork-join model where we run parallel (Steps 1 and 3) and serial (Step 2) code snippets.

This manuscript is structured as follows. Section 2 presents the definitions and notation necessary to describe both Tovar's method and the proposed solution. Section 3 describes Tovar's serial method. Section 4 presents our multi-core solutions of Tovar's method, including two different versions for Steps 1 and 2 of the original algorithm. Preliminary experimental results are provided in Sect. 5, and in Sect. 6 we finish with concluding remarks and suggestions for future work.

2 Preliminaries

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be a set of variables that assume values in $\{0, 1\}$. Let $\mathcal{F} = [f_1, f_2, \ldots, f_n]$ be functions such that $f_i : \{0,1\}^n \to \{0,1\}$ determines the state of the variable x_i, $1 \leq i \leq n$. The functions in $\mathcal{F}$ are called transition functions. Considering discrete time steps, the state of a variable x_i at time $t+1$ is given by

$$x_i^{t+1} = f_i(x_k^t, \ldots, x_l^t),$$

where $x_k, \ldots, x_l$ are the variables in $\mathcal{X}$ that affect x_i.

We call the discrete dynamic system $(\{0,1\}^n, \mathcal{F})$ a *Boolean Network* (BN), where the functions in $\mathcal{F}$ are all Boolean functions and the updates of the variables in $\mathcal{X}$ are done synchronously.

2.1 Coupled Boolean Network

Coupled Boolean Networks (CBN) are a class of discrete-time dynamical systems formed by a network of interacting dynamic entities, where each of them is a BN. The structure of a CBN (the connection pattern between BNs) is represented by a directed graph $G = (V, E)$.

If an edge $(i \to j) \in E$, it means that the BNs i and j are coupled in such a way that i influences j. The way in which the BN i influences the BN j, i.e., the nature of the interaction between i and j is given by a Boolean function, which we call the *coupling function*. We define a coupling function for each edge in E.

The local dynamics of an interacting entity is defined by the corresponding BN.

Suppose a CBN with m BNs among which we take a specific BN j containing n state variables. Now suppose that the BNs $u, \ldots, r$ are coupled to the BN j in such a way that $u, \ldots, r$ influence j. Then the state of a variable x_i of BN j at time $t+1$, considering both the influence coming from variables belonging to BN j itself as well as that coming from coupling signals originating from the BNs $u, \ldots, r$, is given by:

$$x_i^j(t+1) = \begin{cases} f_i^j(x_k^j(t), \ldots, x_l^j(t), y_u^j(t), \ldots, y_r^j(t)), & \text{if } i \in \mathcal{E}^j \\ f_i^j(x_k^j(t), \ldots, x_l^j(t)), & \text{if } i \in \mathcal{X}^j \setminus \mathcal{E}^j, \end{cases}$$

$1 \leq i \leq n$, $1 \leq j \leq m$, where:

- $f_i^j : \{0,1\}^{n+m} \rightarrow \{0,1\}$ is the transition function that updates the state variable x_i^j;
- $x_k^j, \ldots, x_l^j$, $1 \leq k, \ldots, l \leq n$, denote the local variables (which belong to BN j itself) that influence x_i^j;
- $y_u^j, \ldots, y_r^j$, $1 \leq u, \ldots, r \leq m$, denote the coupling signals originating from BNs $u, \ldots, r$ that affect x_i^j, where

$$y_w^j(t) = h_w^j(x_p^w(t), \ldots, x_q^w(t)),$$

  $u \leq w \leq r$, with $x_p^w, \ldots, x_q^w \in \mathcal{S}_w^j$, where

$$h_w^j : \{0,1\}^n \rightarrow \{0,1\}$$

  is the coupling function responsible for updating the coupling signal y_w^j.
- $\mathcal{X}^j$ denotes the set of state variables of the BN j;
- $\mathcal{E}^j$, $\mathcal{E}^j \subseteq \mathcal{X}^j$, denotes the input variables of the BN j (a variable of a BN is said to be an *input variable* if some coupling signal falls on it);
- $\mathcal{S}_w^j$ denotes the output variables of the BN w that influence BN j (a variable of a BN is said to be an *output variable* if it is used as a parameter by some coupling function).

Figure 1 presents a schematic representation of the coupling between BNs, where the BNs $u, \ldots, r$ are coupled to the BN j in such a way that the state variable x_i (belonging to the BN j) receives the influence of $u, \ldots, r$ through the coupling signals $y_u^j, \ldots, y_r^j$, which are determined by the coupling functions $h_u^j, \ldots, h_r^j$, respectively.

The state of a CBN is given by the state of each of the variables in all BNs. The state transition in a CBN is given by the synchronous application of all transition functions (in each BN) and all coupling functions (for all interactions between BNs).

2.2 Stability in CBNs and Target Problem

Our target problem consists in, given a CBN as input, we aim to find stable configurations of its state space, i.e., subsets of the state space that satisfy certain stability properties. These subsets of the state space are called *Attractor Fields* (AF).

There are many ways to define stability in CBNs. For example, we can do so based on properties of the coupling signals or using steady states (local and/or global), among others. In this work, we assume that the concept of stability is associated with the invariability of the coupling signals and the non-alternation of local attractors [8]. More precisely, an AF is a subset of the state space that satisfies the following properties:

- The coupling signals are time-invariant, i.e., the signals produced by the coupling functions do not change when the transition and coupling functions are applied. In other words, the coupling signals are stationary.

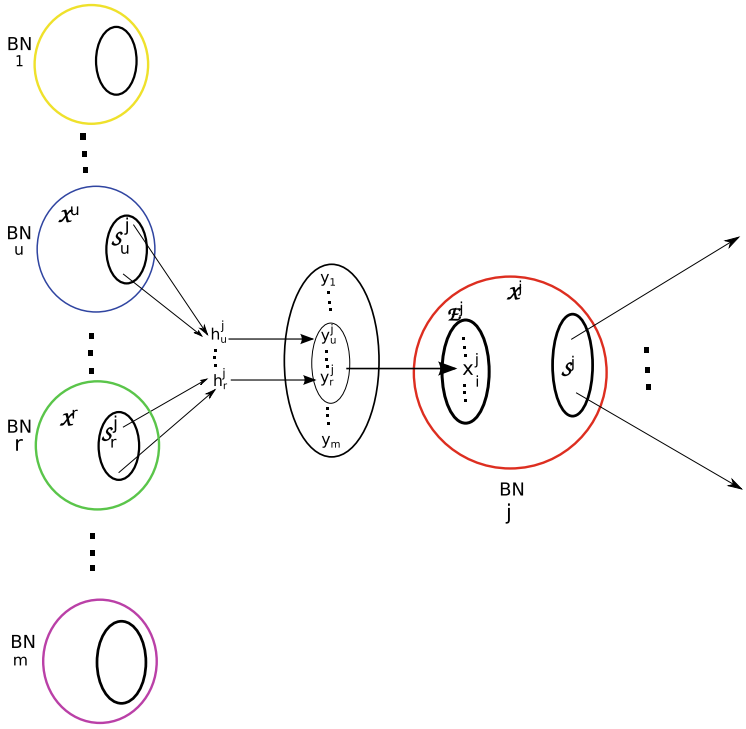

Fig. 1. Illustrative diagram of the coupling between BNs.

– There is no change in local attractors in BNs when the transition and coupling functions are applied. Since the attractors can be periodic (non-stationary[1]), then there can be state transitions in BNs when the transition and coupling functions are applied. However, the BNs do not change attractors, they remain the same in all BNs. In other words, the system "stabilizes" in the attractors in which the BNs are located, although the BNs can change states.

3 Serial Method

The method is defined in three steps, detailed in the following subsections. More details on how the serial method works can be found in Tovar's work [18].

3.1 Serial Step 1: Computing Local Attractors

The input of Step 1 is a CBN and it returns as output the set of local attractors for each BN, as illustrated in Fig. 2, where we see on the left a representation of

[1] A global stationary state, i.e., a configuration where both the coupling signals and the states of the BNs remain the same, is a particular case of an Attractor Field.

an example CBN with 3 BNs, which are distinguished by colors (red, green and blue) and connected by coupling functions $h_{1,2}$, $h_{2,3}$ and $h_{3,1}$. On the right, an example of the output from Step 1: A^1, A^2, and A^3 denote the local attractors associated with the red, green, and blue BNs, respectively.

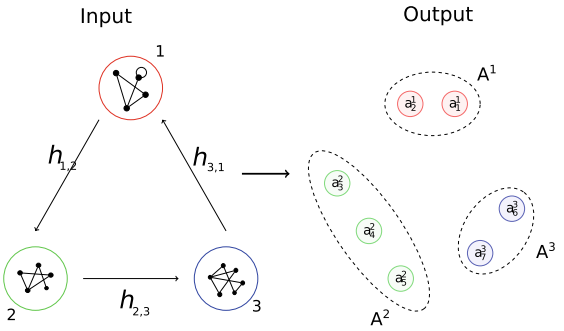

Fig. 2. An example of input and output of Step 1. (Color figure online)

Given a CBN, in Step 1 we use the Dubrova and Teslenko [6] algorithm to calculate the local attractors of each BN. Since BNs can receive external signals generated by coupling functions, it was necessary to adapt the Dubrova and Teslenko algorithm because, in its original version, it does not provide the treatment of this type of external signal.

Here, it is worth reinforcing an important restriction: we assume that the coupling signals are invariant in time, i.e., their values are constant. This restriction derives from the properties that characterize the concept of stability in attractor fields.

Hence, since a BN can receive several coupling signals coming from other BNs, if a BN r receives c coupling signals, in Step 1 we compute (applying the adapted Dubrova and Teslenko algorithm) the local attractors of r for all 2^c possible combinations of fixed coupling signal that fall on r.

3.2 Serial Step 2: Computing the Compatible Attractor Pairs

Given a collection $\mathcal{A}^1, \ldots, \mathcal{A}^m$ of local attractor sets produced in Step 1 and a graph $G = (V, E)$ representing the CBN topology, in Step 2 we basically do:

(i) given an edge $(i \to j) \in E$, then we compute all pairs $\mathcal{A}^i \times \mathcal{A}^j$ of attractors possible to form with $\mathcal{A}^i$ and $\mathcal{A}^j$, where $\mathcal{A}^i$ is the set of local attractors of BN i and $\mathcal{A}^j$ is the set of local attractors of BN j;

(ii) for each pair $(\mathsf{a}^i_w, \mathsf{a}^j_z) \in \mathcal{A}^i \times \mathcal{A}^j$, check whether it satisfies the *stability condition*, which can be described as follows: the output value produced by the local attractor $\mathsf{a}^i_w \in \mathcal{A}^i$ is equal to the input value expected by the local attractor $\mathsf{a}^j_z \in \mathcal{A}^j$.

If two attractors $(\mathsf{a}_w^i, \mathsf{a}_z^j) \in \mathcal{A}^i \times \mathcal{A}^j$ satisfy the stability condition, then we say that they form a *compatible attractor pair*. More details about the stability condition and compatible attractors can be found in Tovar's work [18].

The output of Step 2 is a set of compatible attractor pairs, which is represented by a directed graph $G' = (V', E')$. The inputs and outputs of Step 2 are illustrated in Fig. 3, where we can see, on the left, an example of input with a collection of local attractors: 2 local attractors (a_1^1 and a_2^1) associated with the red BN, 3 local attractors (a_3^2, a_4^2 and a_5^2) associated with the green BN and 2 local attractors (a_6^3 and a_7^3) associated with the blue BN. On the right, an example of output from Step 2, formed by the following set of compatible attractor pairs: (a_2^1, a_3^2); (a_2^1, a_4^2); (a_4^2, a_6^3); (a_4^2, a_7^3); (a_5^2, a_6^3); (a_7^3, a_2^1); (a_7^3, a_1^1); (a_6^3, a_2^1); (a_6^3, a_1^1), which can be represented by a directed graph G'.

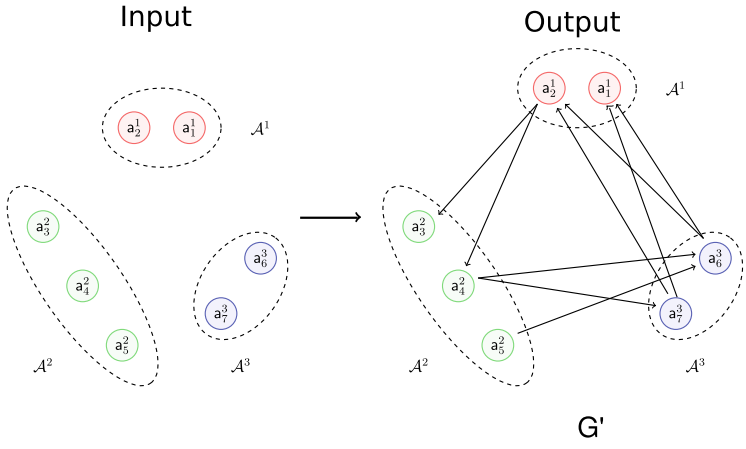

Fig. 3. An example of input and output of Step 2. (Color figure online)

3.3 Serial Step 3: Assembly of Stable Attractors Fields

The input to Step 3 is a collection of compatible attractor pairs produced in Step 2, which can be represented by a directed graph $G' = (V', E')$, as illustrated in Fig. 4, and a graph $G = (V, E)$ representing the CBN topology. Figure 4 illustrates, on the left, an example of input from Step 3 formed by the following set of compatible attractor pairs: (a_2^1, a_3^2); (a_2^1, a_4^2); (a_4^2, a_6^3); (a_4^2, a_7^3); (a_5^2, a_6^3); (a_7^3, a_2^1); (a_7^3, a_1^1); (a_6^3, a_2^1); (a_6^3, a_1^1), which can be represented by a directed graph G'. On the right, an example of the output from Step 3, which constitutes the solution to the problem: a set S of attractor fields, which are represented by directed graphs $G''_1, \ldots, G''_k$ (each graph represents an attractor field). A graph $G'' = (V'', E'') \in S$ is such that: i) vertices denote local attractors of BNs, and no two vertices in V'' ever have the same color, i.e., each of them is associated with a different BN; ii) edges in E'' denote pairs of compatible local attractors.

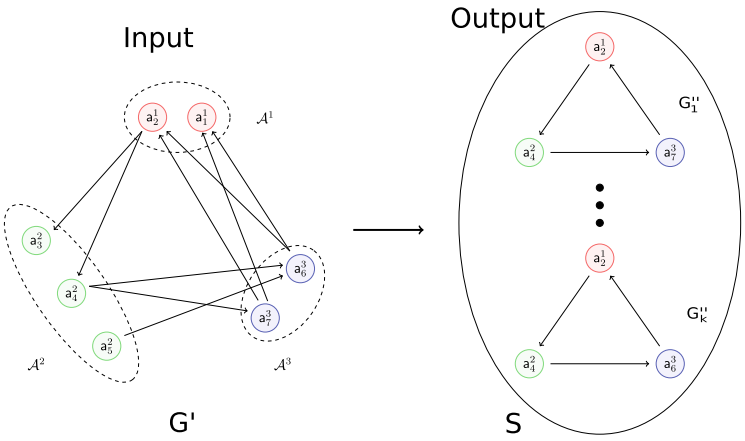

Fig. 4. An example of input and output of Step 3.

First, we label the edges in G' in such a way as to group them according to the pairs of BNs to which they are associated. Then, suppose a set of labels $R = \{1, 2, 3, \ldots, |E|\}$, and a labeling $label : E \rightarrow R$. Given the graph $G' = (V', E')$ that represents the pairs of compatible attractors, then for every edge $e' \in E'$ whose tips are attractors originating in BNs i and j, where (i, j) is an edge of the graph $G = (V, E)$, we do $e' \leftarrow label(i, j)$. Figure 5 illustrates this labeling, where we can see, in the top center, a graph $G = (V, E)$ representing the topology of an example CBN with labels 1, 2 and 3 on the edges. On the bottom left, an example of a graph $G' = (V', E')$ that represents the pairs of compatible attractors. On the right below, an example of labeling on the edges of E'.

In other words, if the CBN contains $|E|$ edges, then we have $|E|$ distinct labels, one for each edge in E, and this labeling classifies the edges of G' into $|E|$ groups, one for each edge in G. Therefore, for a CBN with a set of edges E, and a set of compatible attractor pairs $G' = (V', E')$, we will have an $|E|$-clustering *label* of E', which is a partition of E' into $|E|$ groups (clusters):

$$\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{|E|}\}.$$

A subgraph of G' is said to be a *stable subgraph* if it satisfies the following conditions: i) it has at least 1 edge, ii) it is connected and iii) it has no two edges belonging to the same group $\mathcal{G}_u$, $1 \leq u \leq |E|$. We define the *order* of a stable graph to be its number of edges. i.e., a stable subgraph of G' is of order k if it has k edges. Then, given a group $\mathcal{G}_u$, $1 \leq u \leq |E|$, the compatible pairs in $\mathcal{G}_u$ correspond to stable subgraphs of order 1, since an element of $\mathcal{G}_u$ is an edge of G'. A stable subgraph of order k will be represented by a k-tuple formed by k pairs of vertices of G' (local attractors).

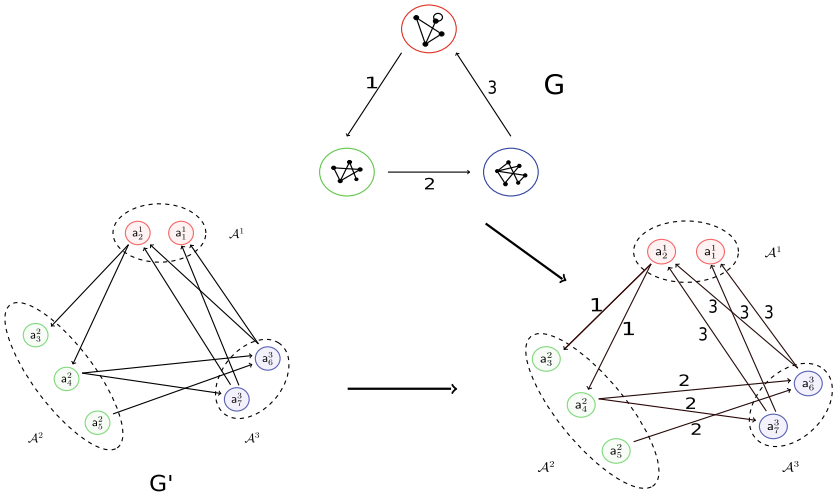

Fig. 5. An example of edge labeling in the graph G' (input of Step 3).

The general idea in Step 3 consists of an iterative procedure to compute stable subgraphs of G' of increasingly higher order until we reach one of order $|E|$, which can be described as follows:

- we choose a group $\mathcal{G}_u \in \mathcal{G}$, $1 \leq u \leq |E|$, with the highest cardinality, and we define a visiting order for the remaining groups starting from the group $\mathcal{G}_u$ performing a breadth-first traversal in the graph G weighted by the cardinality of the groups, which we denote by $\mathcal{G}'$:

$$\mathcal{G}' = [\mathcal{G}'_1, \mathcal{G}'_2, \ldots, \mathcal{G}'_{|E|}]$$

- note that each element of $\mathcal{G}'$ is a **set** of stable subgraph of order 1, i.e., each element of $\mathcal{G}'_u \in \mathcal{G}'$, $1 \leq u \leq |E|$, is a stable subgraph of order 1;
- each element $\mathcal{G}'_u \in \mathcal{G}'$, $1 \leq u \leq |E|$, is represented by a 1-tuple, i.e., $\mathcal{G}'_u$ is a set of 1-tuples;
- a stable subgraph of order k is represented by a k–tuple;
- we denote by S_k a set of k-tuples that represents a partial solution of order k (a set of stable subgraph of order k);
- $S_1 \leftarrow \mathcal{G}'_1$; $k \leftarrow 2$
- repeat until $k = |E|$:
  1. $S_k \leftarrow \emptyset$
  2. for all for $(a, b) \in S_{k-1} \times \mathcal{G}'_k$, do:
     - if (a, b) is a stable subgraph of order k, then $S_k \leftarrow S_k \cup (a, b)$
  3. $k \leftarrow k + 1$

Note that for a given iteration k, $k \geq 2$, a pair (a, b) is such that:

a corresponds to a $(k-1)$–tuple formed by $k-1$ pairs of local attractors, which represents a stable subgraph of order $k-1$;

b corresponds to a 1–tuple formed by 1 pair of local attractors, which represents a stable subgraph of order 1.

Then, to decide whether a given pair (a, b) is a stable subgraph of order k, just check if a and b have an element (local attractor) in common. In case it is true, (a, b) represents a graph with k edges that satisfies the conditions that define a stable subgraph mentioned above. An attractor field is a stable subgraph of order $|E|$, which implies that when we finish computing $S_{|E|}$, we have reached the solution of the problem, since each element of $S_{|E|}$ is a stable subgraph of order $|E|$.

4 Parallel Method

In this work we assume scenarios with few hardware resources, typically a single-chip multicore machine. Basically a board with p processing cores, where each core has its own local memory, but there is also a cache shared between the cores to speed up access to the most used data, improve communication between the cores and reduce read and write latency.

Our method is based on the strategy of Tovar's serial method, i.e., we preserve the three steps provided for in the serial method, but we execute in parallel the tasks provided for in each step, and we perform intra-step load balancing.

4.1 Multicore Step 1

Suppose that the input CBN has m BNs and that we have p processing cores available. In the ideal case, we would have a completely homogeneous CBN where the BNs would all be equal and all of them would have the same number of coupling signals. In this case, it would be enough to distribute the BNs equally among the processing cores, allocating m/p BNs to each core.

However, realistic inputs contain heterogeneous CBNs, where the BNs are different from each other and have different degrees of coupling. This implies that the workload associated with the computation of the local attractors varies greatly, so it is important to perform load balancing at this point.

We define a task in Step 1 as the computation of the local attractors of a given BN; therefore, for a CBN with m BNs, we have m tasks to perform.

The performance associated with a Step 1 task depends on 4 factors: i) the number of local variables of the BN, ii) the number of coupling signals incident on the BN, iii) the number and length of attractors in a BN and iv) the length of the longest path to an attractor.

To define an initial load balancing model in Step 1, we will use only the first two factors because the last two derive from information that we do not have *a priori* and from structural characteristics of the Dubrova and Teslenko algorithm, in which we do not interfere.

Thus, now we define the weight of workload $w1$ associated with a Step 1 task in BN i as being the integer defined as follows:

$$w1(BN_i) = (n - v) \times 2^v,$$

where n is the number of local variables of the BN i and v is the number of coupling signals incident on the BN i.

Therefore, the total weight $tw1$ of the workload associated with a CBN (in Step 1) is given by

$$tw1 = \sum_{i=1}^{m} w1(BN_i),$$

where m is the number of BNs.

We implement two strategies to distribute $tw1$ among the available cores, which we describe in the following two sections.

Multi-core Step 1 (version A). In this first implementation, we directly distribute the tasks among the available cores according to a demand-availability relationship, as follows:

- We create a task pool F containing all tasks in arbitrary order.
- We allocate the first p tasks to the available p cores.
- If a core becomes available, we allocate the first task in F to that core. We repeat this procedure until all tasks are allocated.

In this implementation of Step 1, we have no cost to pre-allocate tasks among cores. However, we do not guarantee approximately equal load balancing among cores.

Multi-core Step 1 (version B). To pre-partition $tw1$ among the cores approximately equally, we distribute the m tasks among the p cores using a greedy load balancing strategy, which can be described as follows:

- Sort tasks in descending order of weight ($w1(BN_i)$), so that the heaviest tasks are assigned first.
- Initialize a list `Load` with p elements, where each element represents the load (total weight) of each core. Initially, all cores have zero load.
- For each task, assign it to the core with the least load at that moment:
  1. for each task $w1(BN_i)$ (in the sorted order of weights), find the core k with the minimum load (i.e., the core with the smallest value in `Load`);
  2. assign the task $w1(BN_i)$ to core k, and update the load of core k by adding $w1(BN_i)$ to its load. For example, if the load of core k before assignment was `Load[k]`, the new load will be `Load[k] +=` $w1(BN_i)$.
- Repeat until all tasks are assigned, i.e., continue assigning tasks to the cores until all m tasks have been distributed. After all tasks are distributed, the `Load` list will contain the total load of each core.

Sorting takes $O(m \log m)$ time. Distributing the tasks: for each task, the algorithm finds the core with the least load, which can be done in $O(\log p)$ using a Heap structure. Therefore, the complexity of distributing all tasks is $O(m \log p)$. Thus, the overall complexity of Step 1 load balancing is $O(m \log m + m \log p)$.

4.2 Multi-core Step 2

Given the topology of a CBN, represented by a directed graph $G = (V, E)$, the input of Step 2 is a collection of sets of local attractors, being a set $\mathcal{A}^u$, $1 \leq u \leq |V|$, of local attractors for each BN, as illustrated in Fig. 3.

Given an edge $(i \to j) \in E$, we define a task in Step 2 as the computation of all compatible attractor pairs in $\mathcal{A}^i \times \mathcal{A}^j$, where $\mathcal{A}^i$ and $\mathcal{A}^j$ denote the local attractor sets of the BNs i and j, respectively.

The performance associated with Step 2 is determined by the number of pairs of local attractors whose compatibility must be checked. In other words, the computation cost of Step 2 depends on the cardinality of the sets of attractors where the compatible pairs are prospected.

Thus, given an edge $(i \to j) \in E$, we define the weight of workload $w2$ in Step 2 associated with the edge $(i \to j)$ as being the cardinality of the Cartesian product $\mathcal{A}^i \times \mathcal{A}^j$, i.e.,

$$w2(i, j) = |\mathcal{A}^i \times \mathcal{A}^j|.$$

Therefore, the total weight $tw2$ of the workload associated with a CBN (in Step 2) is given by

$$tw2 = \sum_{i,j \in E} w2(i, j).$$

Similar to Step 1, here we also implement two strategies to distribute $tw2$ among the available cores, which we describe in the following two sections.

Multi-core Step 2 (version A). In this first implementation of Step 2 we use exactly the same strategy used in version A of the implementation of Step 1 (Sect. 4.1). Then, here we also have no pre-allocation cost, but we do not guarantee an equal distribution of the load among the cores.

Multi-core Step 2 (version B). To divide $tw2$ among the cores approximately equally, we distribute the $|E|$ tasks among the p cores using exactly the same strategy used in version B of the implementation of Step 1 (Sect. 4.1).

Therefore, ordering tasks takes $O(|E| \log |E|)$ time and the complexity of distributing all tasks is $O(|E| \log p)$, in such a way that the overall complexity of Step 2 load balancing is $O(|E| \log |E| + |E| \log p)$.

4.3 Multi-core Step 3

As we saw in Sect. 3.3, in the serial method for a given iteration k, we have to check all pairs in $S_{k-1} \times \mathcal{G}'_k$, where S_{k-1} and $\mathcal{G}'_k$ denote sets of tuples.

Given a pair $(a, b) \in S_{k-1} \times \mathcal{G}'_k$, we define a task of Step 3 as checking whether or not (a, b) forms a stable subgraph of order k.

Similar to Steps 1 and 2, here we also implement two task allocation strategies, which are described in the following subsections.

Multi-core Step 3 (version A). Here we dynamically reallocate tasks at each iteration according to a demand-availability relationship, as follows:

- Suppose an iteration k and p cores available. Let a_i be the i-th tuple of S_{k-1} (consider an arbitrary order).
- First we allocate the first p tasks on the available p cores, as follows:
  1. We allocate all pairs in $a_1 \times \mathcal{G}'_k$ to the 1st available core, which will process the corresponding tasks.
  2. We allocate all pairs in $a_2 \times \mathcal{G}'_k$ to the 2nd available core, which will process the corresponding tasks.
  3. And so on until all cores are busy.
- Then we allocate the remaining tasks as cores become available, according to the following:
  1. The pairs in $a_{p+1} \times \mathcal{G}'_k$ are allocated to the first core that becomes available again.
  2. The pairs in $a_{p+2} \times \mathcal{G}'_k$ are allocated to the first core that becomes available again.
  3. And so on until the pairs in $a_u \times \mathcal{G}'_k$ are allocated to some core, where a_u denotes the last tuple in S_{k-1}.

Multi-core Step 3 (version B). Suppose an iteration k. If the number of available cores is p, we can divide the set S_{k-1} into p roughly equal parts and distribute those parts to different cores. That is, each core i will process a portion of the set S_{k-1}, as follows:

- The core i receives $\mathcal{G}'_k$ and a segment S^i_{k-1} of S_{k-1}, which is a subset $S^i_{k-1} \subseteq S_{k-1}$, with $|S_{k-1}|/p$ elements (See Fig. 6).
- The core i generates all the pairs (a^i, b), where $a^i \in S^i_{k-1}$ and $b \in \mathcal{G}'_k$, and checks whether each of them forms a stable subgraph of order k. This operation is performed sequentially within each core since each pair is generated independently of other pairs.

Since the stable subgraphs of order k generated by each core i can be stored in a local data structure, synchronization can be performed after the execution of all cores. At the end of the execution, the cores can combine (or reduce) the results into a single shared data structure, which represents the partial solution S_k (a set of stable subgraphs of order k). The diagram in Fig. 6 illustrates how the procedure works for a given iteration k.

Unlike the previous Steps, in this version of Step 3 we perform dynamic load balancing, in such a way that at each iteration k we redistribute the $|S_{k-1}| \times |\mathcal{G}'_k|$ tasks among the available cores.

At each iteration k, we perform 1 round of computation and 2 rounds of communication: one to allocate tasks to cores and another one to write to shared memory, where the results of each core are combined into a shared data structure, which represents a partial solution. For a CBN with $|E|$ edges, we perform $|E|-1$ iterations. Therefore, in Step 3 we perform $O(|E|)$ rounds of communication and computation.

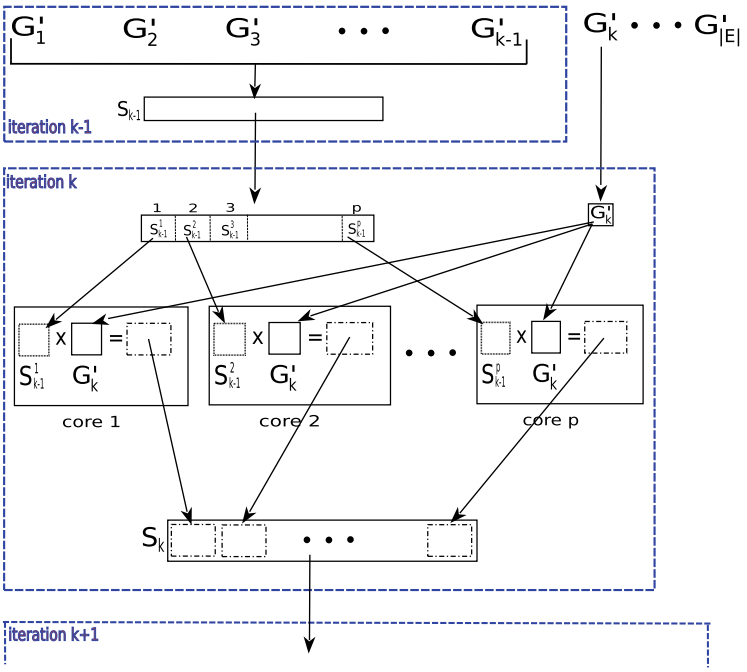

Fig. 6. Flow in an iteration k: the partial solution S_{k-1} is divided among the p available cores; each core receives a copy of $\mathcal{G}'_k$; each core computes the stable subgraphs of order k in $S_{k-1} \times \mathcal{G}'_k$; the results obtained in each core are combined to form the partial solution S_k, which will be similarly used in the iteration $k+1$.

5 Implementation and Experimental Results

The method was implemented in Python 3.8.10 and the experiments were performed using Ubuntu 20.04 on a desktop with Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz, 128 GB RAM and 24 cores.

Under this setup, we performed preliminary experiments to evaluate the performance and scalability of the method considering the scenario where there is an increase in the number of BNs. In all experiments, we adopted a random topology for the CBNs. Steps 1, 2 and 3 of the Serial method are relatively independent, hence we run and evaluate experiments (considering the two parallel versions) for these three steps independently. More details can be seen in the following subsections.

5.1 Speedup

To evaluate the speedup of the parallel versions with regard to the Serial implementation, we increased the number of BNs (from 3 to 1024) and fixed the number of local variables in each BN at 5. We kept the total number of coupling signals proportionally constant with regard to the number of BNs, i.e., we

adopted the number of coupling signals equal to the number of BNs. In this implementation of the Serial method 24 cores were used. In these experiments 1000 instances were generated for each adopted sample. The average execution time obtained for each sample is presented in Table 1.

Table 1. Comparison of execution times, grouped by stage (step) and method, with an increasing number of Boolean networks.

	Experiment 1			**Experiment 2**			**Experiment 3**		
	Step 1			**Step 2**			**Step 3**		
Method	Serial	Parallel A	Parallel B	Serial	Parallel A	Parallel B	Serial	Parallel A	Parallel B
# **BNs**									
3	0.2315s	0.4817s	0.4833s	0.0002s	0.3669s	0.3537s	0.0024s	0.6810s	0.6602s
4	0.3168s	0.5041s	0.5041s	0.0003s	0.3718s	0.3595s	0.0088s	1.0362s	1.0191s
5	0.4034s	0.5129s	0.5131s	0.0004s	0.3727s	0.3591s	0.0534s	1.4150s	1.3976s
6	0.4985s	0.5250s	0.5231s	0.0004s	0.3770s	0.3636s	0.4293s	1.9140s	1.8839s
7	0.5884s	0.5262s	0.5289s	0.0005s	0.3767s	0.3619s	2.5732s	3.0863s	3.1246s
8	0.6738s	0.5211s	0.5257s	0.0006s	0.3795s	0.3675s	23.2837s	11.8042s	12.1993s
16	1.4994s	0.3824s	0.3821s	0.0011s	0.2618s	0.2698s	—	—	—
32	2.8987s	0.4614s	0.4264s	0.0026s	0.4221s	0.4322s	—	—	—
64	5.9565s	0.6593s	0.6168s	0.0075s	1.0274s	1.1707s	—	—	—
128	12.1228s	1.0541s	1.0223s	0.0279s	2.0236s	2.2752s	—	—	—
256	24.5006s	1.7848s	1.8498s	0.0991s	4.1358s	4.6605s	—	—	—
512	49.1510s	3.2944s	3.5305s	0.3754s	9.2657s	9.4604s	—	—	—
1024	108.4408s	7.9494s	8.4293s	1.5056s	21.6097s	21.6038s	—	—	—

Step 1: As we can see in Table 1 (Experiment 1, with red background), for a small number of BNs (up to 6 BNs) the Serial implementation is faster than both parallel implementations. However, starting from 8 BNs, this trend reverses, and the parallel versions outperform the serial implementation. For larger inputs (512 and 1024 BNs) the parallel implementations are vastly superior (speedup between 13 and 14). These data suggest that the greater the number of BNs, the more advantageous the parallel implementations become. The data show that there are no significant performance differences between the two parallel versions of Step 1, which indicates that the best implementation alternative for Step 1 is parallel version A, since it is a simpler strategy compared to version B.

Step 2: As we can see in Table 1 (Experiment 2, with green background), the Serial implementation is faster than both parallel versions A and B, even for inputs with many BNs. This superiority of the Serial implementation derives from the simplicity of the operations in Step 2, which means that developing parallel versions for Step 2 adds complexity to the code without bringing performance gains. In fact, the runtime of the Serial implementation is negligible even for large number of BNs. Thus the best path is to adopt a fork-join model, where we run a parallel version for Step 1 and a serial version for Step 2.

Step 3: As we can see in Table 1 (Experiment 3, with blue background), the Step 3 represents a bottleneck of the method, since on the adopted machine, representative of the scenarios covered by this work where there is little availability of hardware resources, we were able to run CBNs with only up to 8 BNs. This limitation occurred in both the serial and parallel implementations and it derives from a structural characteristic of the dynamics of CBNs, which is the fact that some CBNs contain a huge amount of AFs (in the order of hundreds of millions). These cases occur for some CBNs with 9 or more BNs, which require a substantial memory consumption, often causing memory overflow. Since the runtimes presented in Table 1 represent average values (calculated from samples with 1000 BNs), the average runtimes for experiments with 16 or more BNs are not provided, as at least one case resulted in a memory overflow. Despite this limitation, observing the data for inputs with 3 to 8 BNs, we can see that in the case with 8 BNs the parallel implementations are faster than the serial implementation, which suggests that from this point on it is advantageous to adopt a parallel version. Clearly improvements to the method are needed, specifically a memory management strategy that addresses this memory overflow problem, enabling further experiments to validate these results.

5.2 Scalability

To evaluate the scalability of Step 1, we assume CBNs with 1024 BNs, 1024 coupling signals and 5 variables per BN. Then we execute its parallel implementations using $2, 4, 8, 12, 16, 20$ and 24 cores. In this experiment we generated a sample with 250 instances, then the runtimes presented represent the average of the runtimes obtained, which can be viewed in Fig. 7. We focus on the scalability analysis only for Step 1 because: i) as already concluded in the previous section, it is better to execute Step 2 in serial mode in a fork-join model due to its extreme simplicity, hence there is no reason to evaluate the scalability of Step 2; ii) regarding Step 3, to maintain coherence with the input set used in Step 1, we should evaluate Step 3 using 1024 BNs (as done in Step 1), however this is not possible as discussed in Sect. 5.1. Hence for now we cannot evaluate the scalability of Step 3.

As we can see in Fig. 7, parallel implementations A and B have very similar performance, varying only marginally from each other. And in both implementations we can observe that the speedup grows almost linearly up to 12 cores. Starting from 16 cores, the speedup becomes smoother and stabilizes around 10. We attribute this saturation due to three combined factors: the growth rate of the number of cores includes 12, 20 and 24, which are not a power of 2 (as adopted for the number of CBNs), communication overhead that increases as the number of cores increases, and mainly the use of virtual cores (the machine adopted to run the experiments has only 12 physical cores).

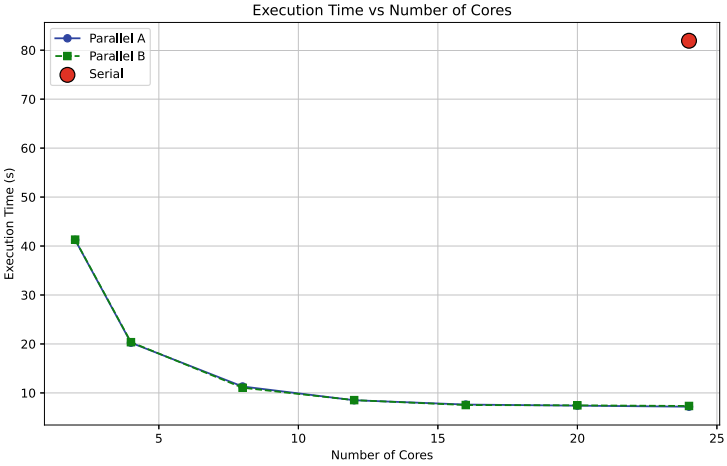

Fig. 7. Runtimes versus number of cores (2, 4, 8, 12, 16, 20, 24).

6 Conclusion

In this paper, we present parallel solutions based on the multi-core architecture for Tovar's method for AFs computation, assuming scenarios with few hardware resources. We develop different implementations for each of the three steps of the serial method, including intra-step load balancing.

We performed and evaluated experiments for each of the three steps of the method with the aim of indicating which "strategy-step" combination leads to the best overall performance. The data allow us to conclude that the best way to optimize the Tovar's method is to adopt a fork-join model where we run parallel (Steps 1 and 3) and serial (Step 2) code snippets.

As future work, a memory management strategy is required for Step 3 to enable the method to operate with hundreds of millions of AFs, which will allow the method to be applied to more realistic input sizes. In addition, the implementation of a fork-joint model is needed to optimize the code with serial (Step 2) and parallel (Steps 1 and 3) snippets. Finally, intensive experiments involving inputs with thousands of BNs and in different scenarios, such as increasing the number of variables per BN and the number of coupling signals, are required to further evaluate the efficiency of the method.

Acknowledgments. This research was supported by Coordenaçao de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) Finance Code 001 and São Paulo Research Foundation (FAPESP) grants #2018/18560-6 and #2018/21934-5.

References

1. Arenas, A., Díaz-Guilera, A., Kurths, J., Moreno, Y., Zhou, C.: Synchronization in complex networks. Phys. Rep. **469**, 93–153 (2008)
2. Chen, H., Liang, J., Lu, J.: Partial synchronization of interconnected Boolean networks. IEEE Trans. Cybern. **47**(1), 258–266 (2017)
3. Chen, H., Liang, J.: Local synchronization of interconnected Boolean networks with stochastic disturbances. IEEE Trans. Neural Netw. Learn. Syst. **31**(2), 452–463 (2019)
4. Cheng, D., Qi, H., Li, Z.: Analysis and control of Boolean networks: a semi-tensor product approach. Springer Science & Business Media (2010)
5. Cohen, M., Georgiou, M., Stevenson, N.L., Miodownik, M., Baum, B.: Dynamic filopodia transmit intermittent delta-notch signaling to drive pattern refinement during lateral inhibition. Dev. Cell **19**(1), 78–89 (2010)
6. Dubrova, E., Teslenko, M.: A sat-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Trans. Comput. Biol. Bioinf. **8**(5), 1393–1399 (2011)
7. Jalan, S., Amritkar, R.E.: Self-organized and driven phase synchronization in coupled maps. Phys. Rev. Lett. **90**, 014101 (2003)
8. Kauffman, S.A.: The Origins of Order. Oxford University Press, New York (1993)
9. Li, F.: Feedback control design for the complete synchronisation of two coupled Boolean networks. Int. J. Syst. Sci. **47**(12), 2996–3003 (2016)
10. Li, R., Chu, T.: Synchronization in an array of coupled Boolean networks. Phys. Lett. A **376**, 3071–3075 (2012)
11. Liu, Y., Sun, L., Lu, J., Liang, J.: Feedback controller design for the synchronization of Boolean control networks. IEEE Trans. Neural Netw. Learn. Syst. **27**(9), 1991–1996 (2015)
12. McCorkindale, A.L., et al.: A gene expression atlas of embryonic neurogenesis in drosophila reveals complex spatiotemporal regulation of lncRNAs. Development **146**(6), dev175265 (2019)
13. Michki, N.: The Developmental Landscape of Neurogenesis in Drosophila melanogaster Revealed Using Targeted Single-Cell RNA-Sequencing and a Multi-Informatic Analysis Paradigm. Ph.D. thesis, University of Michigan (2021)
14. Niu, Y., Liu, H., Wei, Q.: Synchronization of coupled Boolean networks with different update scheme. IEEE Access **8**, 79319–79324 (2020)
15. Pecora, L.M., Carroll, T.L.: Master stability functions for synchronized coupled systems. Phys. Rev. Lett. **80**, 2109 (1998)
16. Pinot, M., Le Borgne, R.: Spatio-temporal regulation of notch activation in asymmetrically dividing sensory organ precursor cells in drosophila melanogaster epithelium. Cells **13**(13), 1133 (2024)
17. Qin, M., Li, H., Yang, X.: Attractor analysis of conjunctive Boolean networks via dependency graph. In: 2024 43rd Chinese Control Conference (CCC), pp. 5345–5350. IEEE (2024)
18. Tovar, C.R., Martins, D.C., Rozante, L.C., Araujo, E.: A method for computing attractor fields in coupled Boolean networks. In: 2022 IEEE 22nd International Conference on Bioinformatics and Bioengineering (BIBE), pp. 315–320. IEEE (2022)

19. Wu, Y., Lu, R., Shi, P., Su, H., Wu, Z.G.: Adaptive output synchronization of heterogeneous network with an uncertain leader. Automatica **76**, 183–192 (2017)
20. Yang, Z., Chu, T.: General synchronization of cascaded Boolean networks within different domains of attraction. In: Proceedings of the International Conference of Control, Dynamic Systems, and Robotics. Ottawa, Ontario, Canada. vol. 173 (2015)