Chaos

Fast algorithm to identify minimal patterns of synchrony through fibration symmetries in large directed networks

Cite as: Chaos **32**, 033120 (2022); https://doi.org/10.1063/5.0066741
Submitted: 12 August 2021 • Accepted: 24 February 2022 • Published Online: 17 March 2022

Higor S. Monteiro, Ian Leifer, Saulo D. S. Reis, et al.

Fast algorithm to identify minimal patterns of synchrony through fibration symmetries in large directed networks

Cite as: Chaos **32**, 033120 (2022); doi: 10.1063/5.0066741
Submitted: 12 August 2021 · Accepted: 24 February 2022 ·
Published Online: 17 March 2022

Higor S. Monteiro,[1] Ian Leifer,[2] Saulo D. S. Reis,[1,a)] José S. Andrade, Jr.,[1] and Hernan A. Makse[2]

AFFILIATIONS

[1]Departamento de Física, Universidade Federal do Ceará, Fortaleza, Ceará 60451-970, Brazil
[2]Levich Institute and Physics Department, The City College of New York, New York, New York 10031, USA

[a)]**Author to whom correspondence should be addressed:** saulo@fisica.ufc.br

ABSTRACT

Recent studies have revealed the interplay between the structure of network circuits with fibration symmetries and the functionality of biological networks within which they have been identified. The presence of these symmetries in complex networks predicts the phenomenon of cluster synchronization, which produces patterns of a synchronized group of nodes. Here, we present a fast, and memory efficient, algorithm to identify fibration symmetries in networks. The algorithm is particularly suitable for large networks since it has a runtime of complexity $\mathcal{O}(M \log N)$ and requires $\mathcal{O}(M + N)$ of memory resources, where N and M are the number of nodes and edges in the network, respectively. The algorithm is a modification of the so-called refinement paradigm to identify circuits that are symmetrical to information flow (i.e., fibers) by finding the coarsest refinement partition over the network. Finally, we show that the algorithm provides an optimal procedure for identifying fibers, overcoming current approaches used in the literature.

Network fibers are circuits with fibration symmetries. These symmetries imply that nodes sharing a fiber receive identical information from the rest of the network, leading to the synchronization of their dynamics. Since these symmetries offer a conceptual approach to identify functional building blocks in networks, in contrast to network motifs,[1] it is desirable to develop an efficient algorithm capable of extracting these circuits for large networks. Different methods have been proposed for the identification of fibers. In particular, a balanced coloring algorithm has been used in the recent work of Morone *et al.*[2] to show the ubiquitousness of these symmetries across several real networks. However, even though these methods are intuitive and relatively simple to implement, they can be highly inefficient regarding either their time or space complexity, limiting their applications to small networks. In this work, we show that refinement algorithms represent a natural approach to identify fibration symmetries, allowing us to build an efficient method to find the minimal components composed by fibers and their external regulators, called fibration building blocks,[2] in very large sparse networks.

I. INTRODUCTION

Progress on network science in the last two decades has produced insightful frameworks for theoretical and practical studies of dynamical systems on networks.[3–5] Dynamical processes constrained by network structures can display novel qualitative behaviors, which are not directly observed in classical dynamical systems.[6] This is especially true for information-processing systems, in which the phenomenon of synchronization[7] is directly related to functionality.

Recently, Morone and collaborators[2] used the formalism of fibration symmetries on graphs[8] to characterize sets of synchronized nodes on complex networks. These sets, known as fibers, can provide vital information about the structure-function relation in a network, particularly in biology. Fibers provide topological conditions to be satisfied to induce synchronization in a subset of nodes.[9] Also, the breaking of the fibration symmetry in the network structure creates subnetworks that can behave as logical computational circuits.[10] From these novel structures, it is possible to define states of synchrony by considering only the topological features of the given network.[11]

The introduction of fibration goes back to the work of Grothendieck in the field of algebraic geometry.[12] Boldi and Vigna[8] adapted this concept to graphs by calling it "graph fibrations." Let G be a graph with fibers $f_1, f_2, \ldots, f_b$, B be a graph with b nodes and ψ be a homomorphism $\psi : G \to B$. ψ is a symmetry fibration[2,10,13] if it "collapses" each fiber in G to the unique node in B while maintaining the connectivity.[2,8] The formal name of the symmetry fibration is surjective minimal graph fibration[2] and collapses the graph into the minimal number of fibers. Fibers are known in the field of dynamical systems by the name balanced coloring[6,11,14] and in computer science by the name balanced equivalence relations.[8,15] The equivalence between these terms is used in this work to propose an efficient algorithm to identify fibers. We will discuss the relation between balanced colorings and fibers in more detail below.

Compared to automorphisms, fibration symmetries are less restrictive regarding perturbations to the structure of the network. This comes from the notion that a graph fibration is a transformation that preserves only the input trees of the nodes of the network, instead of the inputs and outputs of each node preserved by automorphisms that define symmetry groups. For a definition of input tree, we need to first define the concept of input set of a node $v \in V$ of the network $G(V, E)$. An input set of a node v is the set of pairs $(\text{type}(e_G), w)$ such that $s(e_G) = w$ and $t(e_G) = v$, where $s(e_G)$ and $t(e_G)$ are, respectively, the source and the target of the edge $e_G \in E$, and $w \in V$. If the network is a multiplex, meaning that there is more than one type of edge, the $\text{type}(e_G)$ is necessary information to define correctly the input set of v. Otherwise, if the network contains only one type of edge, the input set of v reduces to all its incoming neighbors w. Following this definition, we can define recursively an input tree by considering input sets of input sets as many times as possible. This recursive procedure is infinite if there are cycles in the network.[2]

Intuitively, to fully characterize the information flow paths in a directed network, we can define for each node v an input tree containing all the information pathways that terminate on v. Thus, if two different nodes have isomorphic input trees, indicating that they receive equivalent information from the network, they synchronize their dynamical states, exhibiting correlated dynamics in the network. If the network is composed by different groups of synchronized nodes, we say that there is a cluster synchronization of the dynamics on this network.[11] If a group of nodes in a network has isomorphic input trees, they belong to the same fiber[2,8] and can be collapsed by the fibration into one single node of a representative base network B. In this way, a graph fibration reduces the original network to a base network, where each node in B represents a fiber module of G in which all nodes inside this module have isomorphic input trees. Examples of directed networks formed by three nodes are shown in Fig. 1. Here, we show that these different patterns of connections are responsible for the different cluster synchronization observed in the networks.

As shown in Ref. 16 and from the groupoids theory developed in Refs. 6 and 17 for any directed network, it is possible to define an equivalence relation $\bowtie$ on the set of nodes V that partition the network into equivalence classes. Moreover, an equivalence relation on V is said balanced if, for any $u, v \in V$ with $u \bowtie v$, there exists an input isomorphism β such that $s(e) \bowtie s(\beta(e))$ for every input edge e of u. $\beta(e)$ is an input edge of v, and s is the map associating an edge to its source node.[6,16] This way, when the number of equivalence classes obtained for $\bowtie$ are balanced and minimal, with minimal referring to the minimal number of classes, each class corresponds exactly to the fibers of the network obtained through the symmetry fibration over G.[10,18] To this partitioning, we denote it as the minimal balanced coloring, where each class is associated with a different color, representing distinct fibers.

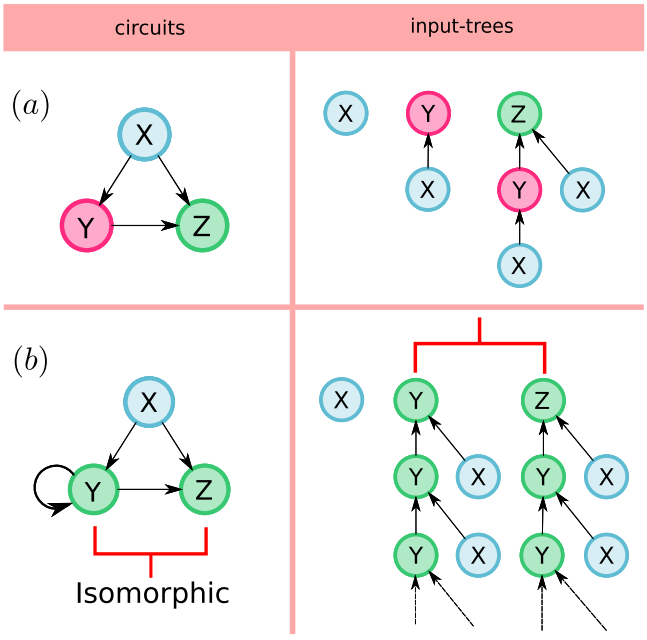


FIG. 1. Example of non-symmetric and symmetric circuits. In (a), the input tree of each node differs from the others, showing non-equivalence on the information flow received by each node. This circuit is a feed-forward loop motif,[1] which has no symmetry and no cluster synchronization. As shown in (b), by adding a simple self-loop to node Y, its input tree becomes isomorphic to the input tree of Z. This circuit is called a feed-forward fiber.[2] From this, their incoming information flows become equivalent, allowing both nodes to synchronize in their dynamics.

Equivalence classes of balanced equivalence relations are defined considering only the network structure and are associated with the network dynamics through the set of admissible ODEs of the network. Admissible ODEs are defined as the most general set of differential equations that respect the structure of the network.[6,17] Every node i has a corresponding dynamical variable $x_i(t)$ belonging to a given phase space P_i. The total phase space P of a network of size N is the product of the individual phase spaces of the dynamics of each node, $P = \prod_{1,\ldots,N} P_{x_i}$. The balanced equivalence relation $\bowtie$ defines the decomposition of the total phase space in the polydiagonal subspaces $\Delta_{\bowtie}$ corresponding to each equivalence class as $\Delta_{\bowtie} = \{x \in P \mid i \bowtie j \Rightarrow x_i = x_j\}$, where nodes i and j belong to the same class.[6] Following Ref. 14, we call these subspaces synchrony subspaces because they are flow-invariant for every admissible ODE of the network. That is, once the system occupies a dynamical state in this polysynchronous subspace, it stays in this subspace.

Therefore, if G has N nodes and has a total of b fibers, we have that the underlying dynamics of G given by its admissible ODEs has a synchronous solution according to its fibration partitioning (or minimal balanced coloring) given by its fibers (balanced classes) $f_1, \ldots, f_b$, meaning

$$\begin{aligned} x_1^1(t) &= x_2^1(t) = \cdots = x_{|f_1|}^1(t), \\ x_1^2(t) &= x_2^2(t) = \cdots = x_{|f_2|}^2(t), \\ &\vdots \\ x_1^b(t) &= x_2^b(t) = \cdots = x_{|f_b|}^b(t). \end{aligned} \tag{1}$$

This notion can be applied to the example of Fig. 1: the admissible ODEs for both networks (a) and (b) are given as

$$\begin{aligned} \dot{x} &= f(x), \\ \dot{y} &= g(y; x, y), \\ \dot{z} &= h(z; x, y), \end{aligned} \tag{2}$$

where the dependence of the first variable of the functions refers to the internal dynamics of each node. However, for Fig. 1(b), we have that node Y has isomorphic input tree with the input tree of Z, for which in this case the dynamics of y and z is modeled by the same function $h = g$ as they belong to the same balanced class of the minimal balanced coloring and, thus, belong to the same fiber of isomorphic input trees. Moreover, although the fibration symmetries guarantee the existence of a synchronous solution as shown in Eq. (1), they do not necessarily guarantee that these solutions are stable, this condition being required to satisfy the existence of cluster synchronization over the network. Throughout the rest of this work, we will use the terms "minimal balanced coloring" and "fibration partitioning" interchangeably.

In practice, methods to capture the correct fibration partitioning over directed networks are, in general, based on what is called a negative iterative strategy.[19,20] In this approach, an adequate initial partition is refined in each iteration with the goal to derive a better partitioning according to a predefined rule. Thus, each group of nodes in each iterative step is split into several, or none, smaller groups with respect to the whole current partition. This strategy defines the refinement paradigm.[19,21] This refinement procedure is also used to find the balanced equivalence relations[16,22,23] of directed networks, also known as the balanced colorings mentioned above. An alternative definition of a balanced coloring, now following an algorithmic approach, can be given as: define $\deg(u, C)$ as the degree of node $u \in V$ with color C, i.e., the number of edges of color C that points to u, then a given coloring of a graph G is said balanced if for any two nodes $v, w \in V$ of the same color the relation $deg(u, C_i) = \deg(v, C_i)$ holds for any $i \in [1, b]$, where b is the number of colors. Thus, it is possible to find different configurations of balanced colorings with different numbers of colors b. However, finding all possible balanced colorings is computationally expensive even for small networks, as shown in Ref. 16. Nonetheless, from a biological standpoint,[2,10] we are mainly interested in the balanced coloring with the minimal number of colors over real biological networks, as obtained through a minimal graph fibration since it allows us to identify the minimal number of components corresponding to the fundamental building blocks of the dynamics of these real networks. This coloring, as defined in Refs. 6 and 2 regarding the study of robust synchronization patterns, is an equivalence relation between the input sets of the nodes, and it is defined by the input tree isomorphisms representative of the fibration symmetry.[2,8,24]

A few different algorithms have been proposed to obtain the minimal balanced coloring over directed networks. In this work, we briefly describe the algorithms in Refs. 22 and 24 corresponding to Aldis's algorithm and the Boldi–Vigna algorithm, respectively. We also present a more detailed description of the algorithm presented by Kamei and Cock as an extension of the algorithm proposed by Belykh and Hasler in Ref. 23 since this method is the one used in Morone *et al.*[2] to uncover the fibration partitioning over genetic networks and other real networks. Although these algorithms are intuitive and easy to implement, they can be inefficient both in the time and memory complexities, meaning that they can be applied only for small to medium networks. Fortunately, because of the principled approach provided by the refinement paradigm and its relation with balanced colorings, we can make use of classical methods,[20,21,25,26] originally not designed for the identification of fibers, to design very efficient algorithms that can be applied to obtain the minimal balanced coloring for large sparse networks.

Here, we extend classical approaches to develop an efficient algorithm capable of identifying the minimal balanced coloring for general directed networks that also outperforms the alternative algorithms mentioned previously. The algorithm built here is a modified version of the algorithm presented by Paige and Tarjan,[26] with runtime complexity of $\mathcal{O}(M \log N)$ and space complexity of $\mathcal{O}(M + N)$, where M and N are the number of edges and nodes in the network, respectively. This algorithm has the same runtime order than the algorithm presented by Cardon and Crochemore.[27] However, compared with the algorithm of Cardon and Crochemore, the Paige–Tarjan (PT) algorithm has a simpler implementation and smaller prefactors, exhibiting a better performance when applied to the networks studied here.

We note that both algorithms were not originally designed for the identification of fibers. Nevertheless, by introducing simple modifications, we use the algorithm of Paige and Tarjan as a foundation of our method. We denote this method as Fast Fibration Partitioning (FFP) to contrast with the PT algorithm since here we apply the method specifically for the identification of fibers. We observe that this method outperforms for sparse networks the $\mathcal{O}(N^2 \log N)$ time complexity and the $\mathcal{O}(N^2)$ memory complexity of the Kamei–Cock (KC) algorithm used in Ref. 2 providing a more efficient approach for further studies on cluster synchronization on complex networks.

This paper is organized as follows. In Sec. II, we describe the balanced coloring algorithm used by Morone *et al.*[2] to contextualize the idea of refinement on fibration applications. In Sec. III, we explain how the general refinement paradigm can be modified to define the relationship between the input tree isomorphism and the algorithm that we employ here. In Sec. IV, we give the details concerning the proper initialization and implementation of the algorithm. In Sec. V, we list the mechanisms of two additional approaches used for fiber identification. In Sec. VI, we discuss all the algorithms introduced in order to compare their complexity. Through computational experiments on random networks, we also

compare the runtime of the main current method for fibration and the method described in this work. Finally, in Sec. VII, we make our conclusions.

II. REFINEMENT FOR FIBRATIONS

The refinement paradigm has been used widely in problems that originated from computer science and discrete mathematics.[21,26] It is also one of the major techniques used to develop efficient algorithms to solve the modular decomposition problem in graphs.[25] From state minimization in deterministic automata models[28] to the ordering of Boolean matrices and the lexicographically breadth-first search ordering of graphs,[21,26,29] the refinement procedure has been used to calculate the so-called balanced equivalence classes over a given set V.[16,26] Beyond its applications, the concept of balanced equivalence classes has also shown significance because it is directly related to the patterns of dynamical processes occurring on complex networks.[30] These equivalence relations are constructed through successive refinement steps according to a disjoint set rule, which should guarantee that the final partition, obtained by a refinement algorithm, is a set of pairwise disjoint sets representing the balanced classes.

A. The Belykh–Hasler algorithm

In Morone *et al.*,[2] the authors used a balanced coloring algorithm to find the correct fibration partitioning for several information networks. The original algorithm was first presented by Belykh and Hasler[23] and, even though they refer to the algorithm as a coloring algorithm, it differs from the classical coloring problems from graph theory. In the case of balanced coloring algorithms, adjacent nodes can have the same color and still be a proper coloring for the network. Precisely, in the Belykh–Hasler algorithm, every node is identified by a unique color representing its class, where nodes having the same color belong to the same class of isomorphic input set nodes, meaning that these nodes are equivalent. Since each class does not overlap with any other, the balanced coloring obtained by the Belykh–Hasler algorithm represents a proper partitioning over the set of nodes V of the network $G(V, E)$.

For the description of the algorithm used in Ref. 2, we define a coloring over the network as balanced if for every pair of nodes $v, w \in V$ belonging to the same class exists an isomorphism between their input sets, that is, there is a one-to-one relation between each element of the input set of v with each element of the input set of w. If for every pair of nodes having the same color, they receive equivalent information via its incoming links, then the coloring is balanced and each color represents a label for one fiber of the network. For instance, in Fig. 1(a), all nodes have different input trees, while nodes Y and Z have isomorphic input trees in Fig. 1(b). Since there is a relation one-to-one between their input trees, both Y and Z synchronize their dynamical states. Thus, equivalent classes of a balanced coloring represent the proper fibration partitioning over a given network, implying that finding the minimal balanced coloring is equivalent to finding fibers over a minimal fibration.

Regarding the refinement paradigm, the Belykh–Hasler algorithm is based on the idea of choosing an initial coloring and refining this coloring until further refinement becomes impossible. Initial coloring or partitioning places all nodes inside the same equivalence class, and each iteration introduces an input driven refinement.

To define an input driven refinement we introduce the input set color vector (ISCV).[10] The input set color vector of a node $v \in V$ is a K-dimensional vector, where K is the number of colors in the network and the jth entry of this vector counts how many nodes of color j belongs to the input set of v. A coloring is then balanced if, and only if, nodes of the same color have the same ISCVs. Moreover, a coloring is balanced and minimal if, and only if, nodes of the same color have the same ISCVs and nodes of different colors have different ISCVs.

In the algorithm, all nodes are initially assigned with the same color, representing the same equivalence class. Then, ISCVs for all nodes are calculated. Refined coloring needs to be balanced; therefore, nodes of the same color need to have the same ISCVs. Refinement step is done by assigning all unique ISCVs with different colors and assigning refined colors to the nodes according to their ISCV. That is, the refined color of the node is defined by the color assigned to the ISCV of this node in the set of the unique ISCVs. If the coloring after the iteration is equivalent to the coloring before the iteration, the algorithm stops and we obtain a balanced coloring. An example of how the algorithm works is shown in Fig. 2(a).

B. Balanced coloring for multiple edge types

The approach presented in Sec. II A was extended in 2013 by Kamei and Cock[16] to be applied for directed graphs with multiple types of edges. This algorithm can be applied to a genetic network with activator and repressor edges, for instance. In this case, instead of considering only the color of the node in the input set both node's color and type of the edge are considered. All the steps taken by the algorithm remain the same, but the size of a ISCV is now equal to the number of colors in the graph multiplied by the number of edge types. For instance, let us consider the same graph as before, but edges now have different types, as shown in Fig. 2(b). This time, initial ISCVs are two-dimensional [see the top matrix in Fig. 2(b)] to account for the two types of edges. From this, the same process as before is repeated until a balanced coloring is obtained and no further refinement is possible. By introducing different types of edges, we obtain a different fibration partitioning compared to Fig. 2(a). Finally, the refinement process for the Kamei–Cock (KC) algorithm is defined by the following steps:

1. Define the initial coloring $\bar{C}_0$, where each node $v \in V$ has a color label $\ell(v)$ associated, and calculate the initial number of colors K.
2. For each node v, define its ISCV(v).
3. Find all the unique ISCVs, assign to each one a different color label, and calculate the new number of different colors K'.
4. The color $\ell(v)$ of each node v is set as equal to its ISCV: $\ell(v) \leftarrow \ell(ISCV(v))$.
5. If set $K \neq K'$, then $K \leftarrow K'$ and go to step 2. Otherwise, the algorithm stops.

Furthermore, there are three important points to be noted for this algorithm: (1) the number of dimensions of the ISCV grows

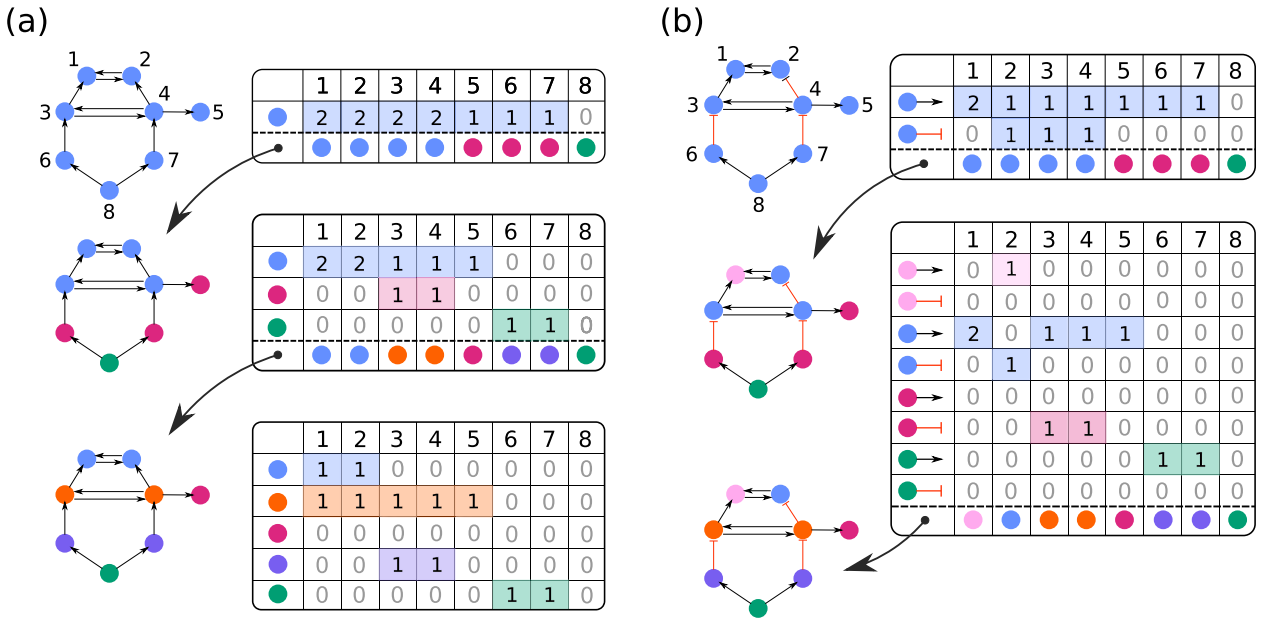


FIG. 2. Belykh–Hasler and Kamei–Cock algorithms. Diagrams (a) and (b) show examples of how the Belykh–Hasler algorithm and the Kamei–Cock algorithm, respectively, work for two distinct networks. For both cases, we define in each iteration the matrix of ISCVs to verify the balancing of the current coloring. In the process, every time the number of colors increases, the matrix increases as well. This process continues until the matrix is balanced and no new colors are necessary. In (a), we consider a network containing eight nodes. Initially, all nodes are assigned to a light blue color. The ISCVs have only one element (one color) and its value is shown at the top left table. Here, all ISCVs are shown together as a matrix. Node 1 receives inputs from nodes 2 and 3, while node 2 receives input from 1 and 4. Thus, both nodes have two inputs of color blue. We note that only nodes 3 and 4 have two inputs from color blue. From this, nodes 1–4 will belong to the same class in the next iteration. In total, there are three unique vectors. Each one of these unique vectors is assigned to a new color and a new partition is created. This time, the matrix of ISCVs grows to dimension 8×3, reflecting the increase on the number of colors. Calculating again the ISCVs, we see that the coloring is not balanced, such that a new refinement is possible. At last, considering the bottom left matrix, we finally arrive to the final balanced coloring, with no new refinement possible, providing the fibers of the given network. For multidimensional edges, for each new color added during the refinement, the ISCV for each node is increased by K_{type}, where K_{type} is the number of edge types. As shown in (b), the presence of new edge's types in the same network of (a) can generate a different balanced coloring.

quickly; (2) the matrix of ISCVs is very sparse, as we can see in the examples shown in Fig. 2; and (3) initial coloring can be assigned to certain nodes. These details are very important especially when the task is to identify fibers in large networks. Therefore, it is easy to grasp the limitations of the method concerning its storage efficiency. For the networks treated in Ref. 2, the KC algorithm does not show any major issue since the authors were only interested in relatively small networks. However, since information network data are continually growing and getting more complex,[31,32] it is pertinent that a method to identify fibrations on large networks should be as efficient as possible. Based on that, we describe in Sec. III a different algorithmic approach, still based on the refinement partitioning paradigm, capable to handle efficiently both with the storage and the runtime in general applications.

III. OPTIMAL IDENTIFICATION OF NETWORK FIBERS

According to the concept of graph fibrations,[8] all nodes inside the same set must receive equivalent information from all the other sets, meaning that the desired partition splits the network into several groups, called classes, each one containing nodes with isomorphic input trees. To obtain that, we treat our problem as the same as finding the coarsest relational refinement partitioning of a set of elements that have a binary relation between them.[26] Since a network is completely defined by a set of node elements and a set of edges that comprehend all the binary relations, this approach can be used. Moreover, because this paradigm should obey the fibration rules, in what follows: (i) we show how a refinement method can be constructed to identify the fibration isomorphisms and (ii) we present the algorithm used to find the coarsest partition in this context and show the implementation of the proposed method.

A. Partition refinement paradigm for input tree isomorphism

Considering the application for a directed network $G(V, E)$, defined by $N = |V|$ nodes connected by $M = |E|$ edges, we can define a network or graph partition $\bar{P}$ over V as a set of pairwise disjoints subsets $\chi^j \subseteq V$ whose union is all V, that is,

$$\bar{P} = \{\chi^j \subseteq V | \chi^i \cap \chi^j = \emptyset, i \neq j\} \tag{3}$$

and

$$V = \bigcup_j \chi^j, \tag{4}$$

where χ^j are the elements of the partition $\bar{P}$, called classes. By taking an additional graph partition $\bar{Q}$ with the property that each one of its classes is contained in the classes of $\bar{P}$, we say that $\bar{Q}$ is a refinement of $\bar{P}$, or $\bar{Q} \preceq \bar{P}$.[21] Moreover, considering a proper refinement, a partition $\bar{P}$ is said to be stable if it is stable with respect to all its classes χ^j. In classical applications, for a class $\chi \in \bar{P}$, we say that χ is stable with respect to a set S if either all elements of χ connects with an element of S or none element of χ points to any element of S. Thus, the goal of a refinement partitioning algorithm is to find the coarsest (minimal number of classes) stable partition over the set V, given a relation $E \subseteq V \times V$. As can be noted, this can be easily extended to graph problems, where the coarsest graph partitioning problem is that of finding, for a given set of directed edges E and an initial partition $\bar{P}$ over V, the minimal number of disjoint classes that are subsets of V forming a stable refinement of $\bar{P}$.

The stability of partitions is the most important concept for the proper identification of equivalence classes and to guarantee the correctness of the algorithm and its termination proof. In the work of Paige and Tarjan,[26] the authors define the stability properties through the definition of a function $s(S, \bar{Q})$ responsible for splitting the classes of the input partition $\bar{Q}$ into stable classes with respect to a set S, also called *pivot set*. The function s defines the *refine* operation, responsible for the refinement of an input partition with respect to a pivot set S, replacing this current unstable partition for a new one that is stable with respect to S. If $s(S, \bar{Q})$ behaves as an identity function for $\bar{Q}$, meaning $\bar{Q} = s(S, \bar{Q})$, then the partition $\bar{Q}$ is already stable with respect to S. The following two properties described in Ref. 26 are the most relevant to our algorithm: the stability inheritance by refinement and the stability inheritance under union. Precisely, any refinement of $\bar{P}$, where $\bar{P}$ is stable with respect to S, is also stable to S, and a partition stable with respect to two different pivot sets S and S' is also stable with respect to their union $S \cup S'$.

The main advantage of the definitions presented is that the stability rule can be modified to represent a more suitable disjoint rule and still preserve the stability inheritances mentioned. With this, we can define a new refinement stability criterion that permits the direct relationship between the refinement partitioning paradigm with the notion of isomorphism between input trees in information networks. Thus, we can use the concept of graph fibrations to determine a disjoint rule that allows the refinement algorithm to find the coarsest graph partitioning where elements inside the same class χ of the final partition all have isomorphic input trees. This way, by a simple modification, we introduce the concept of **input set stability** and its corresponding splitting function s_{input} for the *refine* operation.

In order to identify the group of nodes with isomorphic input trees, we require that the partition should be **input set stable** during each refinement with respect to the pivot sets. In that manner, a graph partition $\bar{P}$ over the network $G(V, E)$ is input set stable with respect to $S \subseteq V$ if, for all the classes $\chi \in \bar{P}$, the following equality is satisfied for all the elements $v, w \in \chi$ and for all the types k of directed edges,

$$|E_k^{-1}(\{v\}) \cap S| = |E_k^{-1}(\{w\}) \cap S|, \tag{5}$$

where $E_k(\{v\})$ and $E_k^{-1}(\{v\})$ represent, respectively, the nodes that directly receives information of type k from v, and the nodes that sends information via a directed edge of type k to v. We stress that both $E_k(\{v\})$ and $E_k^{-1}(\{v\})$ may contain repeated elements. From that, an input set stable graph partition implies that each class receives equivalent information from all the other classes, including from itself. From this, it is straightforward to check that the inheritance by refinement and the inheritance by union of classes is also valid for s_{input}, where the rule given by Eq. (5) guarantees the partitioning in disjoint sets.

After we find the coarsest stable graph partition $\bar{P}$ following the input set stability rule, there will not be any pivot set $S \subseteq V$ and $S \not\subset \chi$ for which the partition $\bar{P}$ is not input-set stable, meaning that for any class $\chi \in \bar{P}$, we have $\bar{P} = s_{input}(\chi, \bar{P})$. As a consequence, any two nodes belonging to the same class receive equivalent information from the rest of the network, including from their own class, implying that they have the same input set. Fortunately, if two nodes in an input set stable network partitioning have the same input set, we can use the results presented by Norris[33] to show that these nodes have isomorphic input trees. According to Norris,[33] for a directed network with N nodes, an isomorphism for any depth k between the input trees of two nodes can be guaranteed as long as these trees are isomorphic up to the $N-1$ depth, where the depth k of a tree is the set of nodes with k edges between them and the root node. Therefore, we can state that the coarsest input set stable graph partition obtained by the refinement procedure using s_{input} is the union of pairwise disjoint sets, where each one of these corresponds to a set of nodes with isomorphic input trees. This result shows a natural mapping between the refinement partitioning paradigm and the identification of fibers in information-processing networks.

B. Fast fibration partitioning

As we have discussed in Sec. III A, to refine a partition, it is necessary a pivot set S to split the classes of the current partition into different classes, each one input set stable with respect to S. In order to do that, one should choose an appropriate list of pivot sets. For instance, if we consider a partition $\bar{P}$ to show that it represents the correct partitioning in groups of isomorphic input tree nodes, it is necessary to verify that $\bar{P}$ is input set stable with respect to each one of its classes. By doing so, it is possible to design an algorithm capable of determining the fibration partitioning of a network by a general straightforward procedure: starting with an initial network partition, refine the current partition with respect to each one of its classes until there is no more unstable class. If applying s_{input} for every class of the current partition leads to no further refinement, then the current partition is input-set stable and corresponds to the correct minimal balanced coloring of the network. However, even though this procedure gives the right answer in a reasonable runtime and it represents the essence of a refinement algorithm, it also gives a high number of unnecessary and repeated operations. Therefore, it is important to choose the appropriate pivot sets during the refinement in order to design an efficient method for the partitioning.

The algorithm we build here is a modified version of the algorithm introduced by Paige and Tarjan,[26] with runtime complexity of $\mathcal{O}(M \log N)$. Although this algorithm was designed for a

different problem, namely, the relational coarsest partition problem, we emphasize that this problem is easily mapped to the identification of fibers on graphs, as pointed out in Sec. III A and in Refs. 8 and 24. By introducing the input set stability definition for graphs, we use the algorithm of Paige and Tarjan as a foundation of our method. Compared to the similar algorithm of Cardon and Crochemore,[27] the Paige–Tarjan algorithm has a simpler implementation and smaller runtime prefactors since it avoids irrelevant pivot set selection, exhibiting a better performance to our problem.

For the identification of an input set stable graph partition, we can benefit from the input set stability properties to construct a refinement algorithm that should achieve, through a finite number of steps, the minimal input set stable partition originating from an initial trivial network partitioning. A refinement step should have the effect to refine the current partition of V, unstable with respect to a pivot set $S \subseteq V$, by replacing it for a new partition, now stable for S. For this, we use the splitting function $s_{input}(S, \bar{P})$, which receives, as input, the current partition $\bar{P}$ over the network and the pivot set S, returning as output a new stable partition with respect to S. As pointed out, s_{input} benefits from two stability properties: the inheritance by refinement and the inheritance by union of sets. Because of these inheritances, a pivot S should be used only once by s_{input}, which guarantees that the partition, after all refinement steps, will preserve the stability with respect to S and to the union $\cup_i S_i$ of all pivot sets S_i already used. From these properties, the core of the *refine* operation can be stated as the three following steps:

1. Choose a pivot set S in which the current graph partition $\bar{P}$ is input set unstable.
2. Identify all the classes $\chi_{/S} \in \bar{P}$ that is input set unstable with respect to S and split each one of these into stable classes $\chi_S^j \subset \chi_{/S}$ with respect to S.
3. Replace $\bar{P}$ by $\bar{P}_S = s_{input}(S, \bar{P})$, where for $\bar{P}_S$ each $\chi_{/S}$ is replaced by its split classes χ_S^j.

Through these steps, the *refine* operation can be executed with a time complexity of the order $\mathcal{O}(|S| + \sum_{v \in S} |E(\{v\})|)$. To obtain this time complexity, we should first note that the only possible input set unstable classes $\chi_{/S}$ from $\bar{P}$ are the ones who receive input from the nodes of S. Therefore, we need to iterate over all $|S|$ nodes $v \in S$ and then iterate over all the $|E(\{v\})|$ outgoing edges of each v to properly define all unstable classes $\chi_{/S}$, which are the ones that contain nodes where the condition from Eq. (5) is not satisfied. Thus, the *refine* operation is linear either with the number $\sum_{v \in S} |E(\{v\})|$ of outgoing edges from S or with the number of nodes $|S|$ depending on which one is larger. This ensures the *refine* operation as optimal since we only need to identify the outgoing information leaving each element of S and arriving at the possible input set unstable classes. Since the finest partitioning possible is the one in which every node is itself a class, and because the number of classes increases after each refinement step, the refinement by pivot sets may be executed at most $N - 1$ times. Furthermore, because of the splitting dynamics of the algorithm, each node $v \in V$ is in at most $\log N$ different pivot sets S, as also described in Ref. 26. Therefore, using the time complexity of the *refine* operation for every node, used as a part of a pivot set at most $\log N$ times, we obtain the $\mathcal{O}(M \log N)$ runtime of the algorithm.

Through this, given a pivot set S and a given input set unstable network partition $\bar{P}$, the classes $\chi_{/S} \in \bar{P}$ that are unstable with respect to S can be split into several disjoint classes χ_S^j input set stable with respect to S. Considering the general situation for multiplex networks containing n types of edge, we can denote the split classes as $\chi_S^{j_1 j_2 \cdots j_n}$ related with their unique tuple $(j_1, j_2, \ldots, j_n)_S$ regarding their stability with respect to S. Then, each split class must obey, for k from 1 to n, the property defined by

$$\chi_S^{j_1 j_2 \cdots j_n} = \{v \in \chi_{/S} : |E_k^{-1}(\{v\}) \cap S| = j_k\}, \tag{6}$$

where the number of split classes must be larger than one. Among all split classes obtained from $\chi_{/S}$, the largest one can be excluded as pivot in the following steps of the algorithm. This is a consequence of the disjoint nature of the refinement process and its stability inheritance property. For instance, if we refine a partition with respect to a pivot set and that pivot is responsible for splitting one unstable class χ into two stable classes, χ_1 and $\chi_2 = \chi - \chi_1$, then it is necessary to check only the partition stability with respect to one of these split sets. The complement set is just a redundant pivot, due to the fact that $\chi_1 \cap \chi_2 = \emptyset$ and that the refined partition is input set stable with respect to $\chi = \chi_1 \cup \chi_2$. This ensures that no repeated, redundant, or union of repeated sets is used during any step of the algorithm.

At this point, we can finally state the complete algorithm to identify the correct partition $\bar{F}$ of a directed network $G(V, E)$, representing a balanced coloring. The first step is to define the initial partitioning $\bar{P}_0$ which requires some preprocessing. We provide the details of this step in Sec. IV. Here, we can consider the simpler case, where $\bar{P}_0 = \{V\}$. We also define L as a queue data structure to store the pivot sets during the execution of the algorithm. From this, the algorithm, which we denote as fast fibration partitioning (FFP), is given by the following steps:

1. Define the initial partition $\bar{P}_0$, push all its classes to a queue L, and set $\bar{P} = \bar{P}_0$.
2. Remove from L its first pivot set S.
3. Replace $\bar{P}$ by $s_{input}(S, \bar{P})$.
4. Whenever a class $\chi \in \bar{P}$ splits into two or more nonempty classes, push all these to the back of L, except the largest one.
5. If L is not empty, go back to step 2. Otherwise, the algorithm stops and the final partition $\bar{F}$ represents the minimal balanced coloring.

At the end, the final partition $\bar{P} = \bar{F}$ represents the coarsest input set stable partition, or minimal balanced coloring, of $G(V, E)$ and each class of $\bar{F}$ is a fiber.

The final algorithm has time and space requirements of order, respectively, $\mathcal{O}(M \log N)$ and $\mathcal{O}(M + N)$, which allows its application for large networks. It has a simple implementation, where the refinement process, pictured in Fig. 4, represents the most complex part of the algorithm.

IV. INITIALIZATION AND IMPLEMENTATION DETAILS

As pointed out, to correctly implement the refinement algorithm, it is very important that the initial partition, represented by $\bar{P}_0$, satisfies determined conditions to ensure that the nodes will

not be placed inside wrong classes of the final partition $\bar{F}$. For this, given the network $G(V, E)$ and before any refinement, we can define specific groups of nodes based on their input from other nodes. These groups are related with the network $G_{scc}(V_{scc}, E_{scc})$, obtained by the partitioning of G into strongly connected components (SCC). In this way, each element of V_{scc} represents the set of nodes that are reachable from all the other nodes in this same element via a directed path. For convenience, we define isolated nodes in V_{scc} as components themselves. From this, to initialize the partition $\bar{P}_0$, we assign to each component $v' \in V_{scc}$ a class label according to the condition whether v' receives or not input from other components $w' \in V_{scc}$, where $v' \neq w'$. This step zero is necessary because in real networks, in particular, in the context of biology,[2,10] each node is expected to receive an input in order to perform a biological function coordinated with the rest of the system. Therefore, we assume that nodes with no inputs actually have inputs that are not present in the network due to missing data or the simplicity of the model. That is, incoming links either have not been found yet or the incoming link is coming from another type of object that is ignored in the network, for example, in case of the gene regulatory network, it can be an input coming from a metabolite. Therefore, to avoid imposing extra symmetries by giving no-input nodes the same color, we assign them different colors and assume they cannot have correlated dynamics based only on the admissible ODEs of the network, as also done in.[2,10,13] This may be confusing to the reader because it contradicts the definition of the minimal balanced coloring.

Strictly speaking, nodes with no inputs have isomorphic input sets and input trees consisting of the node itself rendering such nodes with the same minimal balanced color. As discussed above, from a standpoint of the biological system, such coloring is not reasonable. Hence, the coloring we consider is not exactly minimal, but rather as minimal as possible without violating the condition on the nodes with no inputs. To avoid unnecessary complexity, we abuse the notation here and call such coloring minimal while keeping this distinction in mind.

We define two types of components that must be separated before the application of the refinement process. The first type defines any SCC that does not receive any input from the rest of the network. Each component of this type, represented by $v' \in v_{scc}$, plus all nodes belonging to a connected component receiving external input only from v', and not other SCC, defines one different class $\chi_{v'}$ to be put in the initial partition (pink nodes in Fig. 3). The second type is any component not included by the first type (blue nodes in Fig. 3). All nodes belonging to these second type components are put into the same unique class χ_0 at the beginning of the algorithm. Besides bigger SCCs with no inputs, this first type allows for isolated nodes without incoming edges in G (red nodes in Fig. 3). However, the existence of these isolated components are not, in general, feasible in some real applications, like genetic regulatory networks. In this case, for every gene, there should be at least one transcription factor responsible for its regulation, implying that all nodes in these types of networks have input, even if only from themselves.

The initialization can be described by the following steps: (i) Define the partitioning G_{scc} into elements as SCC. (ii) Then, classify each component as the first or second type exactly as previously explained. (iii) Create for each first type component $v'_j \in V_{scc}$ a new class $\chi^j_{v'}$ including all nodes of the SCC v'_j and any node from a connected component receiving input only from v'_j [self-loops are discarded from this case as exemplified in Fig. 3(b)]. (iv) Put all nodes belonging to the second type component into the class χ_0. (v) Finally, push all these classes into the pivot set queue L. After these steps, the initial state of the algorithm will be defined by $\bar{P}_0$ and L as follows:

$$\begin{aligned} \bar{P}_0 &= \{\chi_0,\ \chi^j_{v'}\}, \\ L &= \{\chi_0,\ \chi^j_{v'}\}, \end{aligned} \tag{7}$$

allowing the proper application of the FFP algorithm.

For the implementation, we can define the partition $\bar{P}$ holding all the classes by a doubly linked list, which allows fast deletion of classes in constant time $\mathcal{O}(1)$ as long as we have the memory address of the class during the procedure. Alternatively, we can define $\bar{P}$ as a simple indexed list containing classes. If we choose to not perform the deletion of classes, each class is a doubly linked list. Each component of the linked list is a node belonging to the class. This structure allows that a class χ does not have to be deleted during the splitting process, but only reduced by removing its nodes. This way, instead of deleting classes we only need to create new ones to store the nodes removed from other classes. Moreover, it is necessary that each node at each iteration has a pointer to its class, or simply to the index in the list $\bar{P}$ where its class is located. This pointer is updated whenever a class is split. Since the splitting condition reduces to checking Eq. (6), and L can be implemented by a simple queue, the data structures described here are the main ones for an efficient implementation of the algorithm.

The reader can refer to the link https://github.com/makselab/FastFibration to access the Julia implementation of the fast fibration partitioning algorithm as described in this work. R library for finding fibration partitioning applying the algorithm developed in Ref. 2 along with functionality for fiber building blocks construction, classification, and significance detection are available at https://github.com/makselab/fibrationSymmetries. All codes are written in high performance languages and further instructions and documentation for their proper use can be found in the referred links.

V. ALTERNATIVE METHODS

As already mentioned, fibers are equivalent to the classes of a minimal balanced coloring over a network, which can be obtained by the methods presented so far. In what follows, we present two different approaches previously developed to find balanced colorings that can be used to identify fibration symmetric circuits and then we compare the runtime and space complexities of the algorithms with ours.

A. Boldi and Vigna's algorithm

The first approach was proposed in 2002 by Boldi and Vigna in their seminal work of graph fibrations.[8] Presented originally as a theorem, the idea is to construct the input tree for each node in order to verify the isomorphism between the nodes' input tree, and use the theorem by Norris[33] as the stop criterion. The initial partition

is defined as a single class containing all the nodes, and the refinement operation is based on the search for isomorphism through the first layer until the $(N-1)$th layer of the input trees. Two nodes are said to be equivalent under $\simeq_k$ if their input trees are isomorphic up to level k. Since a node is always isomorphic to another node, therefore all nodes are equivalent under $\simeq_0$. Moreover, an equivalence relation states that $v \simeq_{k+1} w \iff v \simeq_k w$ and then there exists a bijection ψ from the input set of kth layer of the input tree of v to the input set of kth layer of the input tree of w, such that it satisfies the isomorphism condition, i.e., the image of the edge connects the image of the source and the image of the target of this edge.

In Fig. 5(a), we show an example for a Fibonacci circuit introduced in Ref. 2. For this type of circuit, the number of nodes in a kth level is described by a Fibonacci sequence and, to obtain a new partition, we need to check the isomorphism condition for the number of elements in all levels up to $N-1$. In Ref. 8, the authors propose an algorithm to check the isomorphism between input trees with total runtime of order $\mathcal{O}(N^2 d \log N)$, where d is the maximum in-degree in the network, and memory of order $\mathcal{O}(N \log N)$. However, this runtime can still be very restrictive for large networks, not being suitable for general applications.

B. Aldis's algorithm

An alternative approach for the Kamei–Cock algorithm was proposed by Aldis[22] in 2008. According to his algorithm, all nodes are placed inside the same equivalence class and further partitions are obtained by a refinement process based on the type of the edge and the source of this same edge, similar to the input driven refinement introduced in Sec. II. Aldis's approach consists of two main procedures, both of which are implemented in every iteration. We use the notation $n_t(v), n_t(e) \in \mathbb{N}$ to describe the color of the node v or the edge e in the iteration t. Once the algorithm stops, the balanced coloring is obtained.

In the first procedure, each edge is associated with a pair of the edge color and the edge source color $(n_t(e), n_t(s(e)))$. Then, we list all unique pairs at the current iteration. Each unique pair is associated with a new edge color and the edge coloring of the network is updated. If there are no new colors after the update at this iteration, the algorithm stops unless $t = 0$. During the second procedure, each node v is associated with the colors of the set of edges that points to it, $\langle n_t(e) | t(e) = v \rangle$, where n_t in each set are listed in an increasing fashion and $t(e)$ is the target node of the edge e. Similarly to the ISCVs, we identify all the unique sets and assign to each one a new

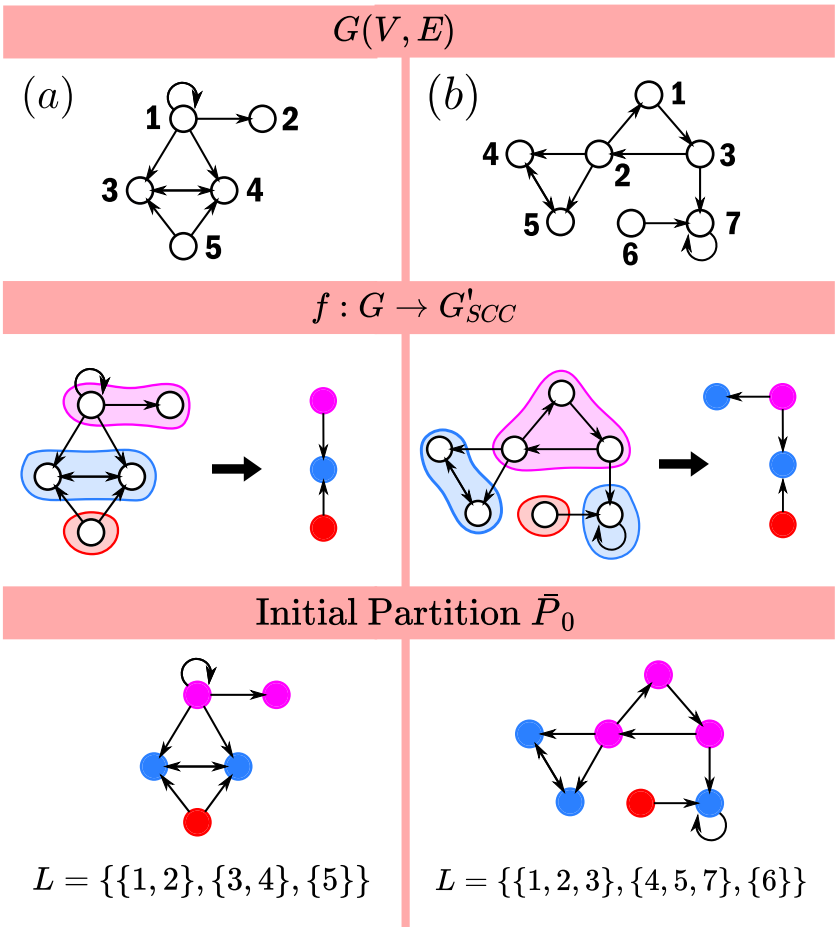


FIG. 3. Definition of the initial partition $\bar{P}_0$. Figures (a) and (b) display two small network examples where we need to take into account different conditions to define correctly the initial partition $\bar{P}_0$. Starting from a given network $G(V,E)$, we first obtain all the strongly connected components (SCC) present in the network to define G_{SCC}. After that, we classify each node of G_{SCC} according to two different conditions. In both (a) and (b), we observe the presence of SCCs that receive input from other SCCs, and they are highlighted with the color blue. All nodes of G belonging to blue nodes of G_{SCC} must be placed into the same class χ_0 of $\bar{P}_0$. In both networks, we note the existence of a strong component that does not receive input from any other different component. This component is highlighted with color pink. For each pink element $v' \in V_{SCC}$, we define a new class $\chi_{v'}$ in $\bar{P}_0$ containing the nodes inside this element and all nodes from a connected component (not a SCC) receiving external input only from v', as shown in (a) for node 2. Moreover, all nodes with no input are also placed in separated classes (red nodes).

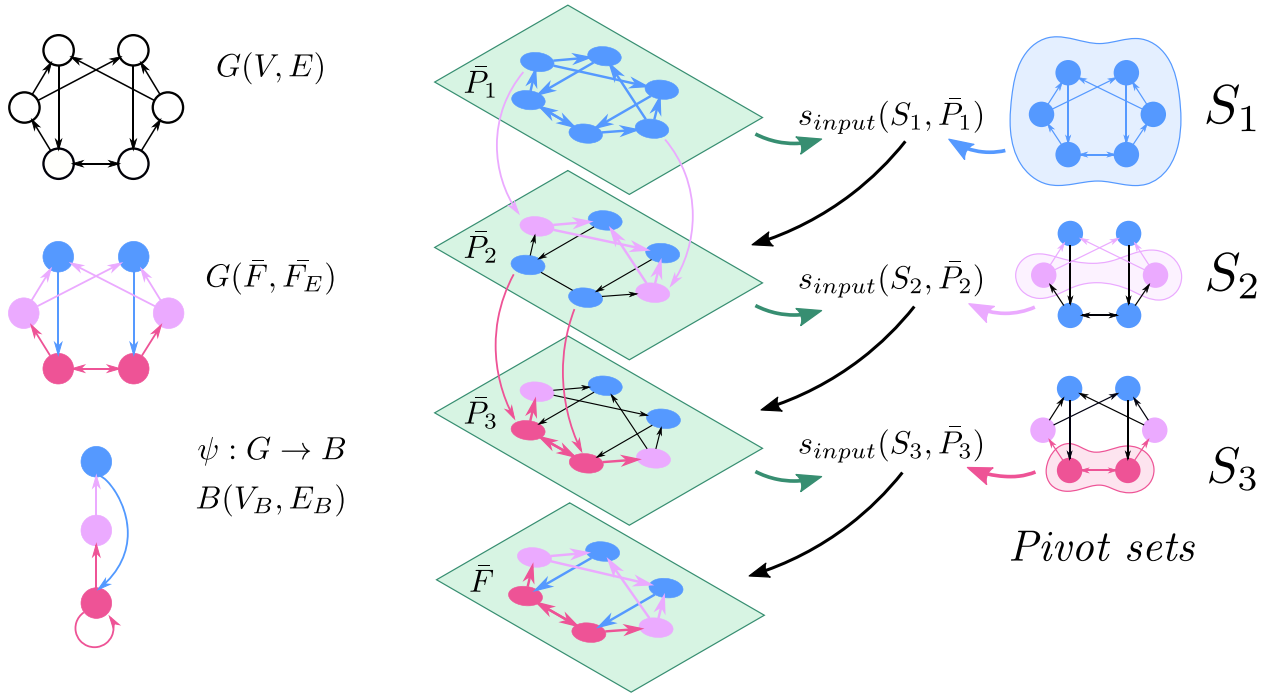


FIG. 4. Refinement mechanism of the splitting function s_{input}. Here, we consider the directed network $G(V, E)$ shown at the left. We start initializing the algorithm by defining the first pivot set S_1 as the initial partition $\bar{P}_1$, containing all nodes of the network. The initial state has $\bar{P}_1$ as partition and $L = \{S_1\}$ as the pivot set queue structure. Following the refinement procedure, we select S_1 as the first pivot set and then refine the current partition obtaining $\bar{P}_2 = s_{input}(S_1, P_1)$, which splits the single blue class into two disjoint classes input set stable with respect to S_1: a upgraded reduced blue class and the new light-pink class. Since the blue class is the largest split block (containing four nodes against two from the pink class), we insert the pink class at the back of L, defining it as the pivot set S_2. Because $L = \{S_2\}$, we repeat the process this time using $\bar{P}_2$ and S_2 as input for s_{input}, which returns a new partition $\bar{P}_3 = s_{input}(S_2, \bar{P}_2)$. The blue class is split into two equal size classes and the new one is used as the pivot set S_3, right after be placed at the back of L. However, $\bar{P}_3$ is input set stable with respect to S_3, implying that none class of $\bar{P}_3$ is split. For this reason, L is now empty and the algorithm stops. The final partition $\bar{F}$ represents the minimal balanced coloring over $G(V, E)$. As a consequence of the partition $\bar{F}$, the set of directed edges is also defined as a partition $\bar{F}_E$, which distinguishes the information coming from different fibers. Finally, from this partitioning, we can define the new network $B(V_B, E_B)$, representing the base network that resulted from the fibration morphism $\psi : G \to B$.

color, from which we update the coloring of the nodes. If after this update, there are not new colors compared with the last iteration, the algorithm stops, and the balanced coloring is obtained.

Figure 5(b) shows the partitioning of a small graph containing five nodes according to Aldis's[22] algorithm. Initially, all nodes are assigned to the same class as well as the edges. Starting from iteration $t = 0$, the pairs $(n_0(e), n_0(s(e)))$ are associated with each edge, which in this case is always $(1, 1)$. In this step, all edges stay in the same class and coloring update does not introduce any new color. However, the algorithm does not stop because $t = 0$. Still at $t = 0$, the set $\langle n_0(e) | t(e) = v \rangle$ is associated to each node v. From this, there are two unique sets and we assign to each one of them a color [in this case, $\langle 1 \rangle \to$ red(1), $\langle 1, 1 \rangle \to$ blue(2)] and we update the node color according to these labels. After that, we go to the next iterations $t = 1, 2, \ldots$ repeating the same procedures until no further refinement is possible, as displayed in Fig. 5(b), and the balanced coloring is reached.

Here, we stress that this algorithm allows for both the set of nodes and the set of edges of the network to have an initial coloring different from the trivial (all nodes with the same color) since the two-procedure implementation guarantees that the algorithm does not depend on the initial conditions. Furthermore, as compared to the Kamei–Cock algorithm, the advantage of Aldis's approach is that it does not need to use a matrix to store the input set vectors, avoiding any problem regarding the sparse structure of the ISCVs. However, the algorithm exhibits an inferior runtime complexity than the KC coloring algorithm[16] (see Table I).

VI. MEASURING THE RELATIVE PERFORMANCE OF THE ALGORITHMS

By now, we have presented four methods to identify cluster synchrony in information-processing networks. Within these four, we argue that the method proposed in this work, which we call fast fibration partitioning (FFP), is the most efficient algorithm so far, being capable to handle efficiently with both computational memory and time resources. As we have described in Secs. I–V, both FFP and KC algorithms are based on the refinement paradigm.

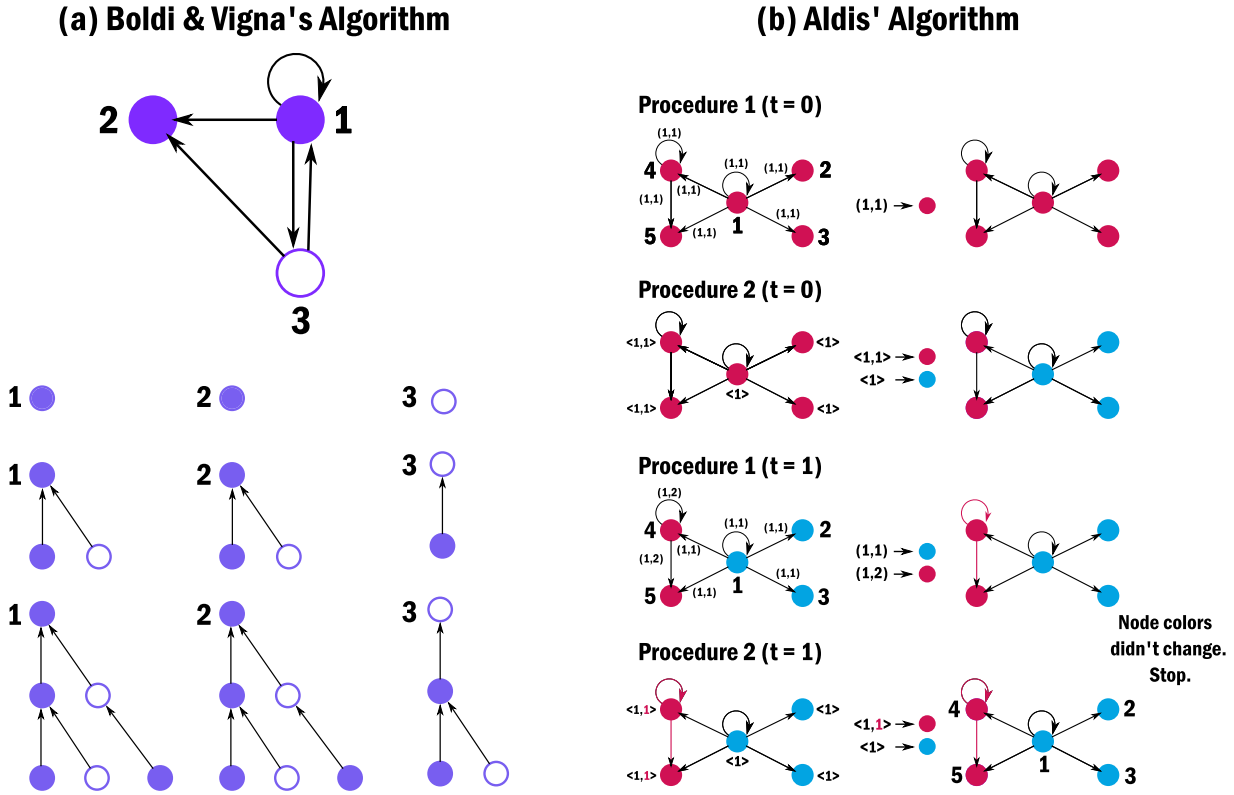


FIG. 5. Alternative algorithms to find fibers. In (a), we see a Fibonacci circuit, introduced in Ref. 2. Following the approach of Boldi and Vigna,[8] at the initial step, all three nodes have the same input tree of depth 0, so the initial partition puts all nodes in the same class. The second step shows that node 3 is already non-isomorphic to 1 and 2; however, we need to go further $N - 1 = 2$ levels to assign the final isomorphism between input trees. At the final level, the algorithm assigns the isomorphism between the input trees of 1 and 2, defining the Fibonacci fiber. Finally, (b) exemplifies the Aldis's[22] algorithm for the case detailed in Sec. V.

However, due to the minimization nature of the problem, it is possible to design several refinement routines that require different time and space complexities. This is a consequence of the greatest fixed point theorem, demonstrated by Tarski in 1955.[34] This theorem states that we can define the refinement operation by a monotonic function g that splits every unbalanced (or unstable) class into several disjoint balanced classes. Being $g(\bar{P})$ a function of a partition $\bar{P}$ over V, the theorem states that when the greatest fixed point of the function g is reached, meaning the minimum number of classes within $\bar{P}$ such that $g(\bar{P}) = \bar{P}$, then the resulting partition $\bar{P}$ represents the one containing the minimal number of classes over the given sets V that it is stable with respect to the relation $E \subseteq V \times V$. As showed in Ref. 20, the general approach through a refinement paradigm can be defined as the following steps:

1. Define the partition $\bar{P}$ through an initial partition $\bar{P}_0$ over V.
2. Refine the current partition: $\bar{P} \leftarrow g(\bar{P})$.
3. If the partition do not change, the algorithm stops. Otherwise, go to step 2.

TABLE I. Summary of the time and space complexities of the algorithms mentioned in this work. In the algorithm from Boldi and Vigna, d is the maximum in-degree in the network.

	Time complexity	Space complexity
FFP	$\mathcal{O}(M \log N)$	$\mathcal{O}(M + N)$
KC[16]	$\mathcal{O}(N^2 \log N)$	$\mathcal{O}(N^2)$
Aldis[22]	$\mathcal{O}(N(M + N)^3)$	$\mathcal{O}(M + N)$
Boldi and Vigna[8]	$\mathcal{O}(N^2 d \log N)$	$\mathcal{O}(Nd \log N)$

Through this approach, it is possible to construct implementations either with runtime $\mathcal{O}(N^2 \log N)$ or $\mathcal{O}(M \log N)$, where N and M represent, respectively, the number of nodes and the number of edges of the network. Considering graph fibrations, we can define the splitting functions both for the Kamei–Cock coloring (KC) and the fast fibration partitioning algorithm (FFP) as $s_{coloring}$ and s_{input}, respectively. s_{input} is the splitting function defined earlier in this work, while $s_{coloring}$ is a similar function also respecting the input set stability, but implemented according to the mechanism of the KC algorithm. As previously discussed in Sec. III, the

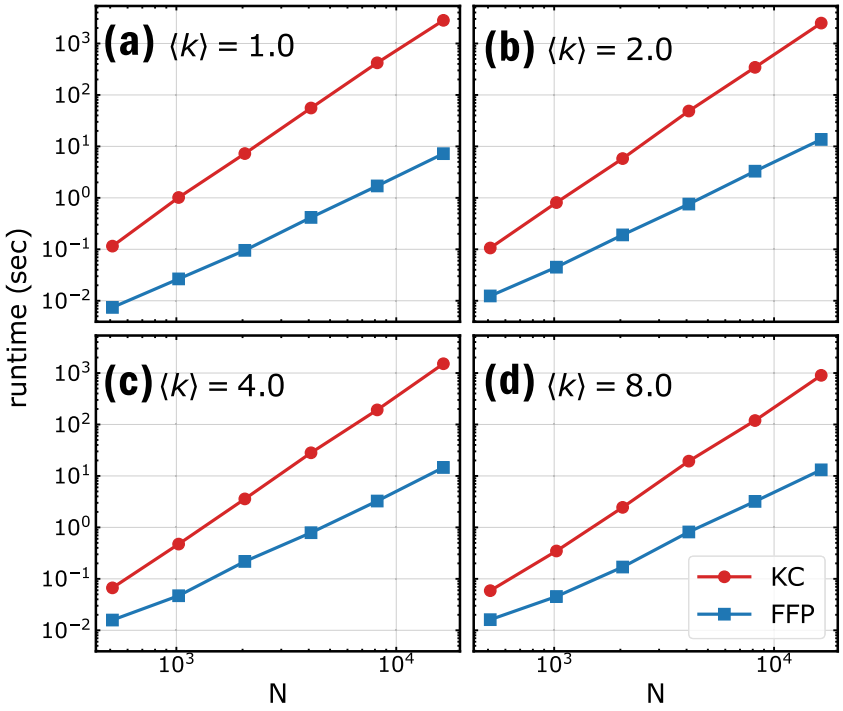


FIG. 6. Comparison between the runtime variation of the KC (red circles) and FFP (blue squares) algorithms as a function of the size N of Erdös–Renyi directed networks. Here, networks have mean degrees $\langle k\rangle = 1$ (a), 2 (b), 4 (c), and 8 (d). The symbols correspond to averages calculated over 20 network realizations and the size N ranges from 512 to 16 384 nodes. In all curves, the errors are smaller than the symbols. As $\langle k\rangle$ increases, i.e., the network becomes denser (and the symmetries become trivial), the performance of the FFP algorithm is decreased, as expected from its time complexity $\mathcal{O}(M \log N)$, while the KC's performance improves, as it requires less iterations to find the trivial partitioning. However, this improvement is bounded as the network loses its symmetries, for which the large gap between both performances remains clear, where the FFP method is faster than the coloring algorithm.

FFP algorithm, with the s_{input} splitting function, performs all refinement operations with runtime equal to $\mathcal{O}(M \log N)$. On the other hand, the KC algorithm, through its $s_{coloring}$ splitting function, has an overall runtime of at least $\mathcal{O}(N^2 \log N)$, as we show as follows.

In the Kamei–Cock algorithm, for each step, the algorithm verifies if each class is balanced by comparing the ISCVs of each node inside the current classes. In that manner, every node has its ISCVs checked and compared with the nodes sharing its same class. These comparisons and the splitting of unbalanced classes demand a runtime, which grows with the size of the matrix of ISCVs, since each node has its ISCV verified. We notice that the matrix containing all ISCVs has $K^i \times K_{type} \times N$ entries, where K^i corresponds to the number of colors at iteration i and K_{type} is equal to the number of edge types within the network. Since the refinement process allows a maximum number of N colors for the worst-case scenario, the time complexity of the Kamei–Cock must be at least quadratic, which is the same order of the space resources necessary to its implementation. Still considering the worst-case scenario where the final coloring contains N colors, we can verify that to refine an initial coloring, where all nodes belong to the same class, into the final one with N colors we need an order of $\log N$ iterations. For instance, if we start the refinement with a single class containing all nodes of the network, and for each iteration, each class is split into two new classes with half the size of the split class, we expect exactly $\log N$ steps to obtain N classes. Therefore, the time complexity of the Kamei–Cock algorithm is $\mathcal{O}(\alpha N^2 \log N)$, where $\alpha = K_{type}$ represents the number of types of edges within the network.

To verify these considerations, we performed the comparisons between both algorithms, KC and FFP for two different cases in Erdös–Renyi networks. In general, fibration symmetries are rarely found in these networks, which allows us to assess the runtime of the methods in their worst-case scenarios. In Figs. 6(a)–6(d), we show the runtime variation of the KC and FFP algorithms as a function of the size of the Erdös–Renyi networks generated with mean degrees $\langle k\rangle = 1, 2, 4$, and 8, respectively. As expected, the FFP algorithm performs much better in sparse networks, with slightly decreasing performance for denser networks. Furthermore, for all cases, the runtime performance of FFP overcomes the one of the KC algorithm. Considering the second case, in Fig. 7, we perform the same analysis but varying the number of edge types for multiplex random networks. As previously mentioned, the presence of several types of edges affects drastically the performance of the KC algorithm, while the performance for the FFP almost does not change. Since for each new edge type introduced in the network the KC algorithm doubles the size of the ISCV matrix, its performance becomes restrictive even for moderate sizes of multiplex networks. As shown in Fig. 7, the runtime difference between the performance of the KC algorithm for $K_{type} = 1$ and for $K_{type} = 8$ is almost 200 s for a network of size $N = 2048$ and mean degree $\langle k\rangle = 1.0$, while this difference is about 0.02 s for the runtime of the FFP algorithm.

At the same time, we recognize that the performance comparison for the main algorithms described is not of interest regarding applications for dynamical systems on networks. Since the patterns of synchrony in random networks are mostly trivial, it is also informative to assess the comparison between performance of the algorithms for real networks where it is known in advance that they have nontrivial fibration partitioning. To perform this assessment, we have selected ten different networks with different sizes from different application contexts. We acquire the datasets from the previous work on fibrations,[2] where the nontrivial partitioning of the

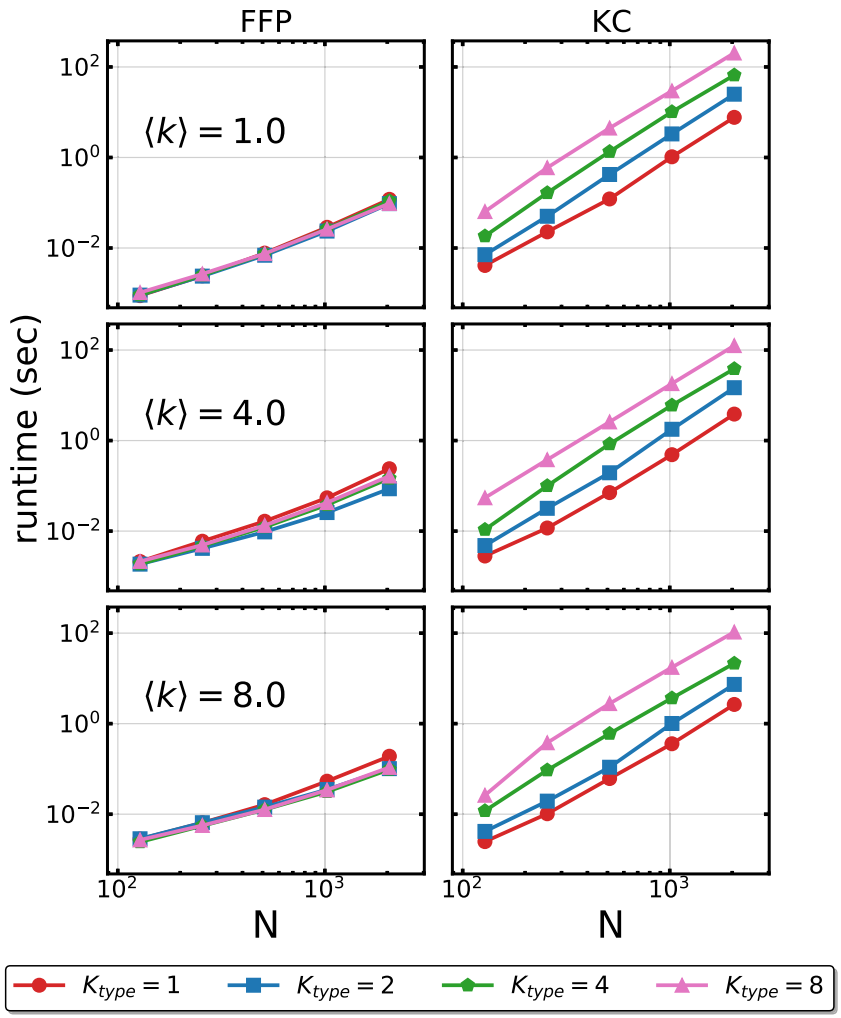
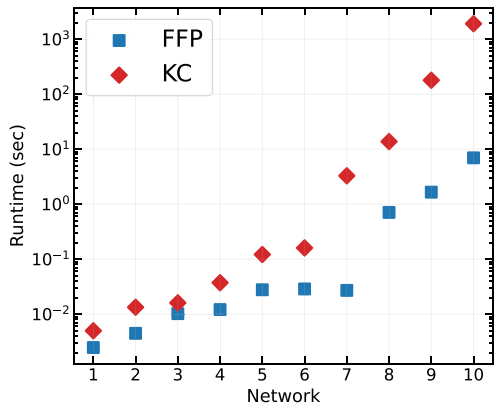


FIG. 7. Comparison between the runtime variation of the FFP (left column) and KC (right column) algorithms as a function of the size N of Erdös–Renyi directed networks with different numbers of edge types. Here, networks have $K_{types} = 1$, 2, 4, and 8 edge's types, and mean degrees $\langle k \rangle = 1$, 4, and 8. The symbols correspond to averages calculated over 20 network realizations and the size N ranges from 128 to 2048 nodes. In all curves, the errors are smaller than the symbols. Since the KC algorithm stores an ISCV matrix containing $K_{type} \times K \times N$ entries, where K represents the number of colors, the variation of the number of edge types increases the number of required operations significantly. On the other hand, the FFP is much less sensitive to K_{type} than the KC algorithm, being linearly dependent only on the size of the current pivot set S during the iterations. Since the runtimes for the KC are very restrictive, we performed these comparisons only for very small networks.



	Dataset name	Category	N	M
1	cAMP-signaling-pathway	Biological	196	1077
2	Focal-adhesion	Biological	207	2151
3	econ-wm3	Economical	259	2948
4	inf-USAir97	Infrastructure	332	2126
5	airport	Infrastructure	500	5960
6	Micobacterium-tuberculosis	Biological	1624	3212
7	trrust-rawdata.mouse	Biological	2456	7057
8	soc-sign-bitcoinalpha	Social	3783	24186
9	p2p-Gnutella08	Internet	6301	20777
10	web-indochina-2004	Internet	11358	47606

FIG. 8. Comparison between the runtime of the FFP (blue) and KC (red) algorithms for several real networks. For each real network, described in the right table and obtained from Ref. 2, we have applied both algorithms 30 times and obtained the mean performance in each case. Because of the prohibitive performance of the KC algorithm for networks 8, 9, and 10, we have done only ten performance evaluations. Nevertheless, for all networks, the errors are smaller than the symbols.

networks was obtained. In Fig. 8, we display the runtime for each one of the networks selected together with their respective information on names, network category, and size. Consistent with the simulations on random networks, the FFP algorithm exhibits superior performance on real networks compared with KC algorithm, especially when we compare the performance for larger networks, as shown for networks of labels 8, 9, and 10 of Fig. 8.

Therefore, we observe that the method proposed here, based on the algorithm of Paige and Tarjan,[20] outperforms the runtime performance of the Kamei–Cock algorithm used in Ref. 2. Furthermore, since the fast fibration partitioning algorithm requires only linear memory resources, the method also overcomes the KC algorithm in space resources since the latter has at least $\mathcal{O}(N^2)$ complexity. In Table I, we list the time and space complexities of each algorithm mentioned in this work. Regarding the algorithms of Boldi–Vigna and Aldis, we provide only the comparison between their complexities with the FFP algorithm. As we observe in Table I, the time complexity of Aldis's algorithm does not allow us to compare its performance since it has runtime[22] $\mathcal{O}(N(M+N)^3)$ and, hence, it is very constrained to small graphs. The Boldi–Vigna algorithm introduced in Ref. 8 with runtime $\mathcal{O}(N^2 d \log N)$ was designed for demonstration purposes and for practical applications the authors of the algorithm[8] used the method of Cardon and Crochemore,[27] from which the Paige–Tarjan algorithm used in this work is an improvement.[26]

VII. CONCLUSIONS

The methods presented in this work represent the main collection of algorithms designed for the identification of fiber circuits. There is a remarkable presence of these circuits in several biological and non-biological networks, and, as shown in the very recent works,[2,10,35] each fiber leads to the synchronization between groups of nodes. Not only that, the symmetry breaking of these fibers can also lead to circuits analogous to logic computational units,[10] affecting directly the functionality of the systems to which these circuits belong to. In summary, we have presented in this paper an optimal procedure to automatically identify fibrations in large networks. We showed that the balanced coloring method defined by the FFP algorithm outperforms, both in terms of runtime and space resources, other methods previously presented in the literature.

ACKNOWLEDGMENTS

This work was funded by National Institutes of Health-National Institute of Biomedical Imaging and Bioengineering (NIH-NIBIB) and National Institutes of Health-National Institute of Mental Health (NIH-NIMH) through the Brain Initiative (Grant Nos. R01 EB028157 and R01 EB022720). Financial support was also provided by the Brazilian agencies Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), and Fundação Cearense de Apoio ao Desenvolvimento Científico (FUNCAP), and the National Institute of Science and Technology for Complex Systems.

AUTHOR DECLARATIONS

Conflict of Interest

The authors declare no conflict of interest.

DATA AVAILABILITY

Code for the Fast Fibration Partitioning algorithm is available in Github at https://github.com/makselab/FastFibration, Ref. 36, together with its documentation for proper use. All data used, both synthetic and real networks, are available at https://osf.io/4t7ev/. We also provide the codes for the Kamei-Cock algorithm used in previous works with extended functionalities in Github at https://github.com/makselab/fibrationSymmetries, Ref. 37.

REFERENCES

[1] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," Science **298**, 824–827 (2002).
[2] F. Morone, I. Leifer, and H. A. Makse, "Fibration symmetries uncover the building blocks of biological networks," Proc. Natl. Acad. Sci. U.S.A. **117**, 8306–8314 (2020).
[3] S. H. Strogatz, "Exploring complex networks," Nature **410**, 268–276 (2001).
[4] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. U. Hwang, "Complex networks: Structure and dynamics," Phys. Rep. **424**, 175–308 (2006).
[5] M. A. Porter and J. P. Gleeson, *Dynamical Systems on Networks*, 1st ed. (Springer International Publishing, 2016).
[6] M. Golubitsky and I. Stewart, "Nonlinear dynamics of networks: The groupoid formalism," Bull. Am. Math. Soc. **43**, 305–365 (2006).
[7] A. Arenas, A. Díaz-Guilera, J. Kurths, Y. Moreno, and C. Zhou, "Synchronization in complex networks," Phys. Rep. **469**, 93–153 (2008).
[8] P. Boldi and S. Vigna, "Fibrations of graphs," Discrete Math. **243**, 21–66 (2002).
[9] B. Ursino, L. V. Gambuzza, V. Latora, and M. Frasca, "Control technique for synchronization of selected nodes in directed networks," IEEE Control Syst. Lett. **3**, 553–558 (2019).
[10] I. Leifer, F. Morone, S. D. S. Reis, J. S. Andrade, M. Sigman, and H. A. Makse, "Circuits with broken symmetries perform core logic computations in genetic networks," PLoS Comput. Biol. **16**, e1007776 (2020).
[11] L. M. Pecora, F. Sorrentino, A. M. Hagerstrom, T. E. Murphy, and R. Roy, "Cluster synchronization and isolated desynchronization in complex networks with symmetries," Nat. Commun. **5**, 4079 (2014).
[12] A. Grothendieck, "Technique de descente et théorémes d'existence en géométrie algébrique, I. généralités. descente par morphismes fidélement plats," Séminaire N. Bourbaki **5**, 299–327 (1960).
[13] I. Leifer, M. Sánchez-Pérez, C. Ishida, and H. A. Makse, "Predicting synchronized gene coexpression patterns from fibration symmetries in gene regulatory networks in bacteria," arXiv:2104.08256 [q-bio.MN] (2021).
[14] E. Nijholt, B. Rink, and J. Sanders, "Graph fibrations and symmetries of network dynamics," J. Differ. Equ. **261**, 4861–4896 (2016).
[15] B. D. McKay, "Practical graph isomorphism," Congresus Numerantium **30**, 45–87 (1981).
[16] H. Kamei and P. J. A. Cock, "Computation of balanced equivalence relations and their lattice for a coupled cell network," SIAM J. Appl. Dyn. Syst. **12**, 352–382 (2013).
[17] I. Stewart, M. Golubitsky, and M. Pivato, "Symmetry groupoids and patterns of synchrony in coupled cell networks," SIAM J. Appl. Dyn. Syst. **2**, 609–646 (2003).
[18] L. DeVille and E. Lerman, "Modular dynamical systems on networks," J. Eur. Math. Soc. **17**, 2977–3013 (2006).
[19] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Design and Analysis of Computer Algorithms* (Addison-Wesley Pub. Co., Reading, MA, 1974).
[20] R. Paige, R. E. Tarjan, and R. Bonic, "A linear time solution to the single function coarsest partition problem," Theoret. Comput. Sci. **40**, 67–84 (1985).

[21]M. Habib, C. Paul, and L. Viennot, "Partition refinement techniques: An interesting algorithmic tool kit," Int. J. Found. Comput. Sci. **10**, 147–170 (1999).
[22]J. W. Aldis, "A polynomial time algorithm to determine maximal balanced equivalence relations," Int. J. Bifurcation Chaos **18**, 407–427 (2008).
[23]I. Belykh and M. Hasler, "Mesoscale and clusters of synchrony in networks of bursting neurons," Chaos **21**, 016106 (2011).
[24]P. Boldi, V. Lonati, M. Santini, and S. Vigna, "Graph fibrations, graph isomorphism, and pagerank," Theoret. Informatics Appl. **40**, 227–253 (2006).
[25]M. Habib and C. Paul, "A survey of the algorithmic aspects of modular decomposition," Comput. Sci. Rev. **4**, 41–59 (2010).
[26]R. Paige and R. E. Tarjan, "Three partition refinement algorithms," SIAM J. Comput. **16**, 973–989 (1987).
[27]A. Cardon and M. Crochemore, "Partitioning a graph in $O(|a| \log_2 |v|)$," Theor. Comput. Sci. **19**, 85–98 (1982).
[28]J. E. Hopcroft, "A $n \log n$ algorithm for minimizing states in finite automaton," in *Theory of Machines and Computations, Proceedings of an International Symposium on the Theory of Machines and Computations, Technion, Haifa, Israel, August 16–19, 1971*, edited by Z. Kohavi and A. Paz (Academic Press, 1971), pp. 189–196.
[29]D. G. Corneil, "Lexicographic breadth first search—A survey," in *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science Vol. 3353, edited by J. Hromkovic, M. Nagi, and B. Westfechtel (Springer, Berlin, 2004), pp. 1–19.
[30]M. Golubitsky, I. Stewart, and A. Török, "Patterns of synchrony in coupled cell networks with multiple arrows," SIAM J. Appl. Dyn. Syst. **4**, 78–100 (2005).
[31]Y. Li and L. Chen, "Big biological data: Challenges and opportunities," Genom. Proteom. Bioinform. **5**, 187–189 (2014).
[32]V. Marx, "The big challenges of big data," Nature **498**, 255–260 (2013).
[33]N. Norris, "Universal covers of graphs: Isomorphism to depth $n-1$ implies isomorphism to all depths," Discrete Appl. Math. **56**, 61–74 (1995).
[34]A. Tarski, "A lattice-theoretical fixpoint theorem and its applications," Pac. J. Math. **5**, 285–309 (1955).
[35]F. Morone and H. A. Makse, "Symmetry group factorization reveals the structure-function relation in the neural connectome of *Caenorhabditis elegans*," Nat. Commun. **10**, 4961 (2019).
[36]See https://github.com/makselab/FastFibration for the Fast Fabrication Partitioning algorithm, together with its documentation for proper use.
[37]See https://github.com/makselab/fibrationSymmetries for the codes for the Kamei-Cock algorithm used in previous works with extended functionalities.