# Project Report: Física Teórica I

•

## Identification and classification of information-processing building blocks on genetic regulatory network

Higor da S. Monteiro

•

Departament of Physics - Universidade Federal do Ceará(UFC)

Fortaleza, Ceará

November, 05, 2019

**Abstract**

Fibration building blocks of information-processing networks represent the sets of nodes that are symmetrical with respect to the processing of input signals received from the rest of the network, defining this way the classes of nodes that process equivalent information. In this report, I reproduced relevant results concerning the identification and classification of the fibration building blocks of directed networks, constructed from real systems data. More specifically, using the transcriptional regulatory network data from the *Escherichia Coli* bacteria, I quantify the clusters of nodes in the network that synchronously process equivalent information, and then I classify these clusters, called fibers, based on its specific topological features. Here, to present the obtained results, I briefly introduce the theory concerning the graph fibration morphism and its main definitions related to information processing symmetries. Then, I detail the computational methods adopted to the identification of the network fibers, presenting the algorithmic approach for the problem in hand, based on the general relational refinement partitioning algorithms described by Paige and Tarjan in 1987. After defining the implementation details of the graph refinement algorithm I apply the method for trial network examples and then for the constructed genetic regulatory network of *Escherichia Coli*, noticing the equivalence between the results obtained by the methods described in this report and the results exhibited in the most recent literature given at *Morone et. al.* (2019). The results showed in this report, concerning the fibers distribution over the network, are tested and verified to support the assertion that the algorithm used captures the correct minimal fibration.

# 1 Introduction

Dynamical processes occurring in networks permit novel qualitative behaviors that are not directly measured in classical dynamical systems [1]. The main feature of these behaviors is the existence of verifiable partial synchronized states, where different elements in the system follow the same or correlated phase space trajectories. These synchronized states are fundamentally important for the study of dynamical phenomena in networks, being still more relevant for information-processing complex systems, in which synchrony is directly related to functionality.

Besides the specific characteristics of the dynamics taking place on the network, it is possible to define states of partial synchrony by considering only the topological features of networks. To this, it is possible to define a natural graph partitioning where each element of it represents the set of nodes that process equivalent incoming information from the rest of the network and even from itself, implying that each of these nodes processes information synchronously. Furthermore, one of the important insights concerning these clusters of synchronized network elements is the direct association of symmetries groups with the resulted distribution of synchronization clusters, which can offer a precise procedure to identify and classify the information-processing building blocks of several networks in nature.

However, recently some works has been recognizing that topological symmetries of the network structure, meaning neighborhood structure invariance under nodes permutation, are not actually necessary for the emergence of synchronization of node states. Thus, instead of very restrictive isomorphisms rules, the blocks of synchronized information processing obeys a more loose concept of symmetry represented by the rules of groupoids. More specifically, information flow in a directed network is maintaneid as invariant under a graph fibration morphism. A graph fibration is a transformation that keeps invariant the set of universal input-trees that group the network nodes into functional blocks called network fibers [2], [3].

The formal definition of an input-tree is explained in detail at [3], [2] and [4]. Here, I give just a direct and intuitive example through the examples showed at figures 1. To fully characterize the information flow directions in a directed network, we can define for each node in the network an input-tree containing all the information pathways that ends at the given node. Thus, if two different nodes have isomorphic input-trees, which means that they receives equivalent information from the network, they synchronize their dynamical states, exhibiting a correlated behavior for the network. Even more, if a group of nodes have isomorphic input-trees, then a graph fibration transformation reduces these nodes

from network $G$ in one single node for a base network $G'$. This way, a fibration reduces the original network in a base network where each node of $G'$ represents a fiber block of $G$. In figure 1 we can see a directed network formed just by three nodes. For each one, they differs only by the patterns of directed links, from which information can flow.

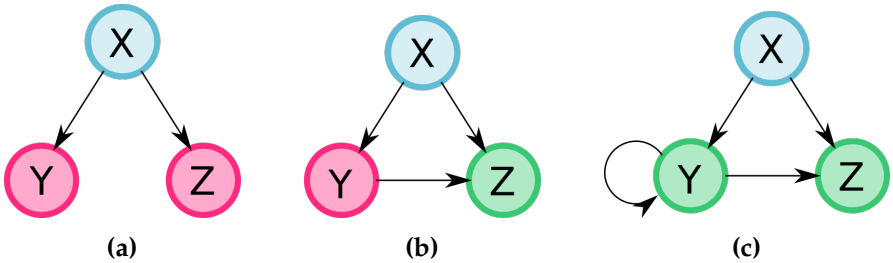



**Figure 1:** First trial network example. Containing a total of $N = 17$ nodes and $M = 22$ arcs, the network has two weakly connected components, defined as the connected networks when we treat all the directed arcs as undirected edges. For each non-trivial fiber I give the same color for the nodes belonging to it.

As we can see in figure 1, in the first case both $Y$ and $Z$ receives information only from $X$, for which it is obvious that $Y$ and $Z$ process the same information and then the two nodes synchronize their states (again not considering the details of the dynamics). However, if we allow that $Y$ sends information to $Z$ along with $X$, then $Z$ do not process the same information than $Y$, since it receives information from $X$ and $Y$, while $Y$ receives only from $X$ (1b). To get around this inconvenience, we can simply add an autorregulation loop for node $Y$, allowing that $Y$ and $Z$ synchronize again (1c).

All this discussion can be put on solid ground by considering the isomorphic relations between the input-trees of each node in the network. It is possible to note in figure (2a) and (2c) that the nodes $Y$ and $Z$ have isomorphic input-trees, which imply that the two nodes have synchronized states. Important to mention that because the addition of the autorregulation loop in the node $Y$, the input-tree that includes this loop becomes of infinite height, meaning the information loop flow in this directed link.

In practice, to identify if two nodes belongs to the same network fiber, *i. e.*, if two nodes have isomorphic input-trees is only necessary to check the first-layer of their input-tree, called input-set [5], which means that if two nodes belong to the same fiber block they have the same number of incoming links from the same fibers nodes. That implies that is not necessary that incoming neighbors be the same nodes, being only necessary that the incoming fibers are the same.

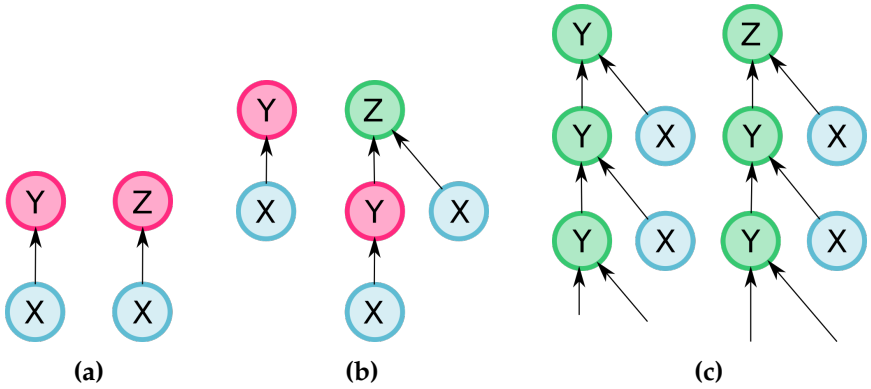



**Figure 2:** First trial network example. Containing a total of $N = 17$ nodes and $M = 22$ arcs, the network has two weakly connected components, defined as the connected networks when we treat all the directed arcs as undirected edges. For each non-trivial fiber I give the same color for the nodes belonging to it.

At last, the identification of network fibration building blocks on biological regulatory networks is important because it allows the recognition of their information-processing building blocks, which is essential for the understanding of the functionalities intrisic to these regulatory network [6]. Moreover, this approach can offer a persistent procedure to the study of biological function in networks comparing with methods based on statistical significance of common occurring subnetworks, like the one presented by the network motifs [7]. The computational approach for a proper identification and classification of fibers in directed networks are explained below.

## 2 *Coarsest Refinement Partitioning* Algorithm

To identify the correct distribution of fibers over the network it is necessary to define an efficient procedure to split the nodes into different disjoint partitions, from which all nodes inside the same partition must receive equivalent information from all other partitions. To do 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. 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. Therefore, in this section, I detail the optimal algorithm used to find this coarsest partitioning in the context of graph fibrations and show the relatively simple implementation of this method.

The algorithm used in this project is a slightly modified version of the algorithm presented by Paige and Tarjan [8], having a runtime complexity of $\mathcal{O}(M \log N)$,

where $M$ and $N$ are, respectively, the number of edges and nodes in the network, being almost linear with the size of the network. It is important to note that this algorithm has the same runtime order than the algorithm from Cardon and Crushemore [9], in which the algorithm of minimal fibration from [3] is based. However, the algorithm presented by Paige and Tarjan has a simpler implementation and smaller prefactors, exhibiting a better approach to our problem. Even though the algorithm has wider general applications, here I give the details of the algorithm implementation to the application for a network, so that I introduce the necessary definitions within this context.

## 2.1 Algorithm description

A network $G(V, E)$ is completely defined by the sets of the nodes elements $V$ and the set of the arcs $E$. The network has $N = |V|$ nodes that can have connections with one another defined by the set $E$, which contains $M = |E|$ ordered pair of nodes, denoting the directed links between the network nodes. Considering a network, we can define a graph partition $\bar{P}$ over $V$ as a set of pairwise disjoints subsets of $V$ whose union is all $V$, that is

$$\bar{P} = \bigcup_i P_i \tag{1}$$

where $P_i$ are the elements of the partition $\bar{P}$, called blocks. If we take an additional graph partition $\bar{Q}$ that has the property that all of its blocks are contained in a block of $\bar{P}$, we say that $\bar{Q}$ is a refinement partition of $\bar{P}$. Moreover, for a block $B \in \bar{P}$, we say that the block $B$ is stable with respect a set $S$ if either all elements of $B$ connects with an element of $S$ or none element of $B$ points to any element of $S$.

The stable partition concept is central to the problem of relational coarsest refinement partitioning and can be easily extended to graph problems, where the coarsest graph partition problem is that of finding, for a given set of directed links $E$ and an initial partition $\bar{P}$ over $V$, the coarsest stable refinement partition of $\bar{P}$, *i. e.*, the minimal number of disjoint blocks subsets of $V$ that forms a stable refinement of $\bar{P}$.

Considering the stability of partitions, for a proper identification of the network fibers as defined in the above section we ought to construct a stable graph partition that is equivalent to the coarsest refinement of the network (minimal number of blocks) with respect to the information processing of each node. For that goal, we need to extend the concept of stability over partitions for the case that accounts for the information processed by each block of the partition. Therefore, to identify the

group of nodes with isomorphic input-tree, we require that the partition should be not only stable but **input-tree stable**. That means that for a subset $S \subseteq V$, a graph partition $\bar{P}$ over the network $G(V, E)$ is input-tree stable with respect to $S$ if for all the block $B \in \bar{P}$, the following equality is satisfied for all the elements $x, y \in B$:

$$|E^{-1}(\{x\}) \cap S| = |E^{-1}(\{y\}) \cap S| \tag{2}$$

where $E(\{x\})$ and $E^{-1}(\{x\})$ represents, respectively, the set of nodes that directly receives information from $x$, and the set of nodes that sends information, via a directed link, to $x$.

This way, we can benefit from the stability properties [8] to construct a refinement algorithm step that can achieve, through a finite number of steps, a input-tree stable partition from an initial unstable partitioning of the network. This way, given a subset $S \subseteq V$, the refinement step has the effect to refine the current partition, input-tree unstable with respect to $S$, by replacing it for a new partition, now input-tree stable for $S$. With that objective, we define a split function *I-split($S$, $\bar{P}$)* that receive as input the current partition and a set $S$ and returns as output a new input-tree stable partition with respect to $S$. Favorably, that function benefits from two major properties of stability (and input-tree stability): the stability inheritance by refinement and by union of sets. By reason of this refinement inheritance, a given set $S$ can be used only once by the function *I-split*, guaranting that the partition, after all other refinement steps, will maintain as stable with respect to $S$. Moreover, since stability is inherited under union of sets $S$, after sets are used in *I-split*, their union cannot be used for the function. Considering all these properties, the essence of the refinement algorithm can be stated as

> **Refinement Algorithm**: Find a set $S$ in which the current partition $\bar{P}$ is input-tree unstable. Replace $\bar{P}$ by the output of *I-split($S$,$\bar{P}$)*. Guarantees that the set $S$ or unions of used sets never be used again.

Since the finest partitioning possible is the one in which every node is itself a block, the refinement step may be proceeded at most $N - 1$ times, guaranting that the algorithm terminates with the correct answer, since stability is inherited by the refinement process. However, to guarantee that the algorithm has an optimal runtime we have to find a efficient way to select the appropriate sets to the refinement step, without choosing repeated ones. Fortunately, this can be easily done for the construction of a input-tree stable partition.

Given a set $S$ of nodes from the network $G(V, E)$ and a given input-tree unstable graph partition $\bar{P}$, the blocks $B \in \bar{P}$ that are input-tree unstable with respect to

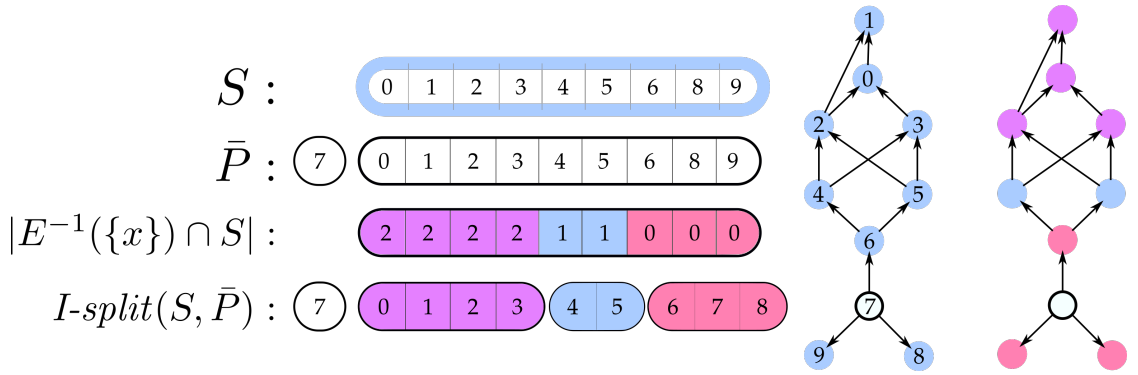



**Figure 3:** A refinement step for the split function. Receiving the input partition $\bar{P}$ and a refinement set $S$, we identify the input-tree unstable blocks and split them into input-tree blocks with respect to $S$. Due to the refinement inheritance stability property, refining the resulted partition $\bar{P}$ with respect to other sets $S'$, that are not equal to $S$ or a union of already used sets $S$, guarantees the input-tree stability to $S$ during all the following algorithm steps.

$S$ can be splitted into several blocks $B_j$ that will be stable input-tree with respect to $S$. Then, each splitted block will have the property defined by

$$B_j = \{x \in B : |E^{-1}(\{x\}) \cap S| = j\} \tag{3}$$

where the number of splitted block must be larger than one to a proper splitting process take place. Then, for each unstable block $B \in \bar{P}$, all the splitted block, except the largest one, can be put at the back of a queue to be used ahead in the algorithm as a refinement set. This ensures that none repeated sets or union of repeated sets can be used during the algorithm execution.

Finally, the complete algorithm to find the correct network fibers of a network $G(V, E)$ consists in initializing a graph partition $\bar{P}$ over all nodes from $V$ except the nodes that do not receive any information from other node, in which each one of these will be defined as isolated fibers already in the beginning of the algorithm. Besides these solitaire nodes, we also identify the nodes that receive information only from themselves and put them in the queue as refinement sets, without isolating them from the main partition $\bar{P}$. This way, the partition $\bar{P}$ is defined as one block containing all the nodes that receive information, even if only from themselves. The algorithm maintains a queue $L$ of possible refinement sets, initially containing the single block of $\bar{P}$ and all the isolated blocks defined at the beginning. Then, we proceed as

**Coarsest Refinement Partitioning Algorithm**: Remove from $L$ its first set $S$. Replace $\bar{P}$ by the *I-split*$(S, \bar{P})$. Whenever a block $B \in \bar{P}$

**Algorithm 1:** *I-split* $(S, \bar{Q}, L)$

**Input** : A set $S$ and a partition $\bar{Q}$

**Output**: Input tree stable partition $\bar{Q}$ with respect to $S$

Initialize $\bar{U}$ as an empty partition;
**for** $\forall B \in \bar{Q}$ **do**
  **if** $\exists \{w_i, w_j\} \subseteq B : |E^{-1}(\{w_i\}) \cap S| \neq |E^{-1}(\{w_j\}) \cap S|$ **then**
    *push* $B$ to $\bar{U}$;
    Initialize $\bar{T}$ as an empty partition;
    **for** $\forall w_i \in B$ **do**
      **if** $\exists X \in \bar{T} : |E^{-1}(\{w_i\}) \cap S| = X(E)$ **then**
        insert $w_i$ in $X$;
      **else**
        create a new block $X$;
        insert $w_i$ in $X$;
        $X(E) \leftarrow |E^{-1}(\{w_i\}) \cap S|$;
        push block $X$ to $\bar{T}$
      **end**
    **end**
  **end**
  *enqueue* all blocks $X \in \bar{T}$ in $L$, except the largest one;
  *copy* all blocks $X \in \bar{T}$ to $\bar{Q}$;
**end**
*delete* all blocks $B \in \bar{U}$ in $\bar{Q}$

splits into two or more nonempty blocks, add all except the largest to the back of $L$.

And this process is repeated until the queue $L$ is empty. At this point, the resulted partition $\bar{P}$ represents the coarsest input-tree stable partitioning of the network $G(V, E)$, where each block represents a network fiber with all its nodes having isomorphic input-trees. The figure 3 displays the splitting operation at the beginning of the refinement algorithm for a small network example.

## 2.2 Data preparation and algorithm implementation

Given the description above, next, I apply the above algorithm for the genetic regulatory network of the *Escherichia Coli* bacteria. The obtention of this genetic network is through the transcriptional regulatory interaction data, where each

gene is regulated by a transcription factor protein. Since a transcription factor in the cell is produced by a gene, it is possible to define directed relations between two genes if a gene is responsible for the production of a transcriptional factor, which regulates the other gene. Considering that a transcription factor can be either an activator(positive) or repressor(negative), or even behaves as both(dual), the directed links between genes can carry different types of information. Because of that, the partitioning algorithm must account for the type of message signals to construct appropriate stable input-trees for the network fibers. Therefore, for a proper application of the algorithm on the *Escherichia Coli* genetic network, we label each node uniquely, either as numbers or string names, and also we do the same for each link with the type of connection between two genes.

Given the regulatory directed network $G(V, E)$ to be partitioned in fibers, it is very important for the correctness of the algorithm that the nodes that do not receive any information, that is, the nodes that do not have any inward connections, be preprocessing as isolated blocks in a different partition. This means that the initial graph partition $\bar{P}$ must be divided in two different partitions $\bar{P}'$ e $\bar{P}''$, where the first one should contains all the single-node blocks containing the solitaire nodes $v$ for which $|E^{-1}(\{v\})| = 0$, and the second contains at the beginning of the algorithm a single block containing all the other nodes $w$ in which $|E^{-1}(\{w\})| \geq 1$. The importance of this preprocessing is to guarantee that solitaire nodes do not be put on the same fibers during the refinement steps, since for any subset $S \subseteq V$, the solitaire nodes satisfies the equality $|E^{-1}(\{v\}) \cap S| = 0$. Furthermore, even though the blocks of $\bar{P}'$ are used as refinement sets, $\bar{P}'$ is not used by the split function in any step. This way, the final graph partition of $G(V, E)$ is the union of $\bar{P}'$ and the result of the refinement partitioning of $\bar{P}''$, that is, the coarsest or minimal input-tree stable partition of $G$.

After preprocessing the solitaire nodes, we have to define the data structures for the partition $\bar{Q}$ and its blocks $B$. We should define the implementation of a partition by a doubly-linked list of blocks, which allows that deletion of blocks be done in constante time $O(1)$ as long we have the block memory address during the procedure. A block is just a structure record containing indexing data along with a doubly-linked list containing all the nodes that belongs to it. Together with the queue of blocks $L$, these constructions are the main data structures necessary for an efficient implementation of the refinement partitioning algorithm. At the beginning of the algorithm, we enqueue all the blocks of $\bar{P}'$ e $\bar{P}''$ to $L$ (order is not important for the output correctness), where the nodes that receives information only from themselves, $E^{-1}(\{v\}) = \{v\}$, are pushed to $L$ as single-node refinement blocks. Then, we start the algorithm initializing a partition $\bar{Q} = \bar{P}''$ and by

**Algorithm 2:** Coarsest Refinement Graph Partitioning

**Input :** A network $G(V, E)$

**Output:** The minimal fibration partition $\bar{Q}$ of $G(V, E)$

1 Initialize $S$ as an empty set;
2 Initialize $L$ as an empty queue;
3 Initialize $\bar{P}'$, $\bar{P}''$, $\bar{Q}$ as empty partitions;
4 $\forall w_i \in V \Rightarrow B'_i = \{w_i : |E^{-1}(\{w_i\})| = 0\}$;
5 $\forall v_i \in V \Rightarrow A_i = \{v_i : E^{-1}(\{v_i\}) = \{v_i\}\}$;
6 $B'' = \{v \in V : |E^{-1}(\{v\})| \geq 1\}$;
7 *push* all $B'_i$ to $\bar{P}'$;
8 *push* $B''$ to $\bar{P}''$;
9 *enqueue* $B'' \in \bar{P}''$ to $L$;
10 *enqueue* all $A_i$ to $L$;
11 *enqueue* all blocks $B' \in \bar{P}'$ to $L$;
12 $\bar{Q} \leftarrow \bar{P}''$;
13 **while** $|L| \neq 0$ **do**
14 $\quad S \leftarrow$ *dequeue*$(L)$;
15 $\quad \bar{Q} \leftarrow$ *I-input*$(S, \bar{Q}, L)$
16 **end**

removing the first element set $S$ of $L$, to apply the *I-split*$(S, \bar{Q})$ function to identify the input-tree unstable blocks of $\bar{Q}$ and to split them into input-tree stable blocks with respect to $S$. After that, all the splitted blocks are pushed to the end of $L$, with exception the largest resulted blocks for each splitted block. As we have mentioned, the algorithm terminates when there is no more sets in $L$. This is the whole algorithm, even though the implementation of the *I-split* function might not be so straightforward, since a given block $B$ can be splitted into an arbitrary number of blocks of different sizes. Considering this, I propose in the pseudocode 1 an efficient implementation construction to the splitting function.

In the algorithm 1, our *I-split* function receives a set $S$ and the partition $\bar{Q}$ as input and returns a input-tree stable $\bar{Q}$ with respect to $S$ as output. Besides its list of nodes, a block $X$ should have another attribute acessed as $X(E)$, which receives, during the splitting process, the number of inward links coming from the current refinement set $S$. This way, a node $w$ belonging to the block splitted can be put in the block $X \in \bar{T}$ that has the same attribute value, like stated in the conditional expression at line 7 of the algorithm 1. In cases where the type of connections are non-trivial, i. e., there is more than one type of message signal,

the attribute $X(E)$ can be replaced by a list $X(E(i))$ for each type $i$ of message presented in the network. For that situation, the conditional line 7 must be verified for all types of connnections: $\exists\, X \in \bar{T} : |E_k^{-1}(\{w_i\}) \cap S| = X(E(k))$.

Finally, considering all the discussion and implementation presented above, we can explicit the complete algorithm in just a few basic steps in algorithm 2.

A concrete implementation of the algorithm in a programming language is presented at the link `https://github.com/higorsmonteiro/fiberblocks` together with *E. Coli* prepared data and another trial network examples. Further information about the application of the codes in other genetic regulatory network data can be found at the same link given.

## 2.3 Input-tree branching ratio

After the proper identification of the fibers over the network, we should classify each one in accordance with two fundamental numbers: the fundamental class number $n$ and the subclass number $\ell$. The subclass value is easily obtained, since it represents the number of external nodes that directly regulates the fiber (all the fiber nodes). However, the fundamental class, or input-tree branching ratio, may be not so straightforward to calculate by a direct fiber neighborhood verification, since it involves the quantification of cycles of information that leaves a fiber and goes back to it. Fortunately, the value of $n$ can be obtained if one chooses an eigenvalue decomposition algorithm of the adjacency matrix $\mathbf{A}$ characteristic of the appropriate strongly connected component that includes the fiber. This way, $n$ is given by the largest eigenvalue $\lambda_{max}$ of the subnetwork containing the loop of the information coming out of the fiber. At the same time, $n$ is also the branching ratio of the fiber input-tree, where the ratio is defined by the number of nodes $Q_{j+1}$ belonging to a given layer level $j+1$ of the input-tree and the number of nodes $Q_j$ at the previous layer level $j$ for large enough $j \to \infty$. By this, the branching ratio is defined as

$$n = \lim_{j\to\infty} \left( \frac{Q_{j+1}}{Q_j} \right) \tag{4}$$

where $n$ can be either an integer branching ratio or a golden fibonacci ratio $\varphi_d$ resulted by external loop pathways with size $d \geq 2$.

Concerning the definition above, an alternative approach to obtain the fundamental class number is by a direct construction of the input-tree fiber until a reasonable depth is reached. Here I propose a simple method to calculate the value of $n$ by counting the number of nodes $Q_j$ in each layer level $j$ of the input-tree of a given fiber node. For infinite input-trees, *i. e.*, fibers containing inside or external loops, we use the fast convergence for both the integer and fibonacci ratios to let

the limit depth of the input-tree fixed by a value $\delta$. For integer branching ratio, the value of $n$ is obtained exactly even for a small number of layers, while for fractal golden ratio the rate of convergence of $n$ to $\varphi_d$ is exponential, requiring also small number of layers for reasonable values of $\varphi_d$. The aim of the algorithm is, for each step, to build two following layers of the input-tree to calculate directly the value of $n$ by counting the number of nodes $Q_j$ and $Q_{j+1}$ presented in each of these layers. We define $n_j$ as the ratio $\frac{Q_{j+1}}{Q_j}$, and the algorithm stop when the the layer $j+1$ is equal to $\delta$.

The algorithm consists of two steps, the first one being the verification of the existence of loops on the input-tree, from which only if it is true the algorithm goes to the second step. The existence of loops in the input-tree implies an infinite tree. For this, there are two cases possible: inside cycles of information, implying integer branching ratio $n$, and fibonacci cycles of information that leaves the fiber and then returns to it. This can be easily verified by identifying the strongly connected components that includes nodes of the fiber. If the strongly connected component contains only fiber nodes, then the branching is integer, otherwise is fractal. The constrast between these two cases is important for the correct construction of the input-tree, since for fibonacci cycles the algorithm must guarantee that only the nodes $w \in \Lambda$ belonging to the shortest cycle path and, of course, the nodes of the fiber will be placed on the input-tree, thus guaranting the correct $\varphi_d$ value.

The implementation of the algorithm is quite simple, making use of only stacks to store the input-tree layers. Starting from the root node $v$ of the input-tree of the fiber $V_i$ and defined $\Lambda$ for the case of fibonacci cycles, we find its nearest incoming neighbors $E^{-1}(\{v\}) \in \Lambda$ and $E^{-1}(\{v\}) \in V_i$ and store them in a first stack $S_1$, representing this way the second layer $j = 1$ of the input-tree, from which we can obtain the value of its size $Q_1$. Next, we pop each element of $S_1$ and get their incoming neighbors $E^{-1}(\{v \leftarrow pop(S_1)\})$ and store them in $S_2$ until $S_1$ is empty. Then, $S_2$ will store the second layer $j = 2$ of the input-tree, from which $Q_2$ is obtained, allowing the calculation of $n_2$. Since at this time $S_1$ is empty, the incoming neighbors of each element of $S_2$ are stored in $S_1$ until $S_2$ is empty, and this process is repeated, exchanging $S_1$ e $S_2$ as incoming neighbors supplier and deposit in each loop step. This repeating exchange can be implemented with the help of stacks pointers to represent, in each algorithm loop, the supplier and deposit stacks. This way, we can calculate the values of $n_j$ until the limit depth $j + 1 > \delta$ is reached. The algorithm pseudocode implementation is given in 3.

The algorithm 3 assumes that the given node $v$ is part of a strongly connected component, even for the trivial case containing only $v$ connected to itself, because this warrant that $n \neq 0$. This can be easily verified implementing a Depth-First

Search or Breadth-First Search on the network and its tranpose starting at node $v$ to find its strongly connected component, which takes linear time with the size of the strongly connected component. Then, for components containing only fiber nodes the algorithm obtain the correct integer $n$ for small $\delta$ limit tree depth, running linearly with $|E^{-1}(\{V_{SCC}\}) \in \Lambda|$. For component containing outsider nodes, then we define $\Lambda$ containing the nodes in the shortest loop **??** and apply the above algorithm. Of course, for fibonacci loops, determining the size $s$ of the shortest loop already gives indirectly the information $\varphi_s$ needed, being necessary only to define previously a table with the relations $s \to \varphi_s$.

**Algorithm 3:** Input-tree branching ratio determination

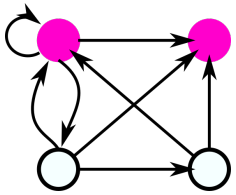
**Input** : A node $v \in V$ for a network $G(V, E)$

**Output**: $n$ representing the branching ratio of the node $v$ input-tree

Initialize $\mathbf{s_1}$ and $\mathbf{s_2}$ as empty stacks;
Initialize **Sup**, **Depot** and **Aux** as stack pointers;
$j \leftarrow 0$;
push($\mathbf{s_1}$, $v$);
$\mathbf{Sup} \leftarrow \mathbf{s_1}$;
$\mathbf{Depot} \leftarrow \mathbf{s_2}$;
**while** $j + 1 \leq \delta$ **do**
    $a_j \leftarrow |\mathbf{Sup}|$;
    **while** $|\mathbf{Sup}| \neq 0$ **do**
        $u \leftarrow$ pop(**Sup**);
        **if** $w \in \Lambda : \forall w \in E^{-1}(\{u\})$ **then**
            push(**Depot**, $w$);
        **end**
    **end**
    $a_{j+1} \leftarrow |\mathbf{Depot}|$;
    $n_{j+1} \leftarrow \left(\frac{a_{j+1}}{a_j}\right)$;
    $\mathbf{Aux} \leftarrow \mathbf{Depot}$;
    $\mathbf{Depot} \leftarrow \mathbf{Sup}$;
    $\mathbf{Sup} \leftarrow \mathbf{Aux}$;
    $j \leftarrow j + 1$;
**end**

The approach given represents an alternative and direct method to obtain the fundamental class number $n$ and, even though not the most convenient, can improve the runtime of the classification comparing with spectral decomposition

methods, for which the fastest algorithm runs in quadratic time with the size of the subnetwork used. However, I am aware that the improvements may not affect significantly the perfomance of the algorithm, since the average size of the subnetworks used for the calculations are generally very small.



## Results

Using the algorithm above, I applied it to a set of trial networks examples before its proper application to the *Escherechia Coli* genetic regulatory network data. The reason for that is merely to show that the not just the algorithmic approach chosen is consistent but the written code per se is correctly implemented. For each network, I defined a adjacency list file containing all the directed connections between the nodes/genes and for each arc I have a string name defining the type of regulation of that specific link, where here, considering the genetic regulation context, the string types can be "positive", "negative" or "dual". The three trial examples are all small networks, the largest one containing a total of $48$ nodes and $77$ arcs, so that all can be used to test the correctness of the algorithmic approach applied. For these three examples we show the network drawing representation and their fibers distribution identified as the result of the refiniment partition.

The first example is showed at the figure 4. This network is disconnected, containing two main weakly connected components, and in addition contains only positive regulation for the interactions between the genes, showing the simpler case where the connections between nodes in a network are all of the same type. In this case, we find five non-trivial fibers, that is, fibers with size larger than one.

At the end of the algorithm, all the identified fibers are input-tree stable with respect to all fibers in the network, including the fiber itself. Then, by labelling each fiber with an index or color we can then construct the fibration building blocks through the calculation of the fundamental class number $n$ and through the subclass number $\ell$. Again, a fibration building block is defined as the nodes belonging to a fiber and all the $\ell$ external nodes that directly regulates all nodes of the current fiber. With that, I calculated the fundamental class number $n_k$ of the fiber $k$ by applying the algorithm described in 3 over the strongly connected component of the fiber node that sends information to at least one of its regulators.

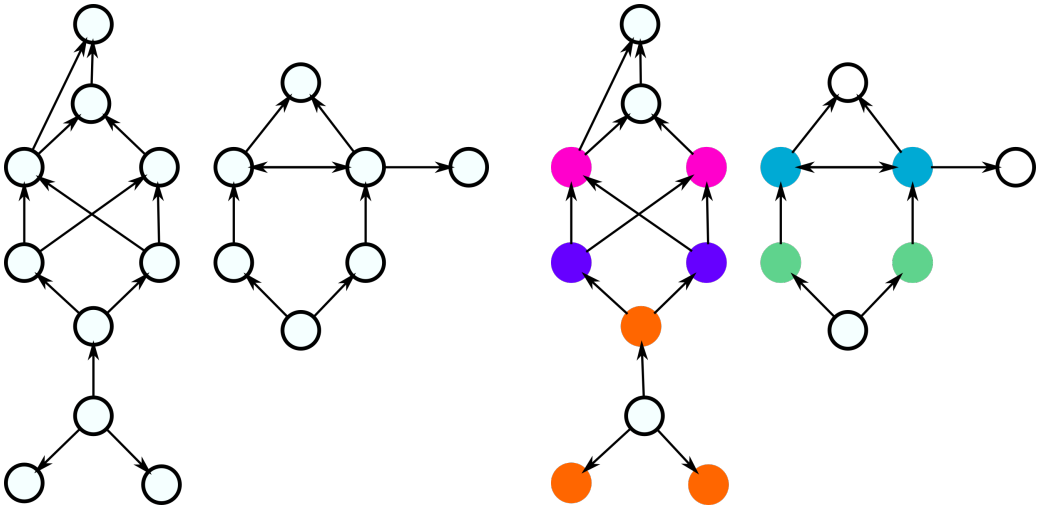
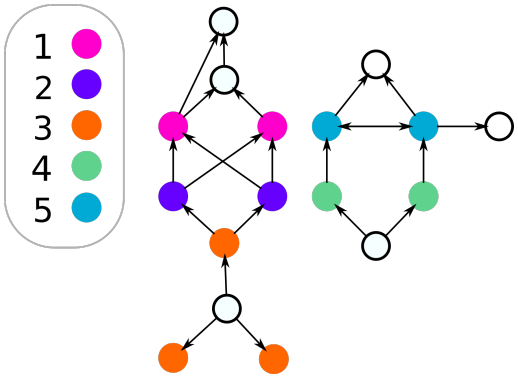



**Figure 4:** First trial network example. Containing a total of $N = 17$ nodes and $M = 22$ arcs, the network has two weakly connected components, defined as the connected networks when we treat all the directed arcs as undirected edges. For each non-trivial fiber I give the same color for the nodes belonging to it.

The value of $n$ quantifies the information cycles of the fibration building blocks and it can be either represent integer branching ratios or fractal golden ration of their input-tree topology class.

Labelling the resulted fiber distribution for the first example in figure 4, we obtain the following fibration classification for each fibration building block of the network containing more than one node:



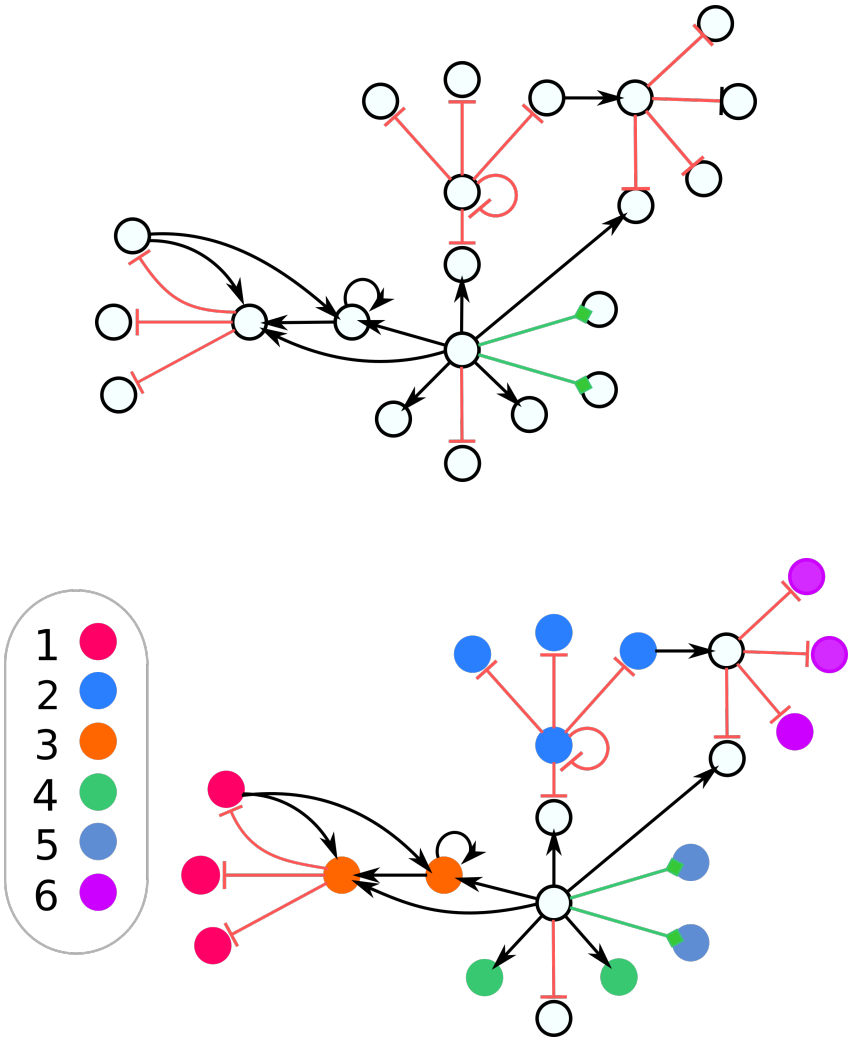

| Fiber | $\|n, \ell\rangle$ |
|---|---|
| 1 | $\|0, 2\rangle$ |
| 2 | $\|0, 1\rangle$ |
| 3 | $\|0, 1\rangle$ |
| 4 | $\|0, 1\rangle$ |
| 5 | $\|0, 2\rangle$ |

**Figure 5:** A table beside a figure

The second example exhibits a connected component containing $N = 21$ nodes, where all the three type of regulation are present. For this network I find the total of eleven fibers, where only six are fibers containing more than one node in it.

Important to note that for two arbitrary nodes be in the same fiber they must not just receive the same number of inward connections but the same number of each type of inward connections, receiving, this way, equivalent information from the rest of the network. We can see this for the example of the fibers $4$ e $5$, where even though both receives two links from the same central node, the fibers receives different types of information from that node, meaning that they are not equivalent fibers.

If we consider the fiber $3$, besides the inner autoregulation loop we have a loop information going outside the fiber and going back through an external regulator inside the fiber $1$. In cases like that one the branching ratio of the fiber has a fractal value given by a golden branch ratio $\varphi_d$ resulted by a cycle of information represented by a fibonacci sequence. This way, fiber $3$ can be classified by the values $|\varphi_d, \ell = 2\rangle$ where the fiber has two external regulator nodes. The fibration classification for the non-trivial fibers is showed in the table 2.



| Fiber | $\|n, \ell\rangle$ |
|---|---|
| 1 | $\|0, 1\rangle$ |
| 2 | $\|0, 1\rangle$ |
| 3 | $\|0, 1\rangle$ |
| 4 | $\|0, 1\rangle$ |
| 5 | $\|0, 2\rangle$ |
| 6 | $\|0, 2\rangle$ |

**Table 2**

Before applying the algorithm for the whole *Escherichia Coli* network, I chose, for the finality of visualization and testing, to apply first the algorithm for a weakly connected component of the *E. Coli* network. The same network is showed at the supplementary information of [3]. The fiber distribution found is consistent with the one presented at [3], presenting the total of fifteen non-trivial fiber groups as shown in the figure 7.

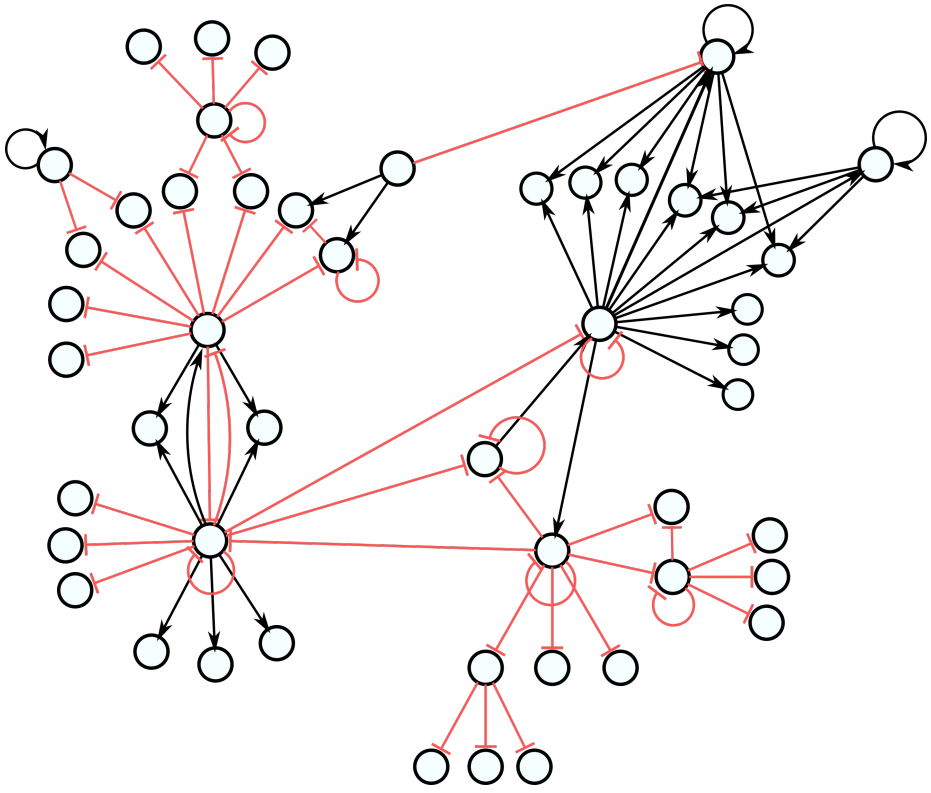


**Figure 6**

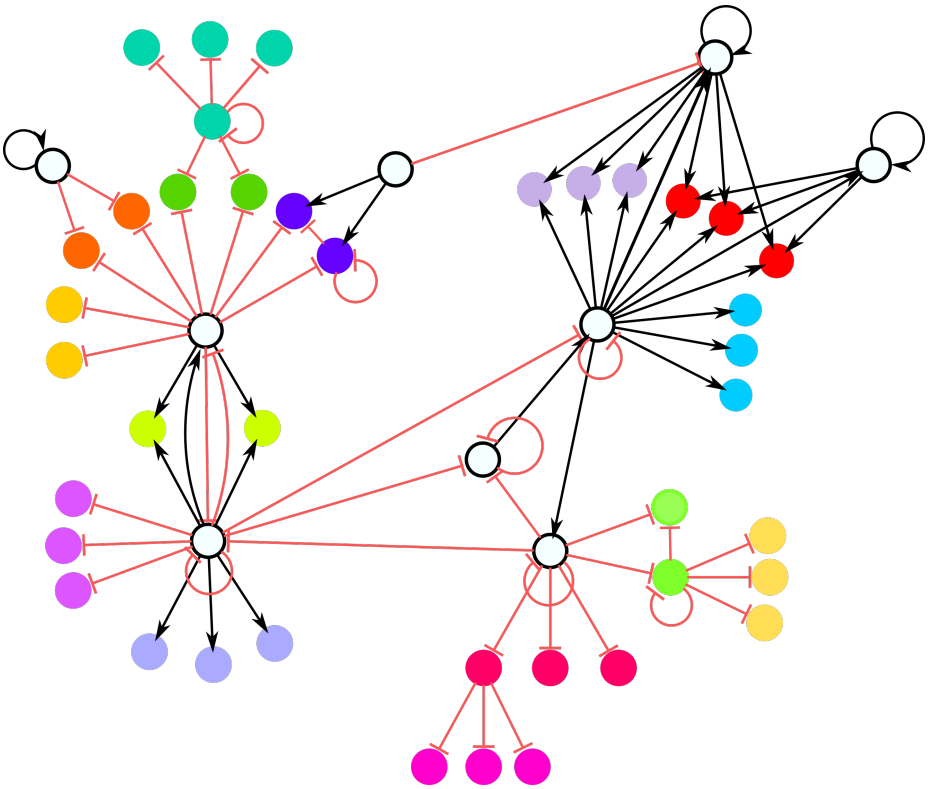


**Figure 7**

The connected component showed at the figure 6 is a subset representation of the nodes of the regulatory network of the *Escherichia Coli* bacteria and the resulted fibers shows just a portion of the whole fibers pattern. In the table X I provide the classification for each fibration building block obtained for the whole bacteria network, including, of course, the fibers presented in the figure 7. In total, I have found $92$ non-trivial fiber groups containing a total of $416$ nodes, obtaining a characteristic fiber statistics for the *Escherichia Coli* network data. To guarantee the correctness of the fiber distribution found I calculated, after the refinement algorithm, the information received for each node inside the fibers and compares if each one process equivalent information. The result of this calculation is stored in the data file given at `https://github.com/higorsmonteiro/fiberblocks/blob/master/Data/fiberVerification.dat`, where for each node in a non-trivial fiber block I give the list of the nodes that sends information messages to the current node along with their fiber indexes and type of link values. Thus, the results of this report show an equivalent fiber distribution over the network comparing with results exhibited at [3] for the same regulatory network data. Even though the table IV an V of [3] show a total number of $91$ fibers and the total number of $414$ of nodes inside non-trivial fibers, the authors spreadsheet reports the total of $416$ nodes, a mismatch which is caused in this report by the presence

of a additional fiber of size two and classified as $|1.6181..., 7\rangle$ containing the genes *marR* e *marA*. This way, the number of nodes in fibers and the genes inside each one are consistent with the results showed in [3].

The statistics concerning the fundamental class number and the external regulators of each fibration block on the network is given in the table 3, showing the additional fiber not reported in [3].

| Structure Fiber | Amount in E. Coli |
|---|---|
| $\|n = 0, \ell = 1\rangle$ | 45 |
| $\|n = 0, \ell = 2\rangle$ | 13 |
| $\|n = 0, \ell = 3\rangle$ | 3 |
| $\|n = 1, \ell = 0\rangle$ | 13 |
| $\|n = 1, \ell = 1\rangle$ | 8 |
| $\|n = 2, \ell = 1\rangle$ | 3 |
| $\|n = 2, \ell = 1\rangle$ | 1 |
| $\|n = 1.3802..., \ell = 1\rangle$ | 1 |
| $\|n = 1.4655..., \ell = 1\rangle$ | 1 |
| $\|n = 1.6181..., \ell = 2\rangle$ | 1 |
| $\|n = 1.6181..., \ell = 7\rangle$ | 1 |
| **Total number of building blocks** | 92 |
| **Total number of nodes inside fibers** | 416 |

**Table 3:** tab:fiberstats

# References


**1.** M. Golubitsky and I. Stewart. Nonlinear dynamics of networks: the groupoid formalism. *Bulletin of the American Mathematical Society*, 46:305–365, 2006.

**2.** P. Boldi, V. Lonati, M. Santini, and S. Vigna. Graph fibrations, graph isomorphism, and pagerank. *Theoretical Informatics and Applications*, 40:227–253, 2006.

**3.** F. Morone, I. Leifer, and H. A. Makse. Fibration building blocks of information-processing networks. *(To be published)*, 2019.

**4.** P. Boldi, , and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243:21–66, 2002.

**5.** N. Norris. Universal covers of graphs: Isomorphism to depth n-1 implies isomorphism to all depths. *Discrete Applied Mathematics*, 56, 1995.

**6.** F. Morone, I. Leifer, S. D. S. Reis, J. S. Andrade, M. Sigman, and H. A. Makse. Circuits with broken symmetries perform core logic computations in genetic networks. *(To be published)*, 2019.

**7.** R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298, 2002.

**8.** R. Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16:973–989, 1987.

**9.** A. Cardon and M. Cruchemore. Partitioning a graph in $o(|a| \log_2 |v|)$. *Theoretical Computer Science*, 19:85–98, 1982.