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

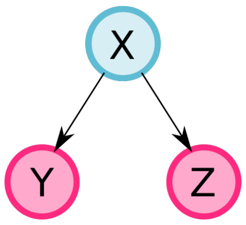
October, 25, 2019

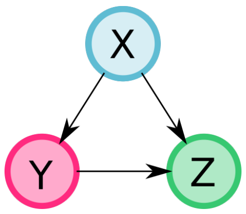
Abstract

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 consistently 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 a slightly different 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 lastly apply the method for trial network examples and then for the constructed genetic regulatory network of *Escherichia Coli*, noticing a visible discrepancy 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 Brief Introduction

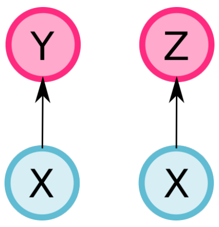
This statement requires citation [1]; this one does too [2] [3] [4] [5]. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean dictum lacus sem, ut varius ante dignissim ac. Sed a mi quis lectus feugiat aliquam. Nunc sed vulputate velit. Sed commodo metus vel felis semper, quis rutrum odio vulputate. Donec a elit porttitor, facilisis nisl sit amet, dignissim arcu. Vivamus accumsan pellentesque nulla at euismod. Duis porta rutrum sem, eu facilisis mi varius sed. Suspendisse potenti. Mauris rhoncus neque nisi, ut laoreet augue pretium luctus. Vestibulum sit amet luctus sem, luctus ultrices leo. Aenean vitae sem leo.

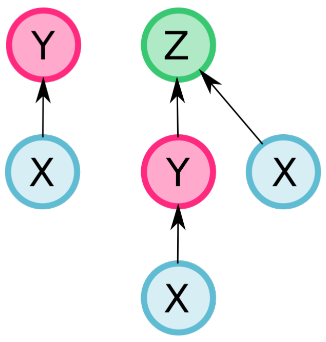




Nullam semper quam at ante convallis posuere. Ut faucibus tellus ac massa luctus consectetur. Nulla pellentesque tortor et aliquam vehicula. Maecenas imperdiet euismod enim ut pharetra. Suspendisse pulvinar sapien vitae placerat pellentesque. Nulla facilisi. Aenean vitae nunc venenatis, vehicula neque in, congue ligula.

Pellentesque quis neque fringilla, varius ligula quis, malesuada dolor. Aenean malesuada urna porta, condimentum nisl sed, scelerisque nisi. Suspendisse ac orci quis massa porta dignissim. Morbi sollicitudin, felis eget tristique laoreet, ante lacus pretium lacus, nec ornare sem lorem a velit. Pellentesque eu erat congue, ullamcorper ante ut, tristique turpis. Nam sodales mi sed nisl tincidunt vestibulum. Interdum et malesuada fames ac ante ipsum primis in faucibus.





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 [4], 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 [2], in which the algorithm of minimal fibration from [1] 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 the 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. Considering this, 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 with 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 stability 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 connections E and an initial partition $\bar{P}$ over V, the coarsest stable refinement 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 direct link, to x.

This way, we can benefit from the stability properties [4] 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 guarantees that the algorithm has a 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 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 exeecution.

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 information from any other node, in which each one of these will be defined as isolated fibers already in the beginning of the algorithm. The partition $\bar{P}$ is defined as one block containing all the other nodes in the network. 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

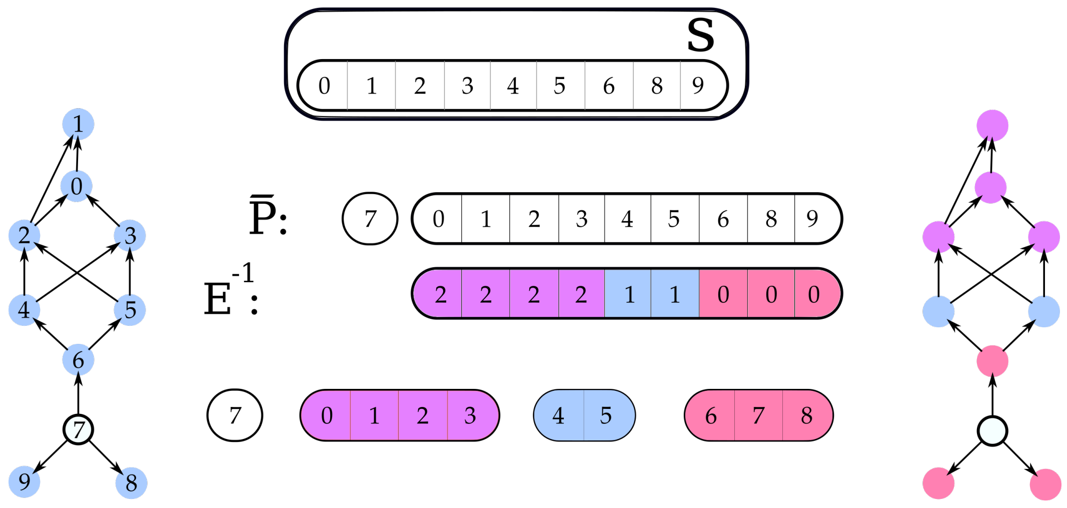


Figure 1: A refinement step for the split function. Receiving the input partition $\bar{P}$ and a refinement set S, we identify the input-tree unstable blocks and split them into input-tree blocks with respect to S. Due to the refinement inheritance stability property, refining the resulted partition $\bar{P}$ 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.

> **Coarsest Refinement Algorithm**: Remove from L its first set S. Replace $\bar{P}$ by the *I-split*(S, $\bar{P}$). Whenever a block $B \in \bar{P}$ splits into two or more nonempty blocks, add all but 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 1 displays the splitting operation at the beginning of the refinement algorithm for a small network example.

2.2 Data preparation and algorithm implementation

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). Then, we start the algorithm initializing a partition $\bar{Q} = \bar{P}''$ and by removing the first element set S of L, and we 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,

```
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 $S$
1  Initialize $\bar{U}$ as an empty partition;
2  for $\forall B \in \bar{Q}$ do
3      if $\exists \{w_i, w_j\} \subseteq B : |E^{-1}(\{w_i\}) \cap S| \neq |E^{-1}(\{w_j\}) \cap S|$ then
4          push $B$ to $\bar{U}$;
5          Initialize $\bar{T}$ as an empty partition;
6          for $\forall w_i \in B$ do
7              if $\exists X \in \bar{T} : |E^{-1}(\{w_i\}) \cap S| = X(E)$ then
8                  insert $w_i$ in $X$;
9              else
10                 create a new block $X$;
11                 insert $w_i$ in $X$;
12                 $X(E) \leftarrow |E^{-1}(\{w_i\}) \cap S|$;
13                 push block $X$ to $\bar{T}$
14             end
15         end
16     end
17     enqueue all blocks $X \in \bar{T}$ in $L$, except the largest one;
18     copy all blocks $X \in \bar{T}$ to $\bar{Q}$;
19 end
20 delete all blocks $B \in \bar{U}$ in $\bar{Q}$
```

I propose in the pseudocode 1 an efficient implementation construction to the splitting function.

In the algorithm 1 above, 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.

Algorithm 2: Coarsest Refinement Graph Partitioning

```
   Input: A network $G(V, E)$
1  Initialize $S$ as an empty set;
2  Initialize $L$ as an empty queue;
3  Initialize $\bar{P}$, $\bar{P}'$, $\bar{P}''$, $\bar{Q}$ as empty partitions;
4  $B' = \{\{w\} \in V : |E^{-1}(\{w\})| = 0\}$;
5  $B'' = \{v \in V : |E^{-1}(\{v\})| \geq 1\}$;
6  push all $B'$ to $\bar{P}'$;
7  push $B''$ to $\bar{P}''$;
8  enqueue $B'' \in \bar{P}''$ to $L$;
9  enqueue all blocks $B' \in \bar{P}'$ to $L$;
10 $\bar{Q} \leftarrow \bar{P}''$;
11 while $|L| \neq 0$ do
12     $S \leftarrow dequeue(L)$;
13     $\bar{Q} \leftarrow I\text{-}input(S, \bar{Q}, L)$
14 end
```

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 2. This network is disconnected, containing two main weakly connected components, and in addition contains only positive regulation between 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.

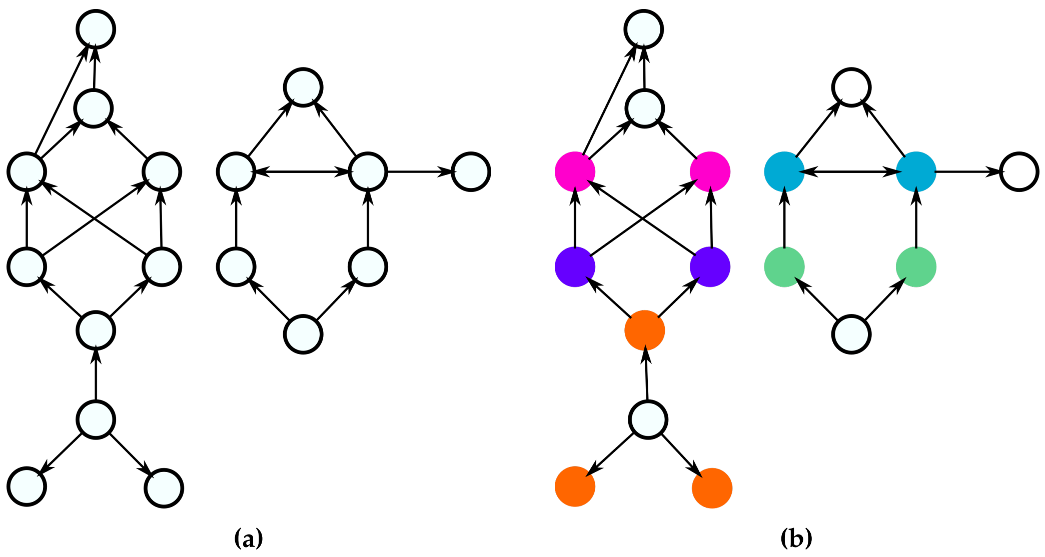
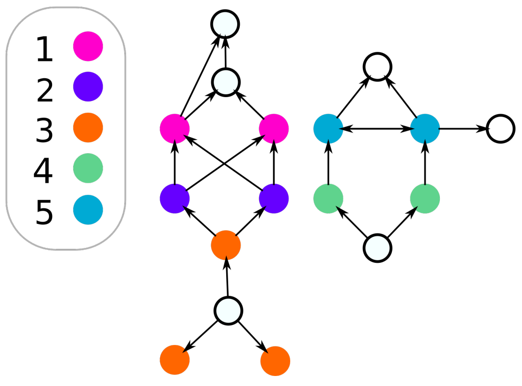


Figure 2: First trial network example. Containing a total of $N = 17$ nodes and $M = 22$ arcs, the network has two weakly connected components, defined as the connected networks when we treat all the directed arcs as undirected edges. For each non-trivial fiber I give the same color for the nodes belonging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

At 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 ℓ. Again, a fibration building block is defined as the nodes belonging to a fiber and all the ℓ external nodes that directly regulates at least one node of the current fiber. With that, we calculate the fundamental class number n_k of the fiber k by building the adjacency matrix $\mathbf{A}^k$ of its fibration building block and then calculating the largest eigenvalue of 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 2, we obtain the following fibration classification for each fibratiion building block of the network containing more than one node:



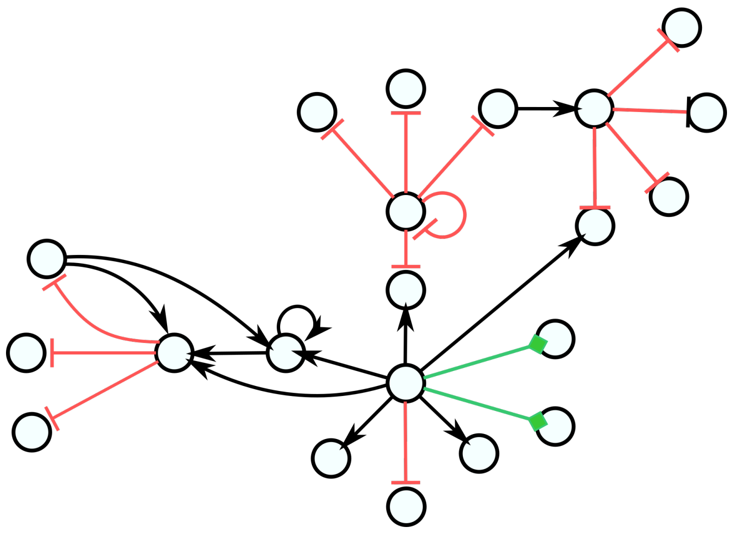
(a)

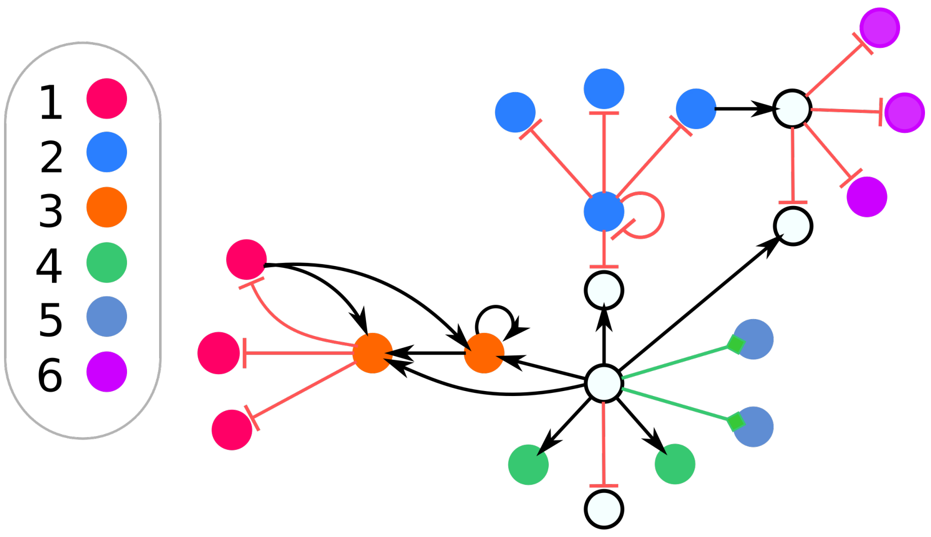
Fiber	$\lvert n,\ell\rangle$
1	$\lvert 0,2\rangle$
2	$\lvert 0,1\rangle$
3	$\lvert 0,1\rangle$
4	$\lvert 0,1\rangle$
5	$\lvert 0,2\rangle$

Figure 3: 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 this the branching ratio of the fiber has a fractal value represented by a golden ratio ϕ_d resulted by a cycle of information represented by a fibonacci sequence. This way, fiber 3 can be classified by the values $\lvert\phi_d, \ell = 2\rangle$ where the fiber has two external regulator nodes. The fibration classification for the non-trivial fibers is showed in the table 2.





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

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 [1]. The fiber distribution found is consistent with the one presented at [1], presenting the total of fifteen non-trivial fiber groups as shown in the figure 5.

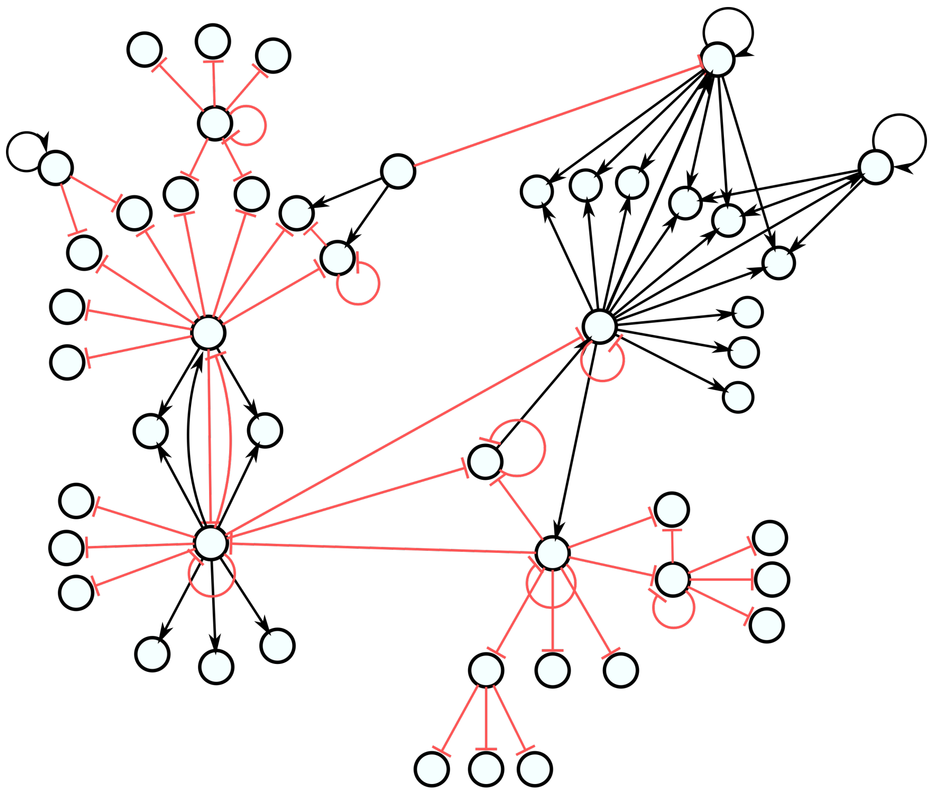


Figure 4

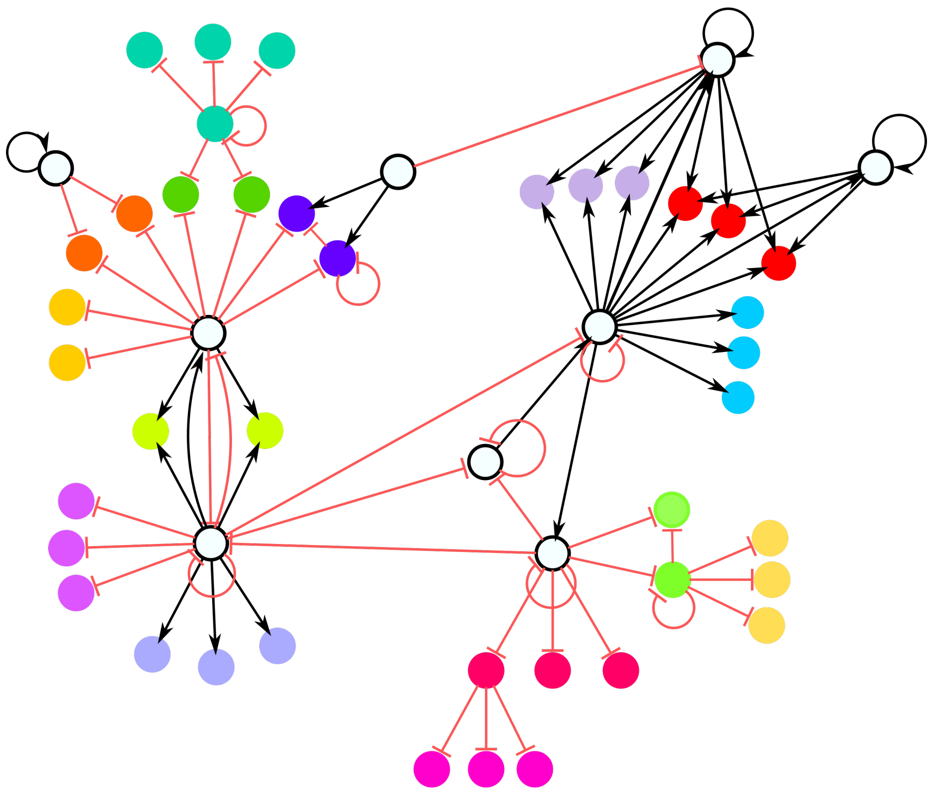


Figure 5

The connected component showed at the figure 4 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 5. In total, I have found 84 non-trivial fiber groups containing a total of 462 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. Even though the algorithm implementation and the provided data above support the correctness of the results of this report, they shows a discrepancy comparing with the results exhibited at [1] for the same regulatory network data.

The statistics concerning the fundamental class number and the external regulators of each fibration block on the network is given in the table X, showing describe the pattern of the statistics of the table that concerns the bacteria fiber statistics.

References

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

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

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

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

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