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

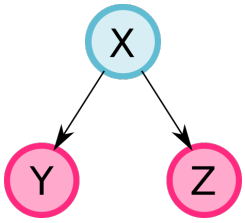
October, 25, 2019

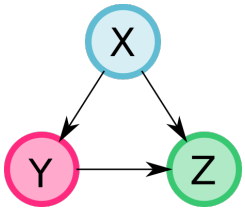
Abstract

Fibration building blocks of a information flow network represent the sets of nodes that are symmetric with respect to the processing of information, that is, the sets of nodes that process equivalent information. Here, we have reproduced important results concerning the identification and classification of the fibration building blocks of directed networks, constructed from real network data. More specifically, using the transcriptional regulatory network data of the *Escherichia Coli* bacteria, we quantify the clusters of nodes that synchronously process equivalent information and then we classify these clusters, called network fibers, based on its specific topological features. This way, in order to consistently present the obtained results, in this report we first give a brief description of the theory concerning the graph fibration morphism and its main definitions related to information flow symmetries. Next, we detail the methods adopted to correctly identify and classify the network fibers. More specifically, to establish an optimal framework, I show the implementation details of the Minimal Balanced Coloring algorithm used to find the corresponding fibers in the network, presenting a slightly improvement for the algorithm complexity and its implementation. At last, showing the proper methods for data preparation, I describe the results obtained concerning the fiber statistics for the specific case of the *Escherichia Coli* regulatory network, to properly compare with the recent results presented at *Morone et. al.* (2019).

1 Brief Introduction

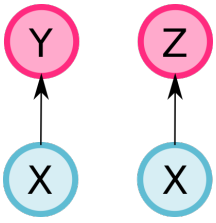
This statement requires citation [1]; this one does too [2] [3] [4] [5]. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean dictum lacus sem, ut varius ante dignissim ac. Sed a mi quis lectus feugiat aliquam. Nunc sed vulputate velit. Sed commodo metus vel felis semper, quis rutrum odio 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.

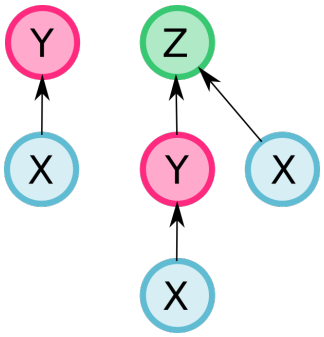




Nullam semper quam at ante convallis posuere. Ut faucibus tellus ac massa luctus consectetur. Nulla pellentesque tortor et aliquam vehicula. Maecenas imperdiet euismod enim ut pharetra. Suspendisse pulvinar sapien vitae placerat pellentesque. Nulla facilisi. Aenean vitae nunc venenatis, vehicula neque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

In order to identify the correct distribution of network fibers we have to find a efficient procedure to split the network in different disjoint partitions, in which all nodes from a same partition should receive information from the same other partitions. To do that, we treat our problem as the same of find the coarsest relational refinement partitioning for the given network. In this section, we detail the optimal algorithm used to find this coarsest partitioning in a context of graph fibrations and show the relative simple implementation of this method.

2.1 Algorithm description

The algorithm used in this project is a slightly modified version of the algorithm presented by Paige and Tarjan [4], having a time complexity of $O(M \log N)$, being very efficient for sparse networks. This algorithm has the same runtime order than the algorithm from Cardon and Crushemore [2], but it has a simpler implementation and smaller prefactors, exhibiting a better approach to our problem. Even though the algorithm applies for general situations, here we give the details of the algorithm to its application for a network, so we introduce the necessary definitions within this context.

A network G is completely defined as $G(V, E)$ by the sets of nodes V and of the connections E. The network has then $N = |V|$ nodes that can have connections with one another defined by the set E containing $M = |E|$ ordered pair of nodes, denoting the directed connections between the network nodes. Over the network, we can define a graph partition P of V as a set of pairwise disjoints subsets of V whose union is all V

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

where P_i are the elements of the partition P, in which we call it blocks. Within this context, if we consider an additional graph partition Q that has the property that all the block of Q is contained in a block of P, we say that Q is a refinement

partition of P.

Considering a block B in P, we say that the block B is stable with respect with a set S if or all elements of B connects an element of S or none element of B points to any element of S. By that we define that a graph partition P of V is stable if it is stable with respect with its own blocks. Having defined that, the graph coarsest partition problem is that of finding, for a given set of connections E and initial partition P over V, the coarsest stable refinement of P.

Considering all that, for the proper identification of the fibers on the network we have to construct a stable graph partition that is equivalent to the coarsest (minimal number of blocks) refinement of the network with respect to the information flow passing through each node. For that, we need to extend the concept of stability over the partition for one that accounts for the information received by each block of the partition. Thus, in order to identify the clusters with isomorphic input-tree for all their nodes, 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 P over the network $G(V, E)$ is input-tree stable with respect to S if for all the block $B \in P$, the equality

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

is satisfied for all the elements $x, y \in B$.

This way, we can benefit from the stability properties [4] to construct a refinement algorithm step that can obtains, through a finite number of steps, a input-tree stable partition from an initial unstable partitioning of the network. With this, given a subset $S \subseteq V$, the refinement step has the effect to refine the current partition, unstable with respect to S, by replacing it for a new partition now stable with S. For that, we define a split function *I-split(S, P)* that receive as input the current partition and a set S and results as output a new input-tree stable partition with respect to S. That function benefits from two major properties of stability (and input-tree stability): the stability inheritance by refinement and by union of sets. Since this refinement inheritance, a given set S can be used only once by the function *I-split*, for which after all other refinement will be 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. With all this considered, the essence of the algorithm can be stated as

> Find a set S in which the current partition P is input-tree unstable. Replace P by the output of *I-split(S,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 can 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 sets. Fortanately, this can be easily done for the construction of a input-tree stable partition.

Given a set S of nodes from the network G and a given input-tree unstable graph partition Q, the blocks $B \in Q$ that are input-tree unstable with respect to S can be splitted in several blocks B_j to have a stable input-tree for 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 split process take place. Then, for each unstable block $B \in Q$ all the splitted block, except the largest one, can be put 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.

Having said all that, the complete algorithm to find the correct network fibers of a network $G(V, E)$ consists in initializing a graph partition Q 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 Q 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 Q and all the isolated blocks defined at the beginning. Then, we proceed as

> Remove from L its first set S. Replace Q by the *I-split(S, Q)*. Whenever a block $B \in Q$ 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 Q represent 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.

2.2 Data preparation and algorithm implementation

We apply the above algorithm in the genetic regulatory network of the *Escherichia Coli* bacteria. We obtain the genetic E. Coli network through its transcriptional

regulatory interactions data, where each gene is regulated by a transcription factor protein. Since a transcription factor production in the cell is regulated by a gene, we can define a directed connection between two genes if a gene regulates the production of a transcriptional factor, which it regulates another gene. Since a transcription factor can be either an activator(positive) or repressor(negative), or even behaves as both(dual), the links between genes can carry different types of messages. Because of that, it is important that the partitioning algorithm accounts the type of message to construct appropriate input-trees for the network fibers partition. Therefore, for a proper application of the algorithm on the *Escherichia Coli* genetic network, we label each node uniquely, either as number or string names, and also each link with the type of connection between genes.

Considering this context, we explain now how the algorithm should proper be implemented to have an optimal perfomance concerning its runtime complexity. After discussing the data structures necessary to deal correctly with data, we design an algorithm recipe to show all the main steps of the process explained above. Also, anyone can acess our own personal implementation for genetic regulatory networks in the link `https://github.com/higorsmonteiro/fiberblocks` on github platform.

Given a 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. This means that the initial graph partition P is divided in two different partition P' e P'', the first one contained all the single-node blocks containing the solitaire nodes v in which $|E^{-1}(\{v\})| = 0$, and the second one containing initialing 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. Even though the blocks of P' are used as refinement sets, P' is not used by the algorithm. Thus, the final partition is the union of P' and the result of the refinement partitioning of P''.

After the above preprocessing we define the data structures for the partition Q for its blocks B. A partition is a doubly linked list of blocks, which allows that deletion of blocks be made in constante time $O(1)$ as long we have the block memory address during the procedure. A block is just a structure containing indexing data along with a doubly linked list containg all the nodes that belongs to it. Together with a 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 P'

e P'' to L. Then, we start the algorithm initializing a partition $Q = P''$ and by removing the top element set S of L, then we apply the *I-split(S, Q)* to identify the input-tree unstable blocks of Q and split them into input-tree stable blocks with respect to S. This way, all the splitted blocks are push 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. Even though this is the whole algorithm, 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. In respect to that below we propose the following implementation construction to the splitting function:

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;
  push all blocks $X \in \bar{T}$ to $\bar{Q}$;
end
delete all blocks $B \in \bar{U}$ in $\bar{Q}$

In the algorithm recipe 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 has 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 to be 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 above.

Finally, considering all the discussion above, we can explicit the complete algorithm in just a few steps.

Algorithm 2: Coarsest Refinement Graph Partitioning

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

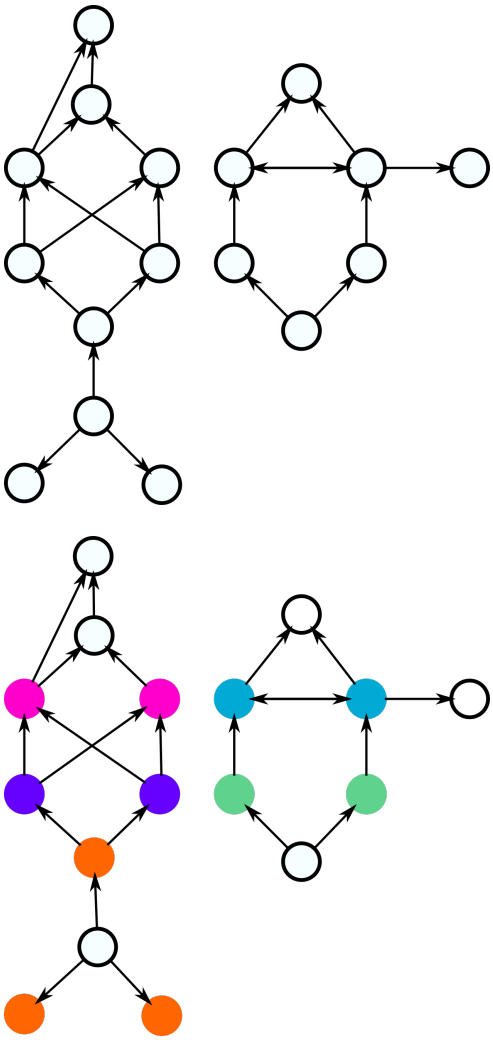
1 Initialize S as an empty set;
2 Initialize L as an empty queue;
3 Initialize $\bar{P}$, $\bar{P}'$, $\bar{P}''$, $\bar{Q}$ as empty partitions;
4 $B' = \{\{w\} \in V : |E^{-1}(\{w\})| = 0\}$;
5 $B'' = \{v \in V : |E^{-1}(\{v\})| \geq 1\}$;
6 *push* all B' to $\bar{P}'$;
7 *push* B'' to $\bar{P}''$;
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 $\quad S \leftarrow$ *dequeue*(L);
13 $\quad \bar{Q} \leftarrow$ *I-input*$(S, \bar{Q}, L)$
14 **end**

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 examples. Further information about the application of the codes in other genetic regulatory network data can be found at the same link given.

Results

3 Conclusion

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.

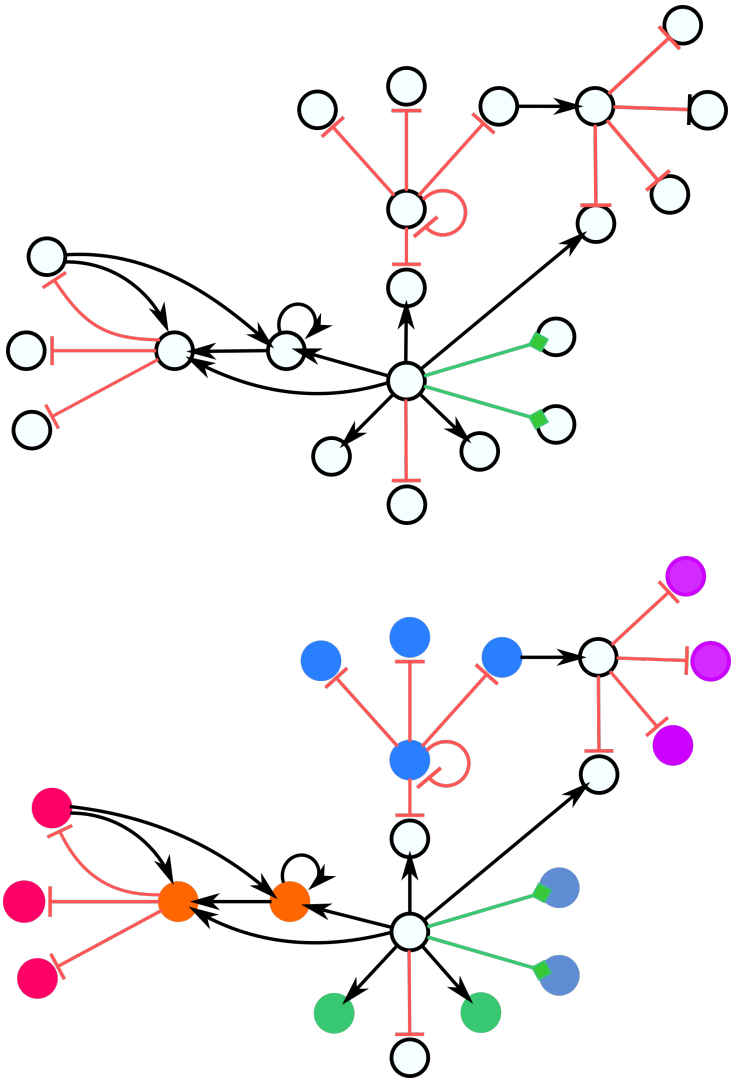
References

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.