Disjoint Motif in a Gene with pattern algorithm

Submitted by:

Thati Ayyappa Swamy[1]-AM.EN.U4AIE21084
B Indra Kiran-AM.EN.U4AIE21078

Abstract:

Biological networks provide great potential to understand how cells function. Network motifs, frequent topological patterns, are key structures through which biological networks operate. Finding motifs in biological networks remains to be computationally challenging task as the size of the motif and the underlying network grow. Often, different copies of a given motif topology in a network share nodes or edges. Counting such overlapping copies introduces significant problems in motif identification.

We develop a scalable algorithm for finding network motifs. Unlike most of the existing studies, our algorithm counts independent copies of each motif topology. We introduce a set of small patterns and prove that we can construct any larger pattern by joining those patterns iteratively. By iteratively joining already identified motifs with those patterns, our algorithm avoids constructing topologies which do not exist in the target network repeatedly counting the frequency of the motifs generated in subsequent iterations.

finding network motifs is scalable and computationally feasible for large motif sizes and a broad range of networks with different sizes and densities.

Introduction:

Biological networks describe how molecules interact to carry out various cellular functions. One common way to represent these networks is to use graphs, where the nodes and the edges represent the interacting molecules and the interactions between these molecules respectively. Studying biological networks has great potential to help understand how cells function and how they respond to extra-cellular stimulants. Such studies have already been used successfully in many applications. Characterizing the variations in drug resistance of different cell lines, or identifying the pathways serving similar functions across different organisms are only few examples among many.

Motifs are frequent topological patterns in a given network. Identifying motifs has been one of the key steps in understanding the functions served by biological networks such as gene regulatory or protein interaction networks. Motifs can be used to uncover the basic structure and design principles of a network. They are also often considered as the basic building blocks of a network and one of the network local properties. Thus, they can be used to classify networks into functional sub-units. It is worth noting that motifs have been used in various applications like prediction of

[1] CSE-Artificial Intelligence (A-Batch), Amrita Vishwa Vidyapeetham, School of engineering, Amritapuri, Kollam.

regulatory elements in genomic sequences. Despite the fact that studying motifs is of utmost importance for network analysis, motifs identification remains to be a computationally hard problem. The roots of the challenges behind motif discovery arise from several reasons. First, even when the motif topology is given, counting motif frequency (i.e., the number of occurrences of this motif), requires solving the subgraph isomorphism problem, which is NP-Complete.

Furthermore, when the motif topology is not known in advance, trying out all alternative topologies is infeasible as the number of such topologies increases exponentially with the number of edges in the motif. There are two ways for motif frequency formulation;

(i) allow for different copies of the same motif to overlap (i.e., share nodes or edges) or

(ii) count disjoint copies of the motif under consideration. Most of the existing methods in the literature on motif counting follow the first formulation.

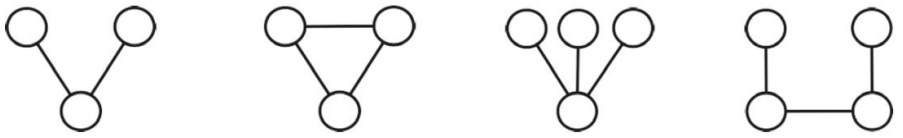
This formulation however has a fundamental drawback arising from the fact that it does not have downward closure property. Briefly, this means that the motif frequency does not decrease monotonically as the motif size increases. We discuss this drawback in detail in Sections "Summary of existing methods" along with why it makes it impossible to determine the largest sized motif in a given network. Several algorithms use the second formulation to compute the frequency of a given motif. Those algorithms, however, do not scale to large networks. Also, they are limited to small motifs as their time complexities grow exponentially with motif size. We elaborate on these methods in Section "Summary of existing methods" as well.

Definition and notation:

We classify the literature on motif identification and counting, based on the underlying frequency measure. This is because the frequency measure dramatically changes the cost of counting motifs as well as how we can interpret the frequency of the underlying pattern. Most of the existing studies use F1 frequency measure to count the embeddings of a pattern in a given graph. These methods carry the drawbacks inherent in the F1 measure. First, F1 ignores the fact that different copies of the same motif can overlap due to the nodes and the edges they share. This can lead to artificially massive number of motif embeddings as the same node or edge can participate in multiple embeddings. F1 counts three copies of the pattern (S1, S2, and S3). Different nodes and edges however contribute to this count at different numbers. The edge (a, b) appears only in S1 while (b, e) appears in both S1 and S3. Second and more importantly, the F1 measure is not downward closed. This is because as we grow a pattern by including new edges or nodes, its count as computed by F1 is not monotonic; it may decrease, stay the same, or increase. Lack of downward closure property makes it nearly impossible to decide if the motif found is the largest one in size while growing a pattern. Thus, using F2 is essential for the tractability of identifying frequent patterns. We use the F2 measure in this paper. Thus, the studies limited to the F1 measure are out of the scope of this paper. Several algorithms tackle the problem of finding frequent patterns in multiple graphs. FSG is one of the key methods in this class. These methods, however, do not count the number of occurrences of a pattern in each graph. They rather check if the given pattern appears at least once in each graph. Vanetik et al.also addressed the same problem. Finding frequent patterns or counting them without overlaps (i.e., using F2 or F3 measures) have

received little attention in the literature. One of the existing algorithms in this category is SUBDU. Flexible Pattern Finder Algorithm (FPF) detects frequent patterns using both F2 and F3.

However, these algorithms are computationally expensive and do not scale to large graphs or motifs. We evaluate SUBDUE and FSG experimentally in Section "Results and discussion".



The four basic patterns used by our algorithm which represent all patterns of two (**a**) or three undirected edges (**b**, **c**, and **d**)

Methods:

In this section we describe our method. Section "Algorithm overview" presents an overview of our algorithm. Section "Joining patterns to find larger patterns" explains the mechanism we use to grow motifs by joining smaller motifs. Section "Finding MIS: Going from F1 to F2" describes how we count disjoint motif instances. Section "Accelerating our algorithm through efficient filters" presents filtering techniques we implement to avoid costly isomorphism tests. Section "Complexity analysis" discusses the complexity analysis of our method.

Algorithm overview:

In this section, we provide an overview of our method for discovering motifs. At the heart of our method lie four unique graph patterns. We call them the basic building patterns for we use them as guide to construct larger motifs of arbitrary sizes and topologies. Figure presents these basic building patterns. We explain why we use these four specific patterns in Section "Joining patterns to find larger patterns" in detail. Algorithm 1 presents the pseudo-code of our method. We elaborate on each key step of our method in subsequent sections. The algorithm takes a graph G, the number of nodes of the target motif μ, and the minimum acceptable motif frequency as input α. For each of the four basic building patterns, it first locates all subgraphs in G that are isomorphic to that pattern (Line 1). Let us denote the set of instances of the ith pattern ($i \in \{1, 2, 3, 4\}$) with Si. In each set Si, it is possible to have overlapping subgraps. It then extracts the maximum set of edge-disjoint subgraphs in each set Si (Line 2) (see Section "Finding MIS: Going from F1 to F2" for details). Let us denote the resulting set with Si for the ith pattern. Notice that the cardinalities of the sets Si and Si are the F1 and F2 measures of the ith pattern respectively. The union of all the sets Si constitutes the current motif instances as well as the basic building pattern instances at this point (Line 3). The algorithm then iteratively grows the current motif set. At each iteration, it joins the current motif set with the basic building pattern set (Line 9). More specifically, a motif instance and a basic building pattern join if they share at least one edge. Joining two such subgraphs either creates a pattern which already exists in the current set (Line 10) or a new pattern (Line 12). At each iteration, after growing the current set, it filters the overlapping subgraphs to identify MIS for each pattern (Line 18). The algorithm removes all patterns with frequency lower than the user supplied cutoff (Line 21). It reports the frequent subgraphs that have as many edges as the target motif size (Line 23). The algorithm terminates when the current set cannot be grown to have any other patterns which satisfy the target motif (i.e., each pattern in the current set is either larger than

the target motif size or its frequency is lower than the user specified frequency).

Algorithm 1:

Motif Discovery algorithm

Input:

• Target motif size μ

• Frequency threshold α

• Input graph G = (V, E)

output:

• Motif topologies, and their instance subgraphs, that each

have same number of nodes as μ and its F2 α

1: BPSf 1 = getAllSubgraphs-Isomorphic-to-BasicPatterns()

2: BPS = extract-maxDisjointSubgraphs-PerPattern(BPSf 1)

3: CurrentSet (CS) = BPS

4: newSet (NS) = φ

5: **while** CS has new patterns and at least one of them with number of nodes < μ and its F2 α do

6: **for** each pattern p1 in CS do

7: **for** each pattern p2 in BSP where p2 = p1 do

8: **for** each subgraph s1 ∈ p1 and s2 ∈ p2 do

9: s3 = join(s1, s2)

10: **if** s3 ∈ existing pattern P then

11: add s3 ∈ P in NS if not duplicate

12: **else**

13: Create Pnew with s3 topology, add s3∈Pnew in NS

14: **end if**

15: **end for**

16: **end for**

17: **end for**

18: CS= extractmaxDisjointSubgraphsPerPattern(NS)

19: **for** each pattern p1 ∈ CS do

20: **if** F2 of p1 < α then

21: Delete p1 and all subgraphs ∈ p1

22: **else if** number of nodes of p1 = μ then

23: put p1 and all subgraphs ∈ p1 in the output

24: **end if**

25: **end for**

26: NS = φ

27: **end while**

Joining patterns to find larger pattern

Here, we describe one join iteration of our method; the process of joining the subgraphs of current set of patterns with the subgraphs of the basic building patterns to construct larger patterns. At the end of the iteration, the resulting set of subgraphs becomes the current set of subgraphs for the next join iteration. Recall that we join two subgraphs only if they share at least one edge. Joining two such subgraphs either yields a pattern that is isomorphic to one of the existing patterns or a new one. In the former case, we consider the set of subgraphs S isomorphic to that pattern. We check if the new subgraph is already in S. If it is in S, we discard it. Otherwise, we store it in S. In the latter case, we save this as a new pattern and also keep the corresponding subgraph. Notice that, although the subgraphs in S do not overlap prior to join, this may no longer hold after new subgraphs are inserted into S. At the end of each join iteration, we select the MIS for each pattern. We defer the discussion on how we do this to Section "Finding MIS: Going from F1 to F2". We then remove the patterns with F2 values below the user supplied frequency threshold, α. This eliminates non-promising patterns, and thus, reduces the number of candidate patterns for the next join iteration. Using the F2 measure ensures that patterns maintain downward closure property. Thus, non-frequent patterns will never grow to yield frequent patterns.

Why do we need different equivalence classes?

If the motif frequency is measured using F1, it is sufficient to join the subgraphs belonging to

existing patterns with only those which belong to the same equivalence class of the simple pattern with two edges (see Fig. 2a) to construct any larger pattern. This however is not true when F2 (or F3) is used to count the motif frequency. To understand the rationale behind this, recall that each equivalence class represents a set of disjoint isomorphic subgraphs. As a result, no two subgraphs from the same equivalence class join for they do not share any edges. Therefore, we need more than one equivalence class to construct new and larger patterns.

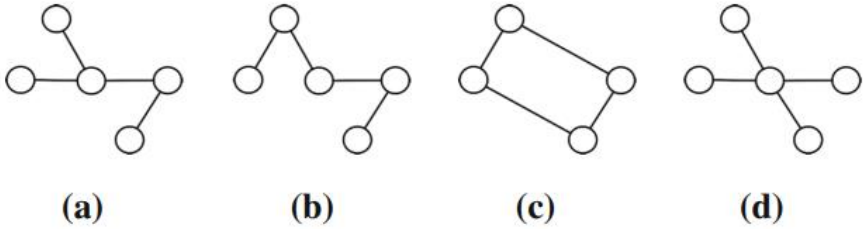


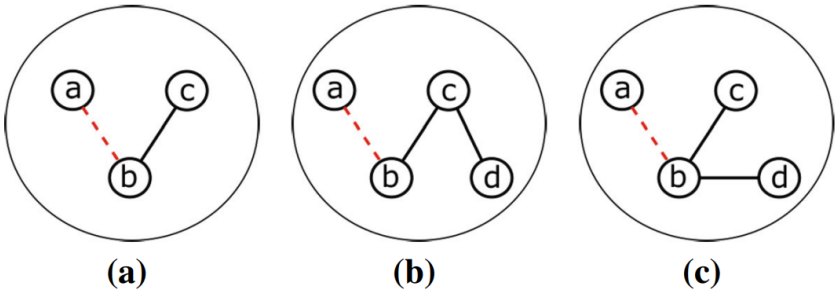
Fig. 3 All patterns which can be constructed with four undirected edges. **a**, **b**, and **d** represent patterns with 4 edges and 5 nodes while **b** represents pattern with four nodes and four edges

one is an overlapping combination of two of the basic building patterns. For instance, the pattern in Fig. 3a can result from joining the basic pattern in Fig. 2a with the basic pattern in Fig. 2c. It is worth noting that we can construct some of the patterns in Fig. 3 by joining two different pairs of basic building patterns. This redundancy ensures we can still locate a specific pattern even if one of those pairs does not exist. Therefore, our method can construct any pattern with four edges from patterns with three or two edges. We conduct our proof for the arbitrary pattern size by induction.

Induction step:

We assume that our method can construct any pattern with up to k edges ($k \geq 3$). We next show that any pattern with k +1 edges can be constructed by joining a pattern with k edges with one of the basic building patterns. Recall that the downward closure property states that those smaller patterns have at least as much frequency as the larger one according to F2. This means that if a pattern with k + 1 edges is frequent, then so is any of the k edge patterns obtained by removing an edge from that pattern.

Consider a graph G and a copy of a pattern P1 of size k edges in G, S1. Also, consider a copy of a pattern P2 with k + 1 edges such that P2 contains P1 and one additional edge. Let us denote this additional edge with (a, b). We need to show that P2 can be obtained from P1 by joining it with at least one of the basic patterns. Since both P1 and P2 are connected graphs, at least one of the two nodes a and b has an edge in P1. Without violating the generality of the proof, let us assume that b has an edge (b, c) in P1. Figure 4a illustrates the two edges (a, b) and (b, c). First, we consider using the basic pattern M1 in Fig. 2a in the join operation. In this case, a copy of M1, {(a, b),(b, c)} will join with S1 having a common edge (b, c) which will result in the pattern P2 with k+1 edges. This join however occurs only if the subgraph {(a, b),(b, c)} is included in the F2 counts of M1 (i.e. within the chosen non-overlapping copies of M1). If this condition fails, we consider the degrees of the two nodes b and c in pattern P1. We start with node c. Let us denote the degree of a node with function deg() (e.g. deg(c) is the degree of node c in pattern P1). If deg(c) > 1, then c has at least one more edge on top of (b, c). Let us denote this edge with (c, d). In this scenario, we join a copy of the motif M4 (Fig. 2d), {(a, b),(b, c),(c, d)} (if this copy exists in the F2 count of M4) to obtain P2.



(a) (b) (c)

Fig. 4 a A subgraph S_2 in a hypothetical graph G. S_2 is isomorphic to a pattern $P2$ of size $k+1$ edges. If we remove the additional edge (a, b) we obtain S_1 which is isomorphic to P_1 where $P_1 \subset P_2$. Notice that S_1 could have arbitrary $k-1$ edges rather than (b, c). Here we obtain S_2 as a result of joining S_1 with the subgraph $\{(a, b), (b, c)\}$ which belongs to $M1$ equivalence class (see Fig. 2a). **b** Failure to accomplish the join in (**a**), we seek to inspect $deg(c)$ and $deg(b)$ in S_1. The first possibility is that $deg(c) > 1$. This means that the subgraph $\{(b, c), (c, d)\}$ exists. We then can join S_1 with the subgraph $\{(a, b), (b, c), (c, d)\}$ which belongs to $M4$ equivalence class (see Fig. 2d) to obtain S_2 which is isomorphic to a pattern $P2$ of size $k+1$ edges. **c** The second possibility is that $deg(b) > 1$. This means that the subgraph $\{(b, c), (b, d)\}$ exists. We then can join S_1 with the subgraph $\{(a, b), (b, c), (b, d)\}$ which belongs to $M3$ equivalence class (see Fig. 2c) to obtain S_2

Finally, if deg(c) = 1, it is guaranteed that deg(b) > 1. This is because if both nodes b and c have degree one, S1 cannot be a connected subgraph. Let us denote one of the additional edges of b with (b, d) (see Fig. 4c). In this case, we join the subgraph that isomorphic to the pattern M3, {(a, b),(b, c),(b, d)}, with S1 to obtain P2. We can do this if this copy exists in the F2 count of M3. In summary, we conclude that any pattern P2 with k + 1 edges can be constructed by joining a pattern P1 with k edges (or k − 1 edges) and one of the basic building patterns to obtain the additional edge (or edges) if at least one of the many possible scenarios hold. We however cannot guarantee that the joins will find all of the instances of the k + 1 edge pattern on the target graph. Recall that as we aim to calculate the frequency of a given motif using F2, there is no self-join of any pattern. Thus, the basic building patterns set is the smallest set of patterns as we cannot construct one of those four patterns using the three other patterns. More specifically, this means that we cannot use only one of those four basic building patterns to construct larger patterns by joining pairs of subgraphs belong to that pattern's equivalence class. This is because if we join the embeddings of a single motif topology. we cannot get any larger pattern as they do not share any edge(s).

Finding MIS: Going from F1 to F2:

Here, we explain how we compute the F2 frequency for a given pattern. We use two algorithms for this purpose. We explain why we have two separate algorithms later in this section after describing the two algorithms. The first one is a heuristic used in the literature. This algorithm constructs a new graph, called the overlap graph for each pattern as follows. Each node in the overlap graph of a pattern denotes an embedding of that pattern in the target graph. We add an edge between two nodes of the overlap graph if the corresponding embeddings represented by those nodes overlap in the original graph. Once the overlap graph is constructed, the algorithm starts by selecting the node with the minimum degree (i.e., overlaps with the minimum number of embeddings) in the overlap graph. We include the subgraph represented by this node in the edge-disjoint set. We then delete that node along with all of its neighbouring nodes in the overlap graph. We update the degree of the neighbours of the deleted nodes. We repeat this process of picking the smallest degree node and shrinking the overlap graph until the overlap graph is empty. The algorithm described above works well for patterns with small number of embeddings. It however becomes computationally impractical as the number of embeddings of the underlying pattern gets large.

and updating it are computationally expensive tasks. Therefore, we use this algorithm for all patterns except for the basic building patterns (where number of embeddings are often too large). The second algorithm addresses the scalability issue of the the first one. This scalability issue is imposed by the expensive task of calculating the degree of each node in the overlap graph (i.e., the number of overlaps of each embedding). Recall from the previous algorithm that this number is considered as a loss value when selecting the node (i.e., embedding) with minimum degree (i.e., number of overlaps) to include in the final MIS of the pattern under consideration. Briefly, the second algorithm we introduce here avoids the expensive task of calculating number of overlaps for each embedding. The algorithm performs this by algebraically computing such numbers instead of performing actual overlapping tests. Once we compute node degrees of the overlap graph, this algorithm selects the disjoint embeddings the same way as the former algorithm described before. More specifically, the algorithm selects the node with the minimum degree and includes its corresponding embedding in the final MIS. It then removes neighbouring nodes to that node from the overlap graph. It repeats this process until the overlap graph is empty. Next, we explain how we compute the degree of a node in the overlap graph for the pattern M1 in Fig. 2a. Our computation is similar for the other three basic building patterns, yet tailored towards their specific topologies. Figure 5 shows a hypothetical subgraph S1 ={(a, c), (b, c)} in the input graph G which is isomorphic to M1. This subgraph is represented by a node in the overlap graph of M1's embeddings. Let us denote the degree of a node in the original graph G with function d() (e.g. d(vi) is the degree of node vi). Another embedding of M1 in G overlaps with S1 only if it contains the edge (a, c), or (b, c).

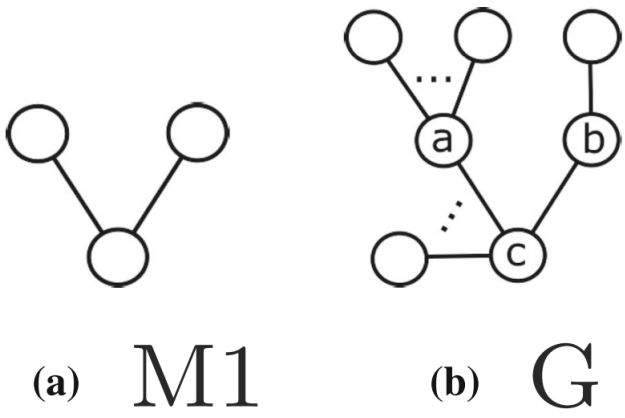


Fig. 5 a One of the basic building patterns. **b** A hypothetical graph that contains subgraphs isomorphic to the pattern M1 in (**a**)

Any edge in G connected to the middle node c forms two overlapping embeddings, one with the subgraph that has edge the (a, c) and the other with the subgraph that has the edge (b, c). We exclude the edges belong to S1 (i.e. the embedding we want to calculate its number of overlaps) itself from the potential edges of G that considered in the overlapping embeddings with S1. Thus, by excluding the two edges (a, c) and (b, c) from c's degree, node c yields 2 × (d(c) - 2) overlaps. In addition, any edge that belongs to node a forms an embedding when combined with the edge (a, c). Excluding the edge (a, c), node a yields d(a) - 1 overlaps. Similarly, node b produces d(b) - 1 overlaps. Thus, the total number of overlaps for the embedding S1 = {(a, c), (b, c)} combined from edges of its three nodes {(a, b, c)} is

$$2(d(c)-2)+d(a)-1+d(b)-1 = 2d(c)+d(a)+d(b)-6$$

To adapt our method to count non-overlapping embeddings of each pattern according to F3 instead of F2, we only need to change how we calculate the MIS of this pattern. More specifically, we change the criteria which states that "two subgraphs overlap if they share at least one edge" to "two subgraphs overlap if they share at least one node" (see Section "Definitions and notation"). This will result in changing the overlap graph constructed using the first method we explain in this section. In addition, it will also have slight change in calculating the total number of overlap of each embedding using the

second method we discuss in this section. Practically, we expect the overlap graph to be denser when we use the F3 measure as compared to that for the F2 measure. To illustrate this, consider the graph G in Fig. 1a and the pattern in Fig. 1c. This patter have 3 embeddings in G which are S1, S2, and S3 defined by the set of edges {(a,b), (a,c), (b,c), (b,e)}, {(e,f), (f,g), (e,g), (e,d)}, {(e,f), (f,g), (e,g), (b,e)} respectively. Figure 6a and Fig. 6b represent the overlap graph of this pattern based on F2 and F3 measures respectively.

Accelerating our algorithm through efficient filters:

Recall that at each iteration, our algorithm generates new subgraphs. For each of these subgraphs, it checks if this subgraph is isomorphic to one of the patterns constructed till that iteration. Isomorphism test is a computationally expensive task. Next, we describe how we avoid a large fraction of these tests. We develop two canonical labeling strategies for patterns. Canonical labeling assigns unique labels to the nodes of a given pattern. If two patterns are isomorphic, then they have the same canonical labeling. The inverse is however not true. Unlike isomorphism test, comparing the canonical labeling is a trivial task. Following from this observation, when we construct a new subgraph, we first compare its canonical labeling to those of existing patterns. We then limit the costly isomorphism test to only those patterns which have the same canonical labeling as the new subgraph. The first canonical labeling counts the degree (i.e. number of incident edges) of each node in the given pattern. It then sorts those degrees and keeps them as a vector we call the degree vector. If two patterns have different degree vectors, then they are guaranteed to have different topologies. Despite its simplicity, this labeling filters out a large fraction of patterns. We generate 1000 pairs of graphs where each pair is non-isomorphic and have the same number of nodes and edges. The degree vector successfully filters 85 % of the 1000 experiments. The second canonical labeling extends on the first one. It was first introduced by . Consider a pattern P = (V, E). Let us define the distance between two nodes vi, vj ∈ V as the number of edges on the shortest path that connects vi and vj and denote it with xij. Let us define the diameter of P as the maximum distance between any two nodes, and denote it with x.

Using this notation, we assign label to node vi as: $\sum_{j}^{j \in V} 2^{x - x_{ij} - d(v_j)}$. Once we compute the labels of all the nodes in the given pattern, we sort them. We call the resulting vector the nodes vector. Similar to the first labeling above, two isomorphic graphs are guaranteed to yield the same labeling. We compute and compare the nodes vector with only the patterns which cannot be eliminated using the first canonical labeling. We then consider the patterns with identical canonical labels for graph isomorphism.

Complexity analysis:

Here we analyze the complexity of our method. We refer to Algorithm 1 as we discuss the steps of our method. For each steep, we explain its complexity. We then summarize the complexity of all steps to denote the overall complexity of our method.

Find all subgraphs isomorphic to each of the four basic patterns:

In this step, we analyze each of the four basic patterns separately since they have different topologies. For the pattern M1 in Fig. 2a, to get all subgraphs isomorphic to this pattern, we consider all edges connected to each node in the underlying network. We select any two edges combination connected to every node.

Here, we denote the degree of a node with function d() (e.g. d(vi) is the degree of node vi).

Extract maximum disjoint set for basic patterns:

: In this step, we use the algebraic algorithm described in Section "Finding MIS: Going from F1 to F2" (second one) to calculate the number of overlaps of each subgraph belonging to each pattern equivalence class. This process takes constant time. We calculate this algebraic equations as we construct subgraphs in the previous step. We then sort those subgraphs within each equivalence class in decreasing order of their number of overlaps.

Recall from previous step that this number is

$$\mathcal{O}\left(\sum_{v_i \in V} d(v_i)^3 + \sum_{e_{ij} \in E} d(v_i)d(v_j)\right)$$

Thus, the complexity of this step is

$$\mathcal{O}\left(\left(\sum_{v_i \in V} d(v_i)^3\right)\right.$$

$$log\left(\sum_{v_i \in V} d(v_i)^3\right) + \left(\sum_{e_{ij} \in E} d(v_i)d(v_j)\right)$$

$$\left. log\left(\sum_{e_{ij} \in E} d(v_i)d(v_j)\right)\right).$$

Join Iterations (Lines 5–27):

In this step, we analyze the complexity of one join iteration. We then summarize the complexity of all join iterations. Let us denote the number of current patterns in iteration i with xi. Notice that, for the first iteration xi = 4. Recall that in each join iteration, we increase the size of each of the current patterns with one or two edges. In addition, the patterns of the first join iteration are at least of size 2. Thus, the size (i.e. number of edges) of each of the current patterns in iteration i is at least i + 2. The number of subgraphs isomorphic to each of the current patterns is at most |E| i+2 since they are non-overlapping subgraphs. Recall that the subgraphs of the basic patterns are non-overlapping within each pattern. Thus, the number of subgraphs of the patterns M1, M2, M3, and M4 are |E|/ 2 , |E|/ 3 , |E|/ 3 , and |E|/ 3 respectively. Collectively, the number of subgraphs of the basic patterns is O(|E|). In the join iteration, we start by joining subgraphs of current patterns with the subgraphs of the basic patterns (Lines 6–9). Thus, the total number of joins we perform at iteration i is

$$\mathcal{O}\left(|E|\frac{|E|}{i+2}x_i\right)$$

the complexity of all join iterations is

$$\mathcal{O}\left(\sum_{i=1}^{\mu-2}\left[x_i\frac{|E|^2}{i}log\left(\frac{|E|}{i+2}\right)\right]\left[x_i + \frac{|E|}{i^2}\right] + x_{i+1}\right)$$

In summary, the complexity of our method considering all the previous steps is

$$\mathcal{O}\left(\left(\sum_{v_i \in V} d(v_i)^3\right)\left(1 + log\left(\sum_{v_i \in V} d(v_i)^3\right)\right)\right.$$
$$+\left(\sum_{e_{ij} \in E} d(v_i)d(v_j)\right)\left(1 + log\left(\sum_{e_{ij} \in E} d(v_i)d(v_j)\right)\right)$$
$$\left. + \sum_{i=1}^{\mu-2}\left(\left[x_i\frac{|E|^2}{i}log\left(\frac{|E|}{i+2}\right)\right]\left[x_i + \frac{|E|}{i^2}\right] + x_{i+1}\right)\right)$$

Notice that xi here depends significantly on the topology and the density of the given network G.

Results and discussion:

The performance of the proposed motif discovery algorithm is evaluated on a real dataset for both undirected and directed networks. The runtime and the number of significant motifs are two primary criteria for evaluation of the proposed motif discovery algorithm. The runtime of the proposed motif discovery algorithm is compared against existing algorithms by varying both motif size and network size. Frequency measure F2 is used to compute motif frequency and z-score is used to measure the statistical significance of the identified network motif. The performance of the proposed algorithm is compared against MFinder, ESU, Grochow–Kellis, and MODA algorithms.

Implementation and environment:

We implement our algorithm in C++ and perform experiments on a computer equipped with AMD Opteron(tm) Processor 1.4 GHz CPU, 500 GBs of main memory running Linux operating system.

Evaluation of running time:

In this experiment, we evaluate the running time of our motif discovery algorithm. Our goal here is to observe the effect of varying parameters.

Conclusion:

In this paper, we developed a scalable method to solve the motif identification problem given an input graph, desired motif size μ, and minimum frequency of desired motif α. We proposed a set of small patterns, we call basic building patterns each containing two or three edges. We proved that any motif with four or more edges can be constructed as a join of these patterns. Our method first locates instances of the basic building patterns. It then iteratively grows known motifs at that iteration by joining them with the instances of these patterns. We developed efficient mechanisms to avoid a significant fraction of the costly isomorphism tests. We also introduced a new and efficient strategy for solve the MIS extraction problem. We analyzed the time complexity of our method based on the number of nodes and edges in the target network and the number of frequent motifs at each iteration. Our experiments on PPI networks from MINT comprehensively demonstrated that our method is significantly faster and more accurate than the existing methods. Fur thermore, we observed using synthetic networks that the running time of our algorithm is reasonable with growing the size of the target network and network density. We also showed using PPI networks that the increase in the running time of our algorithm is dramatically less than that of the competing methods as the motif size grows. We evaluated the statistical significant of the most abundant motif of PPI networks resulting from our algorithm.

Acknowledgments:

The authors acknowledge the intelligence for biological systems Lab of Amrita Vishwa Vidyapeetham for providing computational resources. We thank our very respectable teacher Ms. Manjusha mam and Ms. Aswathi mam for sharing the knowledge and guiding us for doing this project.

Code and Dataset:

Both the code and the data set are uploaded to a file available in the github link. The code has run with no error and with the expected output.

https://github.com/AyyappaSwamy-Sam/DisjointMotif

References:

(i) Zhu X, Gerstein M, Snyder M. Getting connected: analysis and principles of biological networks. Genes Dev. 2007;21(9):1010–1024.

(ii) Charlebois DA, Balázsi G, Kærn M. Coherent feedforward transcriptional regulatory motifs enhance drug resistance. Phys Rev E. 2014;89(5):052708

(iii) Ay F, Kellis M, Kahveci T. SubMAP: aligning metabolic pathways with subnetwork mappings. J Comput Biol. 2011;18(3):219–35

(iv) Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U. Network motifs: simple building blocks of complex networks. Science. 2002;298(5594):824–7.