Table of Contents

1

gLabTrie: a data structure for motif discovery with constraints

M. Mongioví[1*], G. Micale[2*], A. Ferro[3], R. Giugno[3], A. Pulvirenti[3] and D. Shasha[4]

[1] Semantic Technology Lab, ISTC-CNR, Catania, Italy,
misael.mongiovi@istc.cnr.it
[2] Department of Maths and Computer Science, Catania, Italy,
micale@di.unipi.it
[3] Department of Clinical and Experimental Medicine, Catania, Italy,
ferro@dmi.unict.it, giugno@dmi.unict.it,
apulvirenti@dmi.unict.it
[4] Courant Institute of Mathematical Science NYU, New York, USA
shasha@courant.nyu.edu

Abstract. Motif discovery is the problem of finding subgraphs of a network that appear surprisingly often. Each such subgraph may indicate a small scale interaction feature in a genomic interaction network, a significant relationship involving rock musicians, or insight into any other application that can be represented as a network. We look at the problem of constrained search for motifs based on labels (e.g. gene ontology term or musician type for those examples). This chapter presents a brief review of the state of the art in motif finding and then re-engineers the gTrie data structure from Ribeiro et al [23] to support labels. Experiments validate the usefulness of our structure for small subgraphs, showing that we recoup the cost of the index after only a handful of queries.

Keywords: graphs, motif finding, constraints

1 The Problem and its Motivation

A motif in a graph is a subgraph that appears statistically significantly often. Frequently occurring motifs may have practical significance. One familiar example is the ubiquity of feedback networks underlying homeostasis in biological, natural, and even economic systems. Motifs can also be useful in engineering disciplines such as synthetic biology. Kurata et al [1] use the frequent motifs found in biological networks as a library for synthetic biology. In fact Kurata et al pointed out that there are often motifs that behave as a single node in a larger network motif, just as an AND gate in an electronic circuit built out of transistors and resistors acts as a single node in a logic diagram. So, there may be motifs at different levels of abstraction. For the purposes of our chapter, we will take the usefulness of motifs for granted and talk about how to discover such motifs efficiently.

* These authors contributed equally to this work

Further, we will be particularly concerned with graphs whose vertices have labels. A constrained labeled motif query is to find a statistically significant motif satisfying some constraint on the labels.

In the sequel we will define our notion of statistical significance, but informally, this will entail a simulation of the following process: 1. find many random variations of the input graph G where each random variation preserves the degree counts of each node in G and preserves the number of edges linking nodes having each pair of labels. 2. See how often a labeled topological structure of interest is found in those random graphs. If infrequently, then the labeled topological structure is significant in G and constitutes a motif.

The computational challenge in motif finding is that the number of possible subgraphs could, depending on the graph, grow exponentially with the size of the subgraph. For sparser graphs, the growth may be less dramatic, but still rapid.

For that reason, we use data structures to make this fast. Our work builds directly on the gTrie data structure developed by Ribeiro [2] which is why we call our structure gLabTrie.

This chapter begins with a discussion of the data structure and algorithm we will use. We then follow with a discussion of how to find rare structures. Finally, we give an experimental evaluation of our structure and algorithms.

2 gLabTrie structure

2.1 Preliminaries

For simplicity, our discussion will center on undirected graphs, although our method works with directed graphs as well. Given a graph G, we denote by V_G its set of vertices, by E_G its set of edges, by L_G its alphabet of labels and by l_G a function that assigns a label to each vertex. We also write $G = (V_G, E_G, L_G, l_G)$. A subgraph G' of a graph G (denoted by $G' \subseteq G$) is a graph that contain a subset of vertices $V_{G'} \subseteq V_G$ of G and all edges of G whose endpoints are both in $V_{G'}$.

An *isomorphism* between two graphs G_1 and G_2 is a one-to-one mapping $\varphi : V_{G_1} \rightarrow V_{G_2}$ between vertices, which preserves the structure, i.e. $(u, v) \in G_1 \Leftrightarrow (\varphi(u), \varphi(v)) \in G_2$, and the labels, i.e. $l_G(u) = l_G(\varphi(u))$. If there is at least an isomorphism between G_1 and G_2, we say that they are *isomorphic* and write $G_1 \sim G_2$. An *automorphism* in G is an isomorphism between G and itself. Every graph admits at least one automorphism (where each vertex corresponds to itself). Typically, a graph can have many automorphisms. We abuse the notation and write $\varphi(G)$, with $G \subseteq G_1$ to denote the subgraph of G_2 that corresponds to G according to φ (i.e. the subgraph composed of vertices $\varphi(v)$ with $v \in G$ and edges $(\varphi(v_1), \varphi(v_2))$ with $(v_1, v_2) \in G$).

In what follows we use the terms *input network* (denoted by $\mathbb{G}$), *topologies* (denoted by G), i.e. unlabeled graphs that represent motif structures and *topology instances* (denoted by g), i.e. subgraphs of $\mathbb{G}$ that accommodate certain topologies. A *labeled topology* is an undirected (vertex-) labeled connected graph G. An *unlabeled topology* is a labeled topology stripped of its labels. A labeled topology that occurs frequently in G is also called *motif*. An *occurrence* g of a topology G is a connected subgraph of $\mathbb{G}$ that

is isomorphic to G. So, a given topology may have zero, one, or more occurrences in a graph.

Checking whether two (labeled or unlabeled) topologies are isomorphic is an expensive task that requires finding an isomorphism between the topologies or proving that no isomorphism exists. In motif discovery this operation has to be performed frequently to map a topology to the network subgraphs that conform to that topology. To simplify this operation, we map a graph to its *canonical form*, i.e. a string that uniquely identifies a topology and is invariant with respect to isomorphism. In other words two isomorphic graphs should have the same canonical form, while two graphs that are not isomorphic should have different canonical forms. Computing the canonical form of a graph may be expensive, but once it is computed, the isomorphism check entails simple string comparison.

An easy way to find a canonical form for an unlabeled subgraph is to consider all possible adjacency matrices of that subgraph (by reordering vertices in all possible ways), linearize them into strings (by putting all rows of an adjacency matrix contiguously in a unique line) and considering the smallest string (with respect to a lexicographic order). This simple approach guarantees invariance with respect to isomorphism since two isomorphic graphs have the same adjacency matrix except for their rows/columns order. The approach can be generalized to labeled topologies by including the sequence of labels in the string. Since enumerating all possible vertex orders is impractical, more efficient methods have been defined. A widely used method is nauty [22].

The canonical form of a graph is associated to a *canonical order* of vertices, i.e. the order of vertices that produces it. Note that a canonical form may be associated with more than one canonical order since a graph may have several automorphisms.

2.2 Problem definition

We aim to support *label-based queries* in which the user specifies a set of constraints and the system returns all topologies that satisfy the constraints. In our framework, a user specifies a frequency threshold, a p-value threshold and a bag (multiset) of labels that the motifs must contain. An example query would be: "Give me all labeled topologies of size k that have at least two A labels and one B label, occur at least f times and have a p-value smaller than p." We also want the query processing to be fast, so when a user is not satisfied with the response, he or she can change the constraints and quickly get a new response. We accept a slow (but still reasonable) off-line *preprocessing* step in exchange for fast *query processing*.

Formally, we define a *label-based query* (more simply a query) as a quadruple $Q = (C, k, f, p)$, where C is a bag of labels (a bag, also called a multiset, is similar to a set, but an element may occur more than once), k is the requested size of motifs, f is a frequency threshold and p is a p-value threshold.

Definition 1. ***Label-based query processing****. Given a network $\mathbb{G}$ and a query $Q = (C, k, f, p)$, find all labeled topologies T with number of vertices (size) k, whose number of occurrences in $\mathbb{G}$ is at least f and whose p-value is no more than p.*

We solve the defined problem in two steps. During an offline *preprocessing* phase we census the input network to find all labeled motifs up to a certain size K, and organize them in a suitable data structure (that we call the *TopoIndex*). Later, during the online *query processing* phase, we probe the TopoIndex to efficiently retrieve motifs that satisfy the query constraints.

In the remaining part of this section we describe how we extend existing approaches to support labeled motifs and the data structure used for quickly processing queries. Since our approach has been implemented on top of G-Trie, we first give an overview of G-Trie and our subsequent description will refer to it. However, our approach is general in that it can be applied on other network centric algorithms for motif discovery.

2.3 G-Trie method for unlabeled motif discovery

The main data structure of a network-centric method for motif discovery is a key-value map (hash table or search tree) that associates each unlabeled topology (up to a certain size) to a counter. Unlabeled topologies may be represented by their canonical form, so that the isomorphic check is efficient. G-Trie [23] generalizes tries to graphs. A gTrie organizes a set of unlabeled topologies in a multiway tree in such a way that subgraphs correspond to ancestors. An example of gTrie that stores all unlabeled topologies of size up to four vertices is given in Fig. 1.

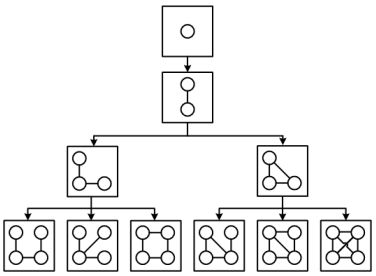

Fig. 1. Example of a gTrie with $K = 4$. The data structure stores all unlabeled topologies with up to 4 vertices. A similar, more detailed example can be found in [23]

Each node[5] of the gTrie stores information associated to the corresponding topology, typically a counter (not shown in the figure). A gTrie can be seen as a map that associates topologies to counters (similar in principle to a hash table or a binary tree).

To compute p-values, the GTrie system counts the number of occurrences of all unlabeled topologies in the input network and compares them with the corresponding number of occurrences in random networks with similar properties. The overall algorithm is in the figure marked Alg. 1.

[5] We use the term node to refer to parts of our data structures and vertex to talk about the graphs in which we find patterns.

Require: network, size K, frequency threshold f, p-value threshold p, number of randomizations r {returns the set of motifs with frequency $\geq f$ and p-value $\leq p$}
  initialize $gTrie$ with depth K
  call $census(network, gTrie)$
  initialize map_count
  for $i = 0 \ldots r$ **do**
    $rand\_net = randomize(network)$
    initialize $gTrie\_rand$ with depth K
    call $census(rand\_net, gTrie\_rand)$
    for all $t \in topologies(gTrie\_rand)$ **do**
      if $gTrie\_rand[t] \geq gTrie[t]$ **then**
        $map\_count[t] = map\_count[t] + 1$
      end if
    end for
  end for
  for all $t \in keys(map\_count)$ **do**
    $pval = map\_count[t]/r$
    if $gTrie[t] \geq f$ and $pval \leq p$ **then**
      output t, $gTrie[t]$, $pval$
    end if
  end for

Algorithm 1: Network-centric algorithm for unlabeled motif discovery: first find topologies in the input network that meet the frequency threshold, then compare the number of occurrences with the number of occurrences in each of a set of random graphs to evaluate the p-value of each such topology.

First a gTrie with all unlabeled topologies up to size K is built in the input network. Then the core procedure, $census()$, which takes as input a network and fills the gTrie[6] with the correct counting, is called. This procedure enumerates all subgraphs of the network one by one and increases the counter of the corresponding topology. Then, a map of counters (map_count) is initialized. This map is a hash table that associates topologies (more precisely canonical forms of topologies) to counters and is used to store the number of random networks in which a given topology occurs more than in the input network. Next, a number of randomizations of the input network are computed and $census()$ is executed on each of them. For every topology found, if its number of occurrence is greater than the one in the input network, its counter is increased. Function $topologies(gTrie)$ returns all topologies stored in $gTrie$ while $gTrie[t]$ refers to the counter associated with topology t in $gTrie$. At the end, frequencies and p-values are computed and all topologies that satisfy the input constraints are returned. In the next paragraphs we give more details about the core procedure, $census()$. Further details on the other parts can be found in [23].

The algorithm for graph census (procedure $census()$) is detailed in Alg. 2. The algorithm is based on the recursive procedure $Match$ that matches paths of the gTrie with all possible subgraph of the input network. At the beginning, the procedure $Match$ is called on the root of the gTrie and with an empty subgraph ($V_{used} = \emptyset$). The procedure picks one vertex at a time and starts to grow a subgraph from that vertex. Every time a new child of a gTrie node is explored, all neighbors of previously taken vertices ($N(V_{used})$) are considered and, if matchable, associated with the current node and added to the current subgraph (V_{used}). When a leaf node is considered, the node counter is increased. This means that a new subgraph isomorphic to the topology associated to that node was found.

Counting correctly entails ensuring that every subgraph is counted exactly once. Without symmetry breaking conditions, the $Match$ procedure would find some subgraphs multiple times. Indeed, if a subgraph has more than one automorphism (isomorphism between it and itself) there are multiple ways to obtain it. For instance, consider a network that contains a triangle with vertex ids 1, 2 and 3. The enumeration would produce the same triangle six times with the following sequences: $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ and $(3, 2, 1)$. Although multiple copies may be discarded by a post-processing step, this would require storing all subgraphs, which would be expensive for large subgraphs. Instead, the census algorithm considers a carefully designed set of symmetry breaking conditions that guarantees that each subgraph is enumerated exactly once. In the specific example the breaking conditions impose that the first vertex's identifier must be smaller than the second one's, and the second vertex's id must be smaller than the third one's. Thus only $(1, 2, 3)$ would be a valid sequence of vertices for the triangle. Details on how symmetry breaking conditions are computed are given in [23].

[6] In general the overall algorithm can work with any data structure that associates keys to values (e.g. hash tables) in place of gTrie. Keys are canonical forms of topologies, while values are counters.

Require: network, gTrie {returns the gTrie filled with the number of occurrences of each topology.}
$Match(gTrie.root, \emptyset)$
return $gTrie$

Procedure $Match(node, V_{used})$
if $V_{used} = \emptyset$ **then**
$V_{cand} \leftarrow V(network)$
else
$V_{cand} \leftarrow \{v \in N(V_{used}) : v$ satisfies symmetry breaking conditions$\}$
end if
$V \leftarrow \emptyset$
for all $v \in V_{cand}$ **do**
if v is connected with V_{used} as defined in $node$ **then**
$V \leftarrow V \cup \{v\}$
end if
end for
for all $v \in V$ **do**
if $isLeaf(node)$ **then**
$node.counter+ = 1$
end if
for all children c of $node$ **do**
$Match(c, V_{used} \cup \{v\})$
end for
end for
End Procedure

Algorithm 2: Census algorithm for unlabeled motif discovery

2.4 gLabTrie data structure for labeled motif discovery

A naive extension for handling labeled networks would consist in incorporating labels into the gTrie nodes. A node would represent a labeled topology as opposed to an unlabeled topology. However, this approach would cause an explosion of the number of gTrie nodes as the number of labels grows. Each node has to maintain both connectivity and label information and hence the same connectivity information would be stored multiple times.

To optimize memory, we resort to a different approach that consists in combining the canonical form of the unlabeled topology with the sequence of labels. This approach introduces the problem of determining the order of labels because the canonical order of unlabeled topologies is not sufficient. To clarify this point let us consider the two subgraphs in Fig. 2. Numbers represent the canonical order of vertices, while letters represent labels. Note that the order between 2 and 3 is ambiguous (1-3-2 would be a valid order as well) since by exchanging them we obtain the same unlabeled canonical form. The two labeled topologies are clearly isomorphic. However, the label sequences in the canonical orders are different (ABC vs. ACB).

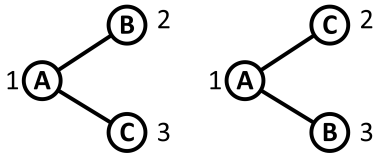

Fig. 2. Example of two unlabeled canonical orders that produce different sequence of labels on isomorphic graphs. The order is given by numbers. The two corresponding sequences of labels are ABC and ACB.

To guarantee that isomorphic labeled topologies have the same label sequence, we solve the ambiguity in the canonical ordering by ordering labels (e.g. in alphabetic order) and using this order to break ties. This is equivalent to choosing, among all possible canonical orders (of the single canonical form) of an unlabeled topology, the one that corresponds to the lexicographically minimum sequence of labels. We refer to a canonical order that satisfies this condition as a *lexically ordered canonical order*. Note that network-centric tools (e.g. gTrie) solve the ambiguity by considering the order of vertex ids to break the ties. Therefore, we just need to ensure that the order of vertex ids is consistent with the order of labels. This can be done by reassigning vertex ids of the input network so that vertices with smaller labels are assigned with smaller vertex ids (i.e. $v \leq u$ if $l_{\mathbb{G}}(v) \leq l_{\mathbb{G}}(u)$). We call a graph that satisfies this condition a *lexically numbered graph*.

The procedure described above solves the problem in Fig. 2. The order of the second topology is forced to be as in Fig. 3 and hence both label sequences would be ABC. Now we prove that this procedure always gives the correct result. Specifically, we prove that:

- if two labeled topologies are isomorphic then their associated labeled canonical forms (topology + label sequence) are equal;

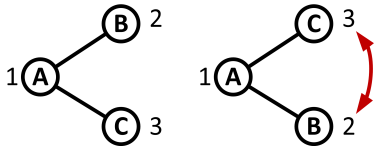

Fig. 3. By considering lexically ordered canonical orders we can guarantee that isomorphic graphs are associated with the same sequence of labels. In this example both sequences of labels are ABC.

– given two labeled topologies, if their corresponding labeled canonical forms are equal, then they are isomorphic (including their labels).

The second condition is trivial. Indeed the canonical order of two topologies defines an association between vertices that preserves both the structure and the label sequence.

In order to prove the first condition, we need to prove that if two labeled topologies are isomorphic then their corresponding sequence of labels coincide. In fact, the labeled canonical form is computed by combining the unlabeled canonical form with the sequence of labels. Since by stripping off the labels two isomorphic topologies remain isomorphic, the two unlabeled canonical forms coincide. Therefore we need to be concerned only about the label sequences.

Lemma 1. *Let G_1, G_2 be two labeled subgraphs of a lexically numbered graph and S_1, S_2 be the sequence of labels given by their lexically ordered canonical order. If G_1 and G_2 are isomorphic then $S_1 = S_2$.*

Proof: by contradiction. Suppose $S_1 \neq S_2$. Without loss of generality consider $S_1 < S_2$. Since G_1 and G_2 are isomorphic, there is at least an isomorphism between G_1 and G_2 (that is a one-to-one association between vertices of G_1 and vertices of G_2). We can use this isomorphism to construct an order of vertices for G_2 that is equivalent to the canonical order of G_1. This is a valid canonical order for G_2 since it produces the same unlabeled canonical form. Moreover this order corresponds to the same sequence of labels as S_1. That constitutes a valid canonical order for G_2 that produces a sequence of labels smaller than S_2. This contradicts the hypothesis that S_2 was obtained by a lexically ordered canonical order. QED

We modify the G-Trie algorithm to support labels. The main change concerns the information associated with gTrie nodes. Specifically, we substitute the counters of gTrie nodes with hash tables that associate label sequences to counters. To retrieve the counter of a labeled topology, we first look up the entry corresponding to its unlabeled topology, then we look up the counter associated with the label sequence in the corresponding hash table.

In summary, we apply the following changes to G-Trie:

1. Introduce a first step that reassigns ids to vertices of the input network so that vertices with smaller labels are assigned with smaller vertex ids (to create lexically numbered graphs);

2. Substitute the counters of gTrie nodes with hash tables that associate label sequences to counters;
3. Change the census procedure to increase the counters of labeled topologies as opposed as unlabeled ones.

Since the major changes are in the census algorithm we focus on the census procedure for labeled motif discovery (the overall algorithm for the labeled case is quite similar to that of the labeled case, but the labeled case requires the addition of the initial vertex ids assignment step). The census algorithm is shown in Alg. 3.

Require: labeled network, gTrie {returns the gTrie filled with hash tables with the number of occurrences of each labeled topology.}
$Match(gTrie.root, \emptyset)$
return $gTrie$

Procedure $Match(node, V_{used})$
if $V_{used} = \emptyset$ **then**
  $V_{cand} \leftarrow V(network)$
else
  $V_{cand} \leftarrow \{v \in N(V_{used}) : v$ satisfies symmetry breaking conditions$\}$
end if
$V \leftarrow \emptyset$
for all $v \in V_{cand}$ **do**
  if v is connected with V_{used} as defined in $node$ **then**
    $V \leftarrow V \cup \{v\}$
  end if
end for
for all $v \in V$ **do**
  if $isLeaf(node)$ **then**
    $label\_seq \leftarrow$ labels of V_{used} in lexically ordered canonical order
    $node.hash\_table[label\_seq] += 1$
    {now we have a hash table as opposed as a counter}
  end if
  for all children c of $node$ **do**
    $Match(c, V_{used} \cup \{v\})$
  end for
end for
End Procedure

Algorithm 3: Census algorithm for labeled motif discovery

2.5 An index for querying motifs

During the preprocessing phase we find all motifs up to size K (a pre-defined parameter) in the input network as described previously. We set neither a frequency threshold nor a p-value threshold at this point so that queriers can set thresholds of interest at

query time. One implication is that all labeled topologies occurring in the input network having size K or less are considered. For simplicity of exposition, in the following, we consider only motifs of size exactly K, although our method handles motifs with size smaller than K, as we explain later. We put all extracted labeled topologies in a data structure, which we call the *TopoIndex*, that facilitates later retrieval. An example of a TopoIndex for $K = 3$ and two labels (A and B) is depicted in Fig. 4.

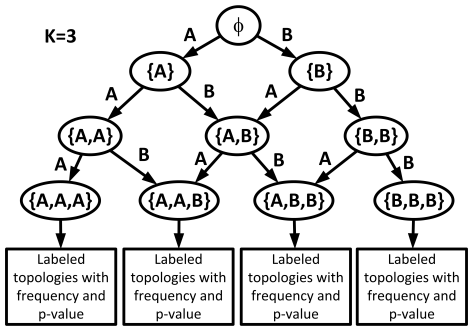

Fig. 4. The TopoIndex. Our data structure for processing label-based queries

The TopoIndex consists of a DAG, which embodies the super-multiset relation between sets, and a collection of lists of topologies contained in the leaves of the DAG. Specifically, nodes of the DAG represent bags of labels (label constraints) and an edge is drawn between two nodes u and v if v is super-multiset of u (i.e. it contains all labels in u with multiplicity below or attained to the one in v), and v has exactly one label more than u. The edge is associated with the label that is different between u and v. Each leaf (node that does not have any outgoing edges) contains a list of all labeled topologies that satisfy the label constraints associated with the leaf, with the topologies' frequencies and p-values.

The described TopoIndex enables fast lookup of a bag of labels and then fast retrieval of associated topologies (by exploring the part of the DAG reachable from the corresponding node). The DAG shown in the example in Fig. 4 is complete, i.e. it contains all possible nodes up to depth 3, but in general it may be not need to be complete. For instance, if there are no topologies with labels ABB and BBB, the nodes ABB, BBB and BB are not included in the DAG, thus saving time and space.

Building the TopoIndex The building procedure is given in Alg. 4. First we group the topologies by their label bags. Then, for each label bag we create a leaf and store it in a hash table that associates label bags with the corresponding nodes. We create the other nodes of the DAG by calling $create\_dag()$ (Alg. 5), which recursively removes one label at a time from nodes and creates nodes up to the root. The time complexity of Alg. 4 is $O(|T||K|\log(|K|) + |LB||K|^2)$, where LB is the set of label bags ($|LB| \leq |T|$). The labels of every topology need to be ordered (for comparison with other label bags) which can be done in time complexity $O(|K|\log(|K|))$. Grouping label bags can

be done in time $O(|T||K|)$ in expected time using a hash table. Inserting the label bags in the TopoIndex can be done in time $O(|LB||K|^2)$ since for every unlabeled topologies at most $|K|$ nodes need to be looked up by the recursive call $create\_dag()$ and looking up a node can be done in time $O(|K|)$. Since $|K|$ is usually very small, the building time is effectively linear in $|T|$.

Require: set T of labeled topologies of size K with associated frequency and p-value
  {returns the root of the TopoIndex data structure}
  group T by label bags
  for each label bag lb and its corresponding set of topologies T_{lb} **do**
    initialize $node$ {create a leaf node}
    $node.label\_bag = lb$
    $node.topologies = T_{lb}$
    $hash\_table[lb] = node$
    call $create\_dag(node, hash\_table)$
  end for
  return $hash\_table[\{\emptyset\}]$

Algorithm 4: Building the TopoIndex

Require: a node $node$ and the hash table of nodes $hash\_table$
  if $node.label\_bag == \emptyset$ **then**
    return
  end if
  for each label l in $node.label\_bag$ **do**
    $lb\_parent = node.label\_bag - \{l\}$
    if $lb\_parent \in keys(hash\_table)$ **then**
      $parent = hash\_table[lb\_parent]$
    else
      initialize $parent$ {create a new node}
      $parent.label\_bag = lb\_parent$
      $hash\_table[lb\_parent] = parent$
      call $create\_dag(parent, hash\_table)$
    end if
    $parent.children[l] = node$
  end for

Algorithm 5: Recursive procedure $create\_dag$ for building the TopoIndex

Query processing Given the TopoIndex described above, and a query $Q = (C, k, f, p)$ with $k = K$, query processing is quite straightforward. To perform a query $Q = (C, k, f, p)$ with $k = K$, first look up the node n of the DAG associated with the set of

labels in C, then explore all nodes of the DAG reachable from n. Finally, retrieve all topologies associated with reachable leaves and return the ones whose frequencies are greater than or equal to f and whose p-values are less than or equal to p.

The TopoIndex can be changed to support queries of size $k \leq K$ by associating internal nodes at depth k to labeled topologies of size k (for all $k = 1 \ldots K-1$). Answering queries with $k > K$ is the subject of our current work.

3 Alternative Methods of Calculating Statistical Significance

One might ask why we care about statistical significance (reflected in the p-value calculation in the previous section). Studies have shown that in many biological networks, small sub-networks of real networks that are much more frequent than random networks of the same size [3, 4] often act as functionally important modules. For example, in [3, 4] the authors identified motifs representing positive and negative autoregulation (subnetworks of one node and one edge), coherent and incoherent feed forward loops (subnetworks of 3 nodes and 3 edges), single-input modules (one node connected to few or many other nodes) and dense overlapping regulons (many nodes connected to few or many other nodes). One function of a coherent feed forward loop formed by a target Z and two transcription factors X and Y is the logic operation AND of a circuit: Z is activated by both X and Y, however Y is also regulated by X. Motif functionality has also been investigated with respect to evolution [6, 7] showing that motifs with the same topologies can have important functionality in different conditions.

That explains our interest in finding statistically over-represented substructures. This section discusses approaches to establishing statistical significance.

Formally, given a graph $G = (V, E)$ (directed or undirected) with n vertices whose ids are uniquely labeled with integers from 1 to n. A connected subgraph induced by a set of vertices of cardinality k (a *topology* for short) is called a motif when it occurs statistically significantly more often than the same subgraph in randomized networks derived from the original network [5].

The random generation method to find motifs given a real network consists of following steps: (i) generate a large set of random networks that share the characteristics of the real network; (ii) find candidate topologies, consisting of subgraphs in the real network; (iii) count the occurrences of these topologies (iv) assess the significance of each topology by computing its number of occurrences in each of the random networks.

The first step creates networks that have the same number of nodes and edges of the real network. Moreover, each node in the generated network maintains its original number of edges leaving and entering the node [8]. Next, by proceeding in an exhaustive manner, an algorithm can define all possible topologies of subgraphs with n nodes and count all the occurrences of such subgraphs in the real and in the random networks [5].

The random generation method consists of two expensive steps: the generation of a large number of networks and the application of subgraph isomorphism algorithms to compute the number of occurrences. Over the last decades, researchers have worked to reduce the expense of both steps. We list the main results in the following sections. For the sake of brevity, we point to the main alternative approaches, but give few details.

3.1 Quasi-analytical methods to assess the statistical significance of a topology

The random generation method described above evaluates the significance of the topology through the computation of a z-score using a Gaussian assumption or a p-value using a resampling approach [4, 5, 9, 10]. The Gaussian assumption may not apply to a particular application, but a reliable p-value requires a large number of random graphs whose analysis turns out to be computational expensive (by far more expensive than analyzing the target network alone). Recently, researchers have investigated the possibility of analyzing the distribution of the topologies, both non-induced and induced, from an analytical point of view that would avoid the need for random generation. Table 1 summarizes the main ideas of the two above approaches.

Table 1. P-value generation. Simulation vs Analytics approach

	Sampling + Permutation test	**Analytical model**
Idea	Generate random graph according to some random model. P-value is the fraction of graphs in which the occurrences in the random graphs is higher than the target one.	The target graph belongs to a given distribution. Define a Random Variable representing the number of occurrences of the motif under the reference model.
Pros	Easy to implement.	Computationally inexpensive.
Cons	Computationally expensive.	May not be possible to identify an appropriate distribution.

Approximation methods, based on the Erdos-Renyi (ED) model, have studied the asymptotic normality of the distribution of the count of the topologies [11]. Unfortunately, the Erdos-Renyi random model is a poor approximation to some networks of interest, such as biological networks [12]. Alternative reference models include the fixed degree Distribution (FDD) [8] that models the random generation method of swapping random edges. The swapping approach guarantees that a given node has the same valence in the random graphs as in the original one. There is also a variant of the FDD called Expected Degree Distribution (EDD) [13] and the Erdos-Renyi Mixture for Graphs (ERMG) [13]. Table 2 depicts the main features and differences of the models.

The EDD model generates random graphs whose degrees follow the distribution of the original graph, but particular nodes may obtain different valences. Conditional to the distribution of node degrees, the probability of edges is modeled as independent and exists with a probability proportional to the product of the degree distributions of the involved nodes. In the ERMG model, the nodes are spread among Q hidden classes with respective proportions
$\alpha_1, \cdots, \alpha_Q$. The edges are independent conditional on the class of the nodes. The connection probability depends on the classes of both nodes.

Table 2. Random Models: ER=Erdos-Renyi, FDD-Fixed Degree Distribution, EDD=Expected Degree Distribution, EMGR=Erdos-Renyi Mixture for Graphs

Name	**Characteristics**	**Graph Distribution generation**
ER	All edges of a graph are independent and exist with probability p.	The connection probability p of the ER model is estimated by the proportion of observed edges in the network.
FDD	Generates graphs whose degrees have exactly a given distribution.	For the given sequence of degrees in the input network, the graph is chosen uniformly at random from the set of all graphs with that degree sequence.
EDD	Generates graphs whose degrees follow a given distribution.	The empirical distribution of the degrees in the network is used as the distribution of the expected degrees.
EMGR	Nodes are spread among Q hidden classes with respective proportions p_1, . . . , p_Q. Edges are independent conditionally to the class of the nodes. The connection probability depends on the classes of both nodes.	Fit a mixture model.

It has been also shown that the use of the Compound-Poisson distribution [14] in the Erdos-Renyi random model allows the accurate approximation of the number of over-represented topologies [13]. In [13] the authors propose a model for the exact calculation of the mean and variance under any model of exchangeable random graphs (exchangeability means that the probability of occurrence of a topology does not dependent on its position in the graph, i.e., on the topological structure of the neighborhood of the topology). Furthermore, the authors have shown that the Polya-Aeppli distribution (also known as the Poisson Geometric distribution which is a special case of the Poisson-Compound distribution) is a good model for the distribution of the count of the topologies (both induced and non-induced) and leads to a more accurate p-value than a Gaussian model for the graphs of many applications. The reason is that the Geometric-Poisson distribution is particularly suitable for describing the number of events that occur in clusters, where a Poisson distribution describes the number of clusters and the counts of events within a cluster follows a geometric distribution. Here, this fits the case when distinct topologies can share nodes and edges (i.e. clumps) [13]. In fact, the authors show that when the number of clumps has a Poisson distribution with mean λ and the sizes of the clumps are independent of each other and have a Geometric distribution $G(1-a)$, the number of observed events X (topologies) has a distribution $P(\lambda, a)$ and leads to an estimate of the number of occurrences of a given topology (see Table 1).

So far, our discussion has concerned label-free (also known as color-free) networks. Schbath et al. [17] propose an analytical model for the computation of p-values for colored patterns. A colored pattern is a topology having a given multiset of colors (vertex labels). For example, a star of size 5 having 4 Bs and 1 C. An occurrence of the pattern is defined as a connected subgraph whose labels have a match with the multiset. Schbath et al would make no distinction between a star having the C in the center or one having the C on the outside. That subtle difference makes our job more difficult, but the starting point for our current research is their excellent work.i

Schabat et al. define analytical formulas for the mean and variance of the number of colored topologies by using the Erdos-Renyi model. Thanks to this, they were able to derive a reliable z-score for each topology. The authors then model the distribution of the count of coloured topologies under the Erdos-Renyi model.

3.2 Random Generation Methods

Whereas the previous subsection discussed analytical method, no published analytical method can discover p-values under our model of query (though, as mentioned, we ourselves are making progress towards that goal). So we turn to random generation methods. Improving random generation methods entails intelligent searching through graphs to enumerate topologies. The basic idea is to start from single nodes and expand them with their neighbourhoods in a tree like fashion, checking at each step that each subgraph in the tree appears only once and that it does not violate the color constraints of the query. This procedure can be further improved by sampling the network [3] or the neighbourhoods in the expanding phases [11]. Alternatively, Grochow and Kellis [18] used subgraph enumeration and symmetry breaking to avoid the search for automorphisms of the subgraphs occurrences. We now give some examples of the state-of-the-art algorithms upon which we build our structure.

The ESU algorithm [11] enumerates all subgraphs of size k by starting from a root vertex v of the graph and computing the occurrences of the topology by extending it node by node. The algorithm uses the concept of exclusive neighborhood which is defined as follows. For a subset $V^{'} \subseteq V$, its open neighborhood $N(V^{'})$ is the set of vertices in $V \setminus V^{'}$ which are adjacent to at least one vertex in $V^{'}$. For each node $v \in V \setminus V^{'}$, the exclusive neighborhood with respect to $V^{'}$ and denoted by $N_{excl}(v, V^{'})$, consists of all vertices that neighbours of v but are not in $V \cup N(V^{'})$.

The key idea of the algorithm is to add into the extension set of v, called $V_{Extension}$, only those vertexes satisfying the two following properties: (i) their vertex ids must be greater than v; (ii) must be neighbors only to the newly added w and not already in $V_{subgraph}$ (i.e. they must be in $N(w, V_{subgraph})$).

Its randomized variant, Rand-ESU, introduces an option that performs a uniform sampling in the graph, thus avoiding the need to explore it all. The algorithm is essentially the same as the original one with the exception that the recursion is carried out with a certain probability that decreases with the depth of the enumeration. In practice, the probability is high in the first steps of the recursion and then decreases as the size of the subgraphs to be explored increases.

The sampling in RAND-ESU is unbiased and is quite simple to implement. On the other hand, RAND-ESU gives only an estimate of the number of occurrences.

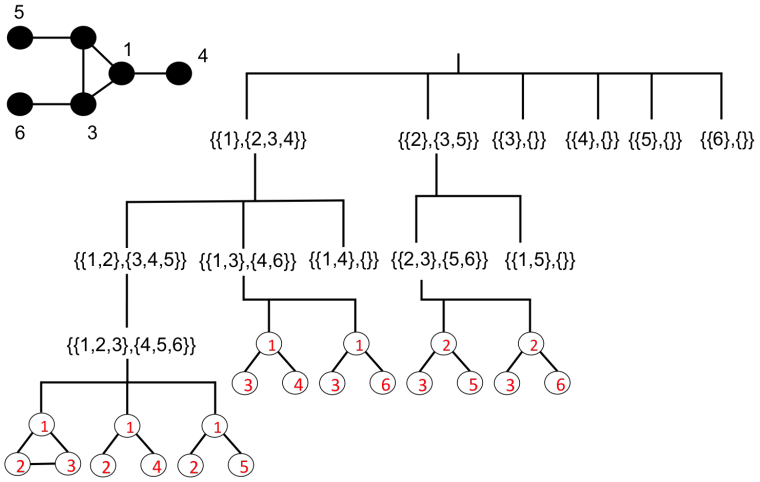

Fig. 5. The ESU tree for generating all subgraphs of k=3 nodes

Graph mining algorithms [19] find frequent subgraphs in a database of graphs or in a single large graph. A subgraph is frequent if its support (occurrence frequency) in a given dataset (or in a graph) is no less than a *minimum support* threshold. Computing the statistical significance of such topologies is done by simulation, as described above.

In this chapter, we consider the problem of searching for topologies of labelled graphs. However, there are several possible definitions of labeled topology.

In [17], the authors define a potential k-colored motif to be any connected subgraph of k nodes containing a specified multiset of colors (defined on the nodes). The motif is "potential" because its statistical significance may not meet a threshold. In this case, different topologies with the same labels define the same motif. Adami et al [20] consider the definition of colored motif as above, and use a measure based on entropy to determine the significance. In [11] and [21], the authors use the definition of motifs colored on both nodes and edges having a specific topology. [11] is based on the ESU algorithm, whereas Ribeiro et al [21] introduce a version of GTrie capable to find colored motifs.

In this chapter, we adopt the motif definition introduced in [21].

Definition 2. *Let G be a labeled graph. Let $m(V_m, E_m, LV_m, LE_m)$ be a subgraph of G with V_m nodes and E_m edges, where LV_m and LE_m are two sets of colors representing the labels of nodes and edges respectively. Let c be the number of isomorphic occurrences of m in G, and let α be a critical value. Let G_R be a random variant of G obtained by applying the edge shuffling method based on the Fixed Degree Distribution, and let c_R be the number of occurrences of m in the random variant G_R. We say that m is a motif of G if, by applying a permutation test using k random variant of G, $G_{R,i}$ (k = 500 usually), $\frac{\#(c_{R,i}>c)}{k} < \alpha$, where $\#(c_{R,i} > c)$ is the number of times the number of occurrences of m in $G_{R,i}$ is greater than in G.*

Because there is no analytical way to compute the significance of such a network motif yet, we will use the simulation on the random generated networks to establish the significance of colored network topologies. Algorithm 6 shows the implementation of a permutation test.

Require: network G, candidate topologies $m_1, m_2, \cdots, m_l$, c_i number of occurrences of m_i in G, number of iterations k, critical value α {returns the p-value of topology m_j}
  $s_j := 0$
  for $j = 1, \ldots l$ **do**
    for $i = 0 \ldots k$ **do**
      $G_{R,i} = randomize(G)$
      $c_{R,j}$:= number of occurrences of m_j in $G_{R,i}$;
      for $j = 0 \ldots l$ **do**
        if $c_{R,j} \geq c_j$ **then**
          s_j++
        end if
      end for
    end for
  end for
  for $j = 0 \ldots l$ **do**
    output p-value of topology m_j is s_j/k
  end for

Algorithm 6: Randomized Generation Test to Discover p-values

In our current efforts, we extend the analytical approach of [17, 13] to compute the significance of topologies given a multiset of colors.

4 Experiments

gLabTrie has been tested on a dataset of social, communication and biological networks. All experiments has been performed on the following configuration: Intel Core i7-2670 2.2Ghz CPU with a RAM of 8 GB. Tab. 1 describes the features of the selected networks.

Table 3. Networks used for experiments.

Name	Type	Nodes	Edges	Reference
FLIGHTS	undirected	2,939	15,677	[24]
BLOGS	directed	1,224	16,715	[25]
PPI	undirected	9,506	37,054	[26]
DBLP	directed	12,591	49,728	[27]
FOLDOC	directed	13,356	120,239	[28]
INTERNET	undirected	20,305	42,568	[29]

FLIGHTS is a network extracted from Openflights.org (http://openflights.org), representing all possible air routes between different airports around the world in 2011 [24]. BLOGS is a directed network of hyperlinks between weblogs on US politics of 2004 [25]. PPI is a protein-protein interaction (PPI) network in human, taken from HPRD database [26]. DBLP is the citation network of DBLP, a database of scientific publications, where each node in the network is a publication and edges connect two citations A and B iff A cites B [27]. FOLDOC is an oriented semantic network taken from the on-line computing dictionary FOLDOC (http://foldoc.org), where nodes are computer science terms and edges connect two terms X and Y iff Y is used to explain the meaning of X [28]. INTERNET represents the business relationships between autonomous systems (ASes) of Internet in 2005 [29].

Nodes of each network have been annotated with the following labels. In FLIGHTS, airports have been associated to one of the five continents. In BLOGS, nodes have been classified depending on their political leaning (liberal and conservative). For the labeling of nodes in PPI, we used Gene Ontology (GO) [30], a hierarchical dictionary of terms related to biological processes, components and functions, which have been extensively used for the analysis of biological networks so far [31, 32]. We annotated proteins with GO processes up to the first level of the hierarchy yielding 11 nodes labels. Ten of them represent specific kinds of biological processes (whole-organism process, metabolism, regulation, cellular organization, development, localization, signaling, response to stimulus, biological adhesion and reproduction). A special label representing the generic biological process has been associated to proteins for which we did not have GO annotations. DBLP nodes has been annotated with different kinds of publications (articles, inproceedings, proceedings, books, incollections, phd thesis and master

thesis) or "www" if the node refers to a cited website. INTERNET ASes have been partitioned into seven classes (large ISPs, small ISPs, customers, universities, internet exchange points, network information centers, not classified) according to the taxonomy described in [33]. Computing terms in FOLDOC have been labeled according to their domains (jargons, computer science, hardware, programming, graphics and multimedia, science, people and organizations, data, networking, documentation, operating systems, languages, software, various terms).

We compared the no-index version of gLabTrie with the index-based approach. We run our algorithm using default randomization parameters ($N_{rand} = 100$, $p = 0.01$ and $f = 2$).

The performance of gLabTrie has been evaluated with respect to three parameters:

a) m: the motif size, i.e. the number of its nodes;
b) c: the number of motif constraints, i.e. the number of specified node labels in the query;
c) l: the number of labels in the input networks.

For tests a) and b) we used real labels, while in case c) we ran our algorithm with randomly assigned labels. To measure the influence of these parameters, we varied the parameter of interest and assigned default values to the other ones ($m = 4$, $c = 4$ and $l = 2$). For each test, we ran gLabTrie on a set of 10 random queries. In the experiments with real labels, label constraints for random queries were generated according to the frequency of a node label: the more frequent a label x, the higher the probability that x is added as a label constraint to the query. In the tests with artificial labels, label constraints were added to the queries according to the uniform distribution of node labels.

Tab. 2 reports the running times for building indexes for motif of size m up to 4 in networks annotated with real labels. In all cases, the performance of gLabTrie strongly depends on the size of the network, its orientation (undirected graphs contain more instances of a certain topology on average) and the number of labels. Most of the time is spent in storing all the motif occurrences of a given size into the database. The number of occurrences increases exponentially with m.

Table 4. Running times (minutes) to build indexes on varying motif size.

Network	$m = 3$	$m = 4$
FLIGHTS	8.59	245.39
BLOGS	7.78	566.83
PPI	21.72	425.59
DBLP	30.91	1211.91
FOLDOC	46.28	1486.59
INTERNET	87.48	40605.23

Tab. 3 shows the results of the comparison between the no-index and the index-based approach of gLabTrie on querying motifs of different sizes, up to size 4. For each

network and each motif size, we reported the mean and the standard deviation. In both cases, the running time includes the time needed to retrieve all the subgraphs matching a given query.

Table 5. Running times (sec) for querying motifs of different size with no-index and index-based approach.

Network	m	No-index	Index
FLIGHTS	3	333.02 ± 4.71	0.01 ± 0.01
	4	364.81 ± 38.70	0.56 ± 0.86
BLOGS	3	155.36 ± 2.27	0.08 ± 0.15
	4	960.07 ± 159.01	1.44 ± 0.54
PPI	3	872.68 ± 21.77	0.01 ± 0.01
	4	866.10 ± 5.17	0.06 ± 0.10
DBLP	3	553.46 ± 4.02	0.11 ± 0.09
	4	882.63 ± 152.05	6.28 ± 5.06
FOLDOC	3	1290.23 ± 8.45	0.02 ± 0.01
	4	1308.40 ± 12.55	0.75 ± 0.22
INTERNET	3	2116.65 ± 6.38	0.70 ± 2.04
	4	2649.60 ± 1305.87	670.22 ± 200.59

The results show (unsurprisingly) that the index-based approach is much faster (100s of times) than having no index. We define q_{min} to be the minimum number of query operations required to recoup the time cost of building the index. For $m = 3$, $q_{min} \simeq 2$, so the time cost of building the index is recouped after two queries on average, while for $m = 4$ we have $q_{min} \simeq 44$.

It is worth noting that the benefit of the index decreases as the size of the network (measured in terms of the number of its nodes) increases. For instance, in the INTERNET network, which is by far the biggest network in our dataset, when $m = 4$ the index-based approach is only 4 times faster than the no-index one. In this case, the disappointing performance of the index-based approach is due to the very high number of query occurrences that the algorithm must retrieve from the dataset, resulting in a large number of I/O operations. In the INTERNET network with $m = 4$ the I/O time is 99% of the total running time, on average.

In Tab. 4, we compare the running times of the no-index and the index-based approach on querying motifs with a variable number of label constraints in the query. Again, network nodes have been annotated with real labels. We set $m = 4$ and we varied c from 1 to 4.

As the number of query label constraints defined by the user increases, the performance of both approaches improves. However, the more selective the query, the greater the benefit of the index. The gain enjoyed by the index is proportional to the size of the network and the number of constraints, because of the exponential decrease of the number of occurrences matching the query. For example, when c goes from 1 to 4, the

Table 6. Running times (sec) for querying motifs with variable number of label constraints with no-index and index-based approach.

Network	c	No-index	Index
FLIGHTS	1	685.06 ± 155.22	13.35 ± 9.6
	2	576.76 ± 116.71	5.70 ± 4.93
	3	412.65 ± 61.02	2.09 ± 3.32
	4	343.85 ± 17.17	0.52 ± 1.06
BLOGS	1	2214.74 ± 8.36	48.32 ± 2.75
	2	1953.47 ± 199.11	21.20 ± 20.42
	3	1430.81 ± 336.02	15.83 ± 20.09
	4	829.10 ± 214.59	6.16 ± 13.74
PPI	1	1228.73 ± 221.05	10.79 ± 10.18
	2	1116.63 ± 216.56	9.35 ± 11.52
	3	897.56 ± 34.87	0.51 ± 0.75
	4	861.30 ± 9.43	0.04 ± 0.06
DBLP	1	4041.38 ± 316.75	139.08 ± 1.84
	2	3186.24 ± 693.96	80.60 ± 34.95
	3	1867.96 ± 475.73	40.26 ± 18.37
	4	871.19 ± 131.45	7.43 ± 4.14
FOLDOC	1	3751.95 ± 686.84	78.29 ± 50.07
	2	2075.93 ± 334.84	7.24 ± 2.42
	3	1288.12 ± 42.41	2.47 ± 2.15
	4	1212.23 ± 16.79	0.58 ± 0.46
INTERNET	1	57988.44 ± 13722.07	11642.50 ± 5519.74
	2	28082.31 ± 13974.45	8984.35 ± 6945.80
	3	9165.14 ± 5849.21	3660.06 ± 3297.24
	4	2101.29 ± 35.61	0.71 ± 1.05

no-index approach becomes $\simeq 28$ times faster and the index-based approach $\simeq 16400$ times faster in the INTERNET network, while in the BLOGS network the two algorithms are only $\simeq 3$ and $\simeq 15$ faster, respectively.

Tab. 5 summarizes the results of the comparison between the performance of the two approaches when the number of labels vary. To perform these experiments, we annotated network nodes with artificial labels. Given a set of l labels, each node has been associated with a random unique label between 1 and l, according to a uniform distribution. We ran five different series of experiments with $l = 2, 6, 10, 14, 18$. In each series, we set $m = 4$ and $c = 4$.

Table 7. Running times (sec) for querying motifs with variable number of node labels with no-index and index-based approach.

Network	l	No-index	Index
FLIGHTS	2	483.08 ± 35.09	4.97 ± 1.29
	6	331.79 ± 3.18	0.09 ± 0.03
	10	327.36 ± 0.63	0.03 ± 0.01
	14	326.89 ± 0.39	0.04 ± 0.02
	18	327.00 ± 0.82	0.09 ± 0.02
BLOGS	2	931.77 ± 254.88	7.37 ± 2.32
	6	192.27 ± 29.18	0.29 ± 0.18
	10	160.67 ± 5.44	0.55 ± 0.05
	14	151.54 ± 2.77	1.16 ± 0.12
	18	149.44 ± 1.36	2.25 ± 0.10
PPI	2	1066.70 ± 95.43	2.10 ± 1.66
	6	870.64 ± 5.42	0.17 ± 0.12
	10	861.66 ± 1.72	0.04 ± 0.02
	14	860.80 ± 1.55	0.05 ± 0.03
	18	851.23 ± 2.16	0.10 ± 0.07
DBLP	2	1686.55 ± 496.97	32.66 ± 20.59
	6	623.22 ± 25.33	1.59 ± 0.94
	10	571.75 ± 11.92	1.23 ± 0.76
	14	559.46 ± 5.47	0.57 ± 0.36
	18	555.38 ± 2.36	1.32 ± 0.64
FOLDOC	2	2749.55 ± 539.16	18.67 ± 6.22
	6	1266.62 ± 43.13	0.81 ± 0.83
	10	1218.74 ± 10.16	1.01 ± 0.80
	14	1204.28 ± 6.85	1.63 ± 0.88
	18	1201.15 ± 1.64	3.07 ± 0.98
INTERNET	2	17270.40 ± 5731.44	1154.77 ± 2030.07
	6	2595.38 ± 382.33	94.18 ± 106.23
	10	2250.07 ± 111.54	23.98 ± 20.00
	14	2162.81 ± 44.00	5.68 ± 4.18
	18	2113.16 ± 20.06	5.08 ± 4.59

The time costs of both approaches decrease when the number of node labels increase. In all networks, the greatest reduction of the running time happens when we move from $l = 2$ to $l = 6$.

5 Conclusion

Our structures gLabTrie and TopoIndex contribute to all aspects of motif finding, by giving a very fast method for finding labeled topological structures in both input networks and related random networks. As this is work in progress, we plan in the near future to (1) find analytical methods for computing p-values on labeled topological structures to avoid the need for random graphs; (2) extend the search algorithms to enable search for topologies having, say, k vertices, even though the TopoIndex holds topologies of only a smaller size.

Acknownledgements

We very much appreciate the suggestions made by the anonymous reviewers to improve the presentation of this manuscript.

Shasha's work has been partially supported by an INRIA International Chair and the U.S. National Science Foundation under grants MCB-1412232, IOS-1339362, MCB-1355462, MCB-1158273, IOS-0922738, and MCB-0929339. This support is greatly appreciated.

References

1. Kurata, H., Maeda, K., Onaka, T., Takata, T.: BioFNet: biological functional network database for analysis and synthesis of biological systems. Briefings in Bioinformatics 15(5), pp. 699–709 (2014)
2. Ribeiro, P., Silva, F.: Querying Subgraph Sets with G-Tries. Proceedings of the 2nd ACM SIGMOD Workshop on Databases and Social Networks, pp. 25–30 (2012)
3. Alon, U.: Network motifs: theory and experimental approaches. Nature Reviews Genetics 8(6), pp. 450–461 (2007)
4. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network Motifs: Simple Building Blocks of Complex Networks. Science 298(5594), pp. 824–827 (2002)
5. Milo, R., Kashtan, N., Itzkovitz, S., Newman, M.E.J., Alon, U.: On the uniform generation of random graphs with prescribed degree sequences. Cond. Matter 2, pp. 1–4 (2004)
6. Kashtan, N., Alon, U. Spontaneous evolution of modularity and network motifs. PNAS 102(39), pp. 13773–13778 (2005)
7. Sol, R.V., Pastor-Satorras, R., Smith, E., Kepler, T.B. A model of large-scale proteome evolution. Advances in Complex Systems 5(1), pp. 43–54 (2002)
8. Newman, M.E.J., Strogatz, S.H., Watts, D.J.: Random graphs with arbitrary degree distributions and their applications. Physical Review E 64(2), 026118 (2001)
9. Prill, R.J., Iglesias, P.A., Levchenko, A.: Dynamic properties of network motifs contribute to biological network organization. PLoS Biology 3(11): e343 (2005)
10. Shen-Orr, S., Milo, R., Mangan, S., Alon, U.: Network motifs in the transcriptional regulation network of Escherichia coli. Nature Genetics 31(1), pp. 64–68 (2002)

11. Wernicke, S.: Efficient detection of network motifs. IEEE/ACM Transactions on Computational Biology and Bioinformatics 3(4), pp. 347–359 (2006)
12. Barabasi, A.L., Albert, R.: Emergence of Scaling in Random Networks. Science 286(5439), pp. 509–512 (1999)
13. Picard, F., Daudin, J., Koskas, M., Schbath, S., Robin, S.: Assessing the Exceptionality of Network Motifs. Journal of Computational Biology 15(1), pp. 1–20 (2008)
14. Adelson, R.M.: Compound poisson distributions. OR 17(1), pp. 73–75 (1966)
15. Barbour, A.D., Holst, L., Janson, S.: Poisson approximation. Oxford Studies in Probability, Oxford-University Press (1992)
16. Nuel, G.: Cumulative distribution function of a geometric poisson distribution. Journal of Statistical Computation and Simulation 78(3), pp. 385–394 (2006)
17. Schbath, S., Lacroix, V., Sagot, M.: Assessing the exceptionality of coloured motifs in networks. EURASIP Journal on Bioinformatics and Systems Biology 2009:616234 (2009)
18. Grochow, J., Kellis, M.: Network Motif Discovery Using Subgraph Enumeration and Symmetry-Breaking. Research in Computational Molecular Biology, Lecture Notes in Computer Science 4453, pp.92–106 (2007)
19. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 721–724 (2002)
20. Adami, C., Qian, J., Rupp, M., Hintze, A.: Information Content of Colored Motifs in Complex Networks. Artificial Life 17(4), pp. 375–390 (2011)
21. Ribeiro, P., Silva, F.: Discovering Colored Network Motifs. Complex Networks V, Studies in Computational Intelligence 549, pp. 107–118 (2014)
22. McKay, B.D.: Practical graph isomorphism. Congressus Numerantium 30, pp. 45–87 (1981)
23. Ribeiro, P., Silva, F.: G-Tries: a data structure for storing and finding subgraphs. Data Mining and Knowledge Discovery 28(2), pp. 337–377 (2014)
24. Opsahl, T.: Why anchorage is not (that) important: Binary ties and sample selection. http://toreopsahl.com/2011/08/12, (August 2011)
25. Adamic, L.A., Glance, N.: The Political Blogosphere and the 2004 U.S. Election: Divided They Blog. Proceedings of the 3rd International Workshop on Link Discovery (LinkKDD), pp. 36–43, ACM, New York (2005)
26. Prasad, T.S.K., Goel, R., Kandasamy, K., Keerthikumar, S. et al.: Human Protein Reference Database - 2009 Update. Nucleic Acids Research 37(suppl. 1), pp. D767–D772 (2009)
27. Ley, M.: The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. Proc. Int. Symposium on String Processing and Information Retrieval 2476, pp. 1–10 (2002)
28. Batagelj, V., Mrvar, M., Zavesnik, M.: Network analysis of dictionaries. Language Technologies, pp. 135–142 (2002)
29. Dimitropoulos, X., Krioukov, D., Huffaker, B., Claffy, K.C., Riley, G.: Inferring AS Relationships: Dead End or Lively Beginning?. 4th Workshop on Efficient and Experimental Algorithms (WEA), Springer Lecture Notes in Computer Science, pp. 113–125 (2005)
30. Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., et al.: Gene Ontology: tool for the unification of biology. Nature Genetics 25(1), pp. 25–29 (2000)
31. Maere, S., Heymans, K., Kuiper, M.: BiNGO: a Cytoscape plugin to assess overrepresentation of Gene Ontology categories in Biological Networks. Bioinformatics 21(16), pp. 3448–3449
32. Bindea, G., Mlecnik, B., Hackl, H., Charoetong, P., Tosolini, M. et al.: ClueGO: a Cytoscape plug-in to decipher functionally grouped gene ontology and pathway annotation networks. Bioinformatics 25(8), pp. 1091–1093
33. Dimitropoulos, X., Krioukov, D., Riley, G., Claffy, K.C.: Revealing the Autonomous System Taxonomy: The Machine Learning Approach. Passive and Active Measurements Workshop (PAM), pp. 91–100 (2006)

Subject Index