

Rand-FaSE: fast approximate subgraph census

Pedro Paredes¹ · Pedro Ribeiro¹

Received: 7 March 2014 / Revised: 25 February 2015 / Accepted: 13 May 2015 / Published online: 27 May 2015
© Springer-Verlag Wien 2015

Abstract Determining the frequency of small subgraphs is an important graph mining primitive. One major class of algorithms for this task is based upon the enumeration of all sets of k connected nodes. These are known as network-centric algorithms. FaSt Subgraph Enumeration (FaSE) is an exact algorithm for subgraph counting that contrasted with its past approaches by performing the isomorphism tests while doing the enumeration, encapsulating the topological information in a g -trie and thus largely reducing the number of required isomorphism tests. Our goal with this paper is to expand this approach by providing an approximate algorithm, which we called Rand-FaSE. It uses an unbiased sampling estimator for the number of subgraphs of each type, allowing an user to trade some accuracy for even faster execution times. We tested our algorithm on a set of representative complex networks, comparing it with the exact alternative, FaSE. We also do an extensive analysis by studying its accuracy and speed gains against previous sampling approaches. With all of this, we believe FaSE and Rand-FaSE pave the way for faster network-centric census algorithms.

Keywords Complex networks · Graph mining · Subgraphs · G-tries · Network motifs · Graphlets

1 Introduction

A large variety of real systems can be seen as a complex network, with graphs appearing as an ubiquitous abstract representation, serving as the base model for multitude of applications (Costa et al. 2011). It is therefore only natural that graph mining has been receiving increasing attention in the past years. One way of studying networks is to search for interesting groups of nodes. These groups may have a relatively large size, as is the case with community detection (Fortunato 2010). However, they can also be of smaller sizes, like it is the case on network motifs discovery (Milo et al 2002) or graphlet-based metrics (Pržulj 2010).

These methodologies have been applied with success to a wide range of real systems, such as in the social networks domain, where motifs have been used, for instance, to characterize and classify co-authorship networks (Choobdar et al. 2012a) or wikipedia edition networks (Wu et al. 2011). Likewise, graphlets have been used to provide a complete characterization of social networks, allowing the selection of an adequate graph model (Janssen et al. 2012). These methodologies have also been successfully applied to other domains, such as biological networks (Sporns and Kötter 2004; Albert and Albert 2004), engineering systems such as electronic circuits (Itzkovitz et al. 2005) and also on software architecture (Valverde and Solé 2005).

Computing the frequencies of subgraphs in the network being analyzed is also known as performing a *subgraph census*, and plays a central role in most of these methods. For example, a network motif is defined as a statistically significant subgraph, which means that its frequency in the original network is much higher than in similar random ones (Milo et al 2002). Thus, this method requires a subgraph census for the original network, but also for an

✉ Pedro Ribeiro
pribeiro@dcc.fc.up.pt

Pedro Paredes
pparedes@dcc.fc.up.pt

¹ CRACS and INESC-TEC, DCC-FCUP,
Universidade do Porto, Porto, Portugal

ensemble of randomized networks (Ribeiro et al. 2009). However, calculating the frequency of a subgraph is a computationally hard task since it is closely related to the classic *subgraph isomorphism* problem, which is known to be NP-Complete (Cook 1971). The execution time of any algorithm that calculates this is bounded by the amount of subgraphs being enumerated and this number grows exponentially. Thus, the applicability is limited to relatively small subgraph sizes. By decreasing the execution time, we are effectively pushing the limits on which a subgraph census computation is feasible. And even increasing by one node the size of the subgraphs being searched, new insight into a network can be gained because patterns previously unknown may emerge.

We can divide the previous algorithms for this problem into three main groups related to their conceptual approach. *Network-Centric* algorithms, such as the one we present in this work, ESU (Wernicke 2006) or KAVOSH (Kashani et al. 2009), compute the frequency of all possible k -sized subgraphs in the original network. By contrast, *Subgraph-Centric* algorithms, such as the one by Grochow and Kellis (2007), search for one single specific subgraph. Finally, the *Set-Centric* approach of g-tries (Ribeiro and Silva 2014b) is conceptually in the middle, allowing for computing the frequency of a customized set of subgraphs.

In this work, we aim at improving the network-centric approach, and thus our algorithm requires as its input a network and a subgraph size k . Note that for the proposed task, subgraph-centric methods would still be able to do the full enumeration, albeit they would need to search individually for all possible k -sized subgraphs. Likewise, set-centric methods would need to receive as input the same set of all possible k -sized subgraphs, regardless of having no guarantees that all possible subgraph types will appear on the network being analyzed. Network-centric methods can be summarized through two major steps: enumeration of connected sets of k nodes and isomorphism tests to determine to which subgraph type each enumerated set belongs to. Past classical approaches do this independently: the enumeration part gives origin to sets of k nodes and afterward each one of this sets is inputted into an isomorphism computation (typically by calculating a canonical labeling) so that the correspondent subgraph type frequency can be incremented. This means that the number of performed isomorphism tests is equal to the number of occurrences of subgraphs, even though the actual number of existent subgraph types is generally much smaller.

This paper extends the work done in Paredes and Ribeiro (2013), where we presented the FaSE algorithm, a network-centric approach that aims precisely at reducing this very redundancy of performing one isomorphism test per subgraph occurrence. Instead of postponing this calculation step, we consider it while doing the enumeration

by storing it in a customized version of a g-trie, a tree-like data structure that works as a prefix tree of graphs, so as to take advantage of the underlying structure of the graphs being enumerated. Whenever a new node is selected to be added to the enumerating set we either create a new edge on the g-trie or try to follow an existing one that corresponds to a topologically equivalent graph, where equivalence here is defined by an intermediate set of classes given by the way we label each subgraph type (a process we called LS-Labeling). A path from the root node to any node in the tree corresponds to a different node permutation of a certain graph type (something that is given by our labeling algorithm). To know the true subgraph type of each occurrence, we compute a canonical labeling for each leaf in the g-trie. By doing so, we are able to only compute one isomorphism test per leaf and thus avoid repeating this calculation for two subgraphs that have the same node permutation and are equivalent on our g-trie. Note that as we have shown, the number of leafs is proportional to the number of isomorphism classes and is usually very small compared with the total number of subgraphs. All in all, we end up getting an internal subgraph representation that looks out for the topology and common substructures of the enumerated graphs, which allows for further improvements based on this information.

In this paper, we also expand upon our previous description of FaSE, by further explaining its behavior. Moreover, we introduce an approximation approach that samples the search space to estimate the exact value of the frequency of present subgraph types. This classic trade of accuracy for speed technique has been applied in the past, since many real-world networks are of large scale and thus it is unfeasible to perform an exhaustive full enumeration. Most previous approaches work in a similar fashion by sampling a fraction of all enumerating subgraphs (Kashtan et al. 2004; Omidi et al 2009; Wernicke 2006). Our sampling works by considering the enumeration as a recursion tree (which is possible due to the way we enforce this process) and by only exploring certain branches with a predefined probability. We end up with an unbiased sample that we use to estimate the real frequency of all subgraph types. This results in an algorithm that is able to achieve higher values of subgraphs sampled per second and thus obtain the same accuracy of the previous network-centric approaches, but doing so in far lesser time.

To confirm this, we tested our approach on a set of both representative real-world simple directed and undirected complex networks with varied topological features. We compared our results with two base network-centric approaches. For completeness we included the results obtained for the exact approach that were already presented in Paredes and Ribeiro (2013). Furthermore, we test the approximate approach by comparing it with the exact

approach and with previous approaches, to showcase its behavior speed-wise, accuracy-wise and convergence-wise. We show that we obtain considerable speedups in both exact and approximate approaches, being roughly an order of magnitude faster than past methods which compute an isomorphism test per subgraph occurrence. The source code of the preliminary version of our algorithm is available at <http://www.dcc.fc.up.pt/gtries/fase/>.

The remainder of this paper is organized as follows: Section 2 defines the problem being solved and further describes some of the past approaches that are more relevant to our current work. Section 3 describes in detail our proposed exact methodology. Section 4 explores our approximate approach. Section 5 shows our experimental results. Section 6 concludes the paper and also gives some directions for future work.

1.1 Related work

In the network-centric realm, the two main base algorithms are ESU (Wernicke 2006) and Kavosh (Kashani et al. 2009). Even though they are conceptually similar, since they both work by iterating through all K -subgraph occurrences incrementally and in the end perform isomorphism tests, they use two underlying different approaches. Although their execution times are usually pretty close, past tests show that Kavosh performs slightly better on average. An improvement over these approaches is the very recent QuateXelero algorithm (Khakabimamaghani et al. 2013). It avoids having to do one isomorphism test per occurrence by storing the underlying topology of the subgraphs being enumerated in a quaternary tree. A contemporaneous algorithm with a similar methodology is our own work, FaSE (Paredes and Ribeiro 2013), which is the base algorithm for this paper's work. FaSE differs from QuateXelero because it uses a different underlying topological structure, the g -trie. Also, we supply a sampling version of FaSE, capable of providing faster approximate results while the current QuateXelero implementation only provides exact results. A different improvement approach is followed by NetMODE (Li et al. 2012), that considers only very small subgraph sizes and either caches the results of isomorphism tests or builds a customized isomorphism test or a particular subgraph size. Our work differs because we aim at a more complete generality, with no rigid restrictions on the subgraphs size.

Regarding subgraph-centric approaches, the work of Grochow and Kellis (2007) stands out. It works by taking a single subgraph type and computing its frequency on the input network by breaking symmetries. We would like to point out that this approach is conceptually different from the one taken in this work, since a full subgraph census

would require a separate computation per subgraph and pre-generated set of subgraphs.

As for the set-centric approach, the state-of-the-art is the usage of g -tries (Ribeiro and Silva 2014b), a work previously developed by us. Like in the subgraph-centric approach, this algorithm makes use of symmetry breaking conditions to enumerate not one, but a set of subgraphs. Note that the data structure used in this work is similar to these g -tries (and that is why we used the same name). However, our work does not use symmetry conditions and is network centric in its nature, thus does not requiring a pre-generated set of subgraphs to search for.

Another possible assumption is to only consider certain types of graphs and thus explore specific combinatorial features of that graph type, as was done in Marcus and Shavitt (2010). Our work differs from both this and the pre-calculation approach since it aims at generalness and applicability in all types of graphs.

For approximate approaches, one of the first to appear was Kashtan et al. (2004), an algorithm that provided a biased estimator by doing a random walk on the network. To correct the bias, it calculated the probability to sample each subgraph and used it to weight each sampled subgraph. As an extension of the ESU algorithm there exists Rand-ESU (Wernicke 2006), which works by placing probabilities in each level of the enumeration, thus giving an unbiased estimator for the number of subgraphs of each isomorphism class, similar to what was done on this work. Another extension of an exact method is Rand-gtries (Ribeiro and Silva 2010), which works in a similar fashion to Rand-ESU. A more recent approach is given by GUISE (Bhuiyan et al. 2012), which works using a Markov Chain Monte Carlo sampling method. However, it is also more specialized on a more specific census, namely undirected subgraphs of sizes 3–5. Our work differs because right from the start we aim towards total generalization and we support both directed and undirected networks of any size for which we have enough memory to store the subgraph classes. Finally, we should note that as in the exact algorithms, there are approximate approaches that are geared only towards certain subgraph types and try to exploit specific properties of those types. For instance, Fascia (Slota and Madduri 2013) provides an approximate count of non-induced tree-like subgraphs.

2 Preliminaries

2.1 Terminology and notation

To ensure consistency in the terminology throughout the paper, we will review the used notation. A *graph* G is composed of a set of vertices $V(G)$ and a set of edges

$E(G)$, represented by pairs $(a, b) : a, b \in V(G)$. We define the *size* of G , denoted by $|V(G)|$, as the number of vertices and we assume that all vertices are assigned consecutive integers from 0 to $|V(G)| - 1$. Furthermore, for two vertices u and v of a graph G , we write $u > v$ to denote that the label of vertex u is larger than the label of vertex v . A graph with size k is denoted as a k -graph. A graph G is called undirected if $\forall u, v \in V(G), (u, v) \in E(G) \leftrightarrow (v, u) \in E(G)$ and directed otherwise.

A subgraph G_k of a graph G is a k -graph where $V(G_k) \subseteq V(G)$ and $E(G_k) \subseteq E(G)$. This subgraph is *induced* iff $\forall u, v \in V(G_k) : (u, v) \in E(G) \leftrightarrow (u, v) \in E(G_k)$ and is called *connected* if all vertex pairs are connected by a sequence of edges. The *neighborhood* of a vertex $v \in V(G)$ is defined as $N(v) = \{u : (u, v) \in E(G) \vee (v, u) \in E(G)\}$ and similarly we define the neighborhood of a subgraph G_k of G , denoted as $N(G_k)$, as the set of all of the neighbors of vertices in $V(G_k)$ not included in G_k . The *exclusive neighborhood* of a vertex v in a graph G relative to a subgraph G_k is defined as: $N_{exc}(v, G_k) = \{u : u \in N(v) \wedge u \notin N(G_k) \wedge u \notin G_k\}$.

Two graphs G and H are said *isomorphic*, denoted as $G \sim H$, if there is a bijection ϕ between $V(G)$ and $V(H)$ such that $\forall u, v \in V(G) : (u, v) \in E(G) \leftrightarrow (\phi(u), \phi(v)) \in E(H)$. It is clear that the isomorphism relation is an equivalence relation and so we call each equivalence class an *isomorphism class*. For a particular k -subgraph, G_k , of a graph G , we denote the set of all subgraphs of G that belong to the same isomorphism class of G_k by $S(G_k, G)$ and we call *frequency* to the number of subgraphs of G that belong to that class and denote it as: $F(G_k, G) = |S(G_k, G)|$.

For a subgraph G_k of a graph G , an estimator of the value of $F(G_k, G)$ is denoted as $\hat{F}(G_k, G)$. It is called *unbiased* if its expected value, denoted as $\mathbb{E}(\hat{F}(G_k, G))$, is equal to $F(G_k, G)$ and *biased* if not.

2.2 Problem definition

We will now define more precisely the problem we are trying to solve:

Definition 1 (*Subgraph census problem*) Given an integer k and a graph G , determine the frequency of all connected induced k -subgraphs of G . Two occurrences of a subgraph are considered different if they have at least one node that they do not share.

It is important to notice that we are only concerned with subgraphs that are both connected and induced. Note also how we distinguish occurrences. Other frequency concepts do exist and have been tested (Schreiber and Schwobbermeyer 2004), but here we use the standard definition. This has direct implications on the number of existing subgraphs, with no downward closure on the frequencies, since

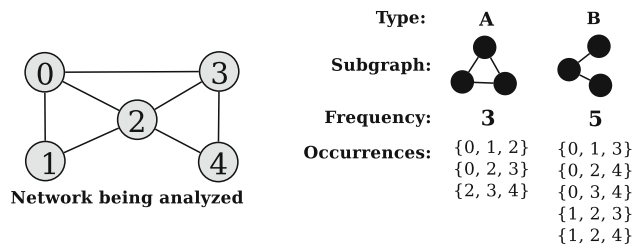


Fig. 1 An example 3-subgraph census

a subgraph may appear more times than a subgraph contained in it. Figure 1 exemplifies a subgraph census for $k = 3$ in a graph G with five nodes.

2.3 Base network-centric enumeration algorithms

In this section, we will discuss in some detail the previous enumeration approaches that are more relevant to this work, namely the ESU and Kavosh algorithms, which constitute the two core network-centric enumeration algorithms that are inclusively used by other methods. For instance, Quatexelero uses ESU as the underlying enumeration algorithm, while NetMODE resorts to Kavosh. Furthermore, we will discuss how the ESU algorithm is used as an approximate algorithm in the Rand-ESU approach (Kavosh has not been extended to a sampling version).

2.3.1 ESU

The ESU algorithm works by enumerating all k -subgraphs of a network and in the end performing an isomorphism test per enumerated occurrence. The enumeration step is thus the most important one and the breakthrough it brought was the ability to enumerate all occurrence once and only once.

It keeps two vertex sets, which we will call V_S and V_E . The former represents the subgraph being currently enumerated and since we are enumerating induced subgraphs, we only require a vertex list. The latter is a list of vertices that neighbor any vertex in the current subgraph and can be added to the subgraph being enumerated, that is V_S .

Initially, it sets $V_S = \{v\}$ for each vertex v in the input network G and $V_E = N(v)$. Then, for each vertex u in V_E , it removes it from V_E and makes $V_S = V_S \cup \{u\}$, effectively adding it to the subgraph being enumerated and $V_E = V_E \cup \{u \in N_{exc}(u, V_S) : u > v\}$ (where v is the original vertex to be added to V_S as stated in the beginning of the paragraph). The N_{exc} here makes sure we only grow the list of possibilities with vertices not already in V_S and the condition $u > v$ is used to break symmetries, consequently preventing any subgraph from being found twice. This process is done several times until V_S has K elements,

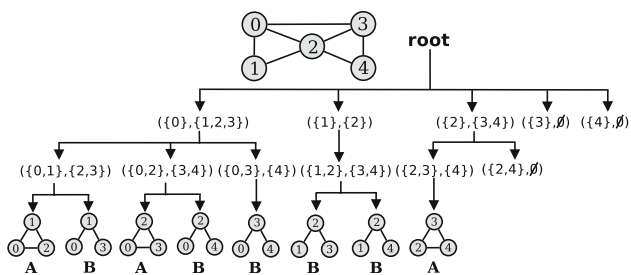


Fig. 2 An example induced ESU search tree leading to eight different 3-subgraphs occurrences

which means V_S contains a single occurrence of a K -subgraph.

Since ESU works recursively in a set incrementation fashion, it creates an implicit recursion search tree. In each node, we consider a certain V_S and V_E representing the partially (or fully if it is a K -subgraph) enumerated subgraph. Note that this feature that ESU displays of incrementing a set of vertices will be very important for our own algorithm in the next section. Figure 2 exemplifies this implicit enumeration tree for a 3-subgraph census.

After the enumeration, the third-party *nauty* (McKay 2012) algorithm is used for isomorphism testing, so that each occurrence is attributed to the correct isomorphism class and the respective frequency is increment.

2.3.2 Kavosh

Like ESU, the core idea of the *Kavosh* is to find all subgraphs that include a particular vertex, then remove that vertex and continue from there iteratively. It differs, however, because it builds an implicit tree rooted at the chosen vertex (with tree children being network neighbor vertices), and then generates all combinations with the desired number of nodes. For instance, if we are searching for 3-subgraphs, and considering that at the tree root level we can only have one vertex, we could have the combinations with pattern 1–2 (one vertex at root level 0, two vertices at level 1) or with pattern 1–1–1 (one vertex at root level 0, one at level 1 and one at level 2). In an analogous way, 4-subgraphs would lead to patterns 1–1–1–1, 1–1–2, 1–2–1 and 1–3. Figure 3 exemplifies this combinatorial search, by showing all patterns emerging from a single root node.

The combinations are done using a *revolving door* algorithm (Kreher and Stinson 1999) and as in ESU the isomorphism detection is done using *nauty* (McKay 2012)

2.3.3 Rand-ESU

The approximate version of ESU is very similar to the exact one. The idea behind it is very similar to the one we

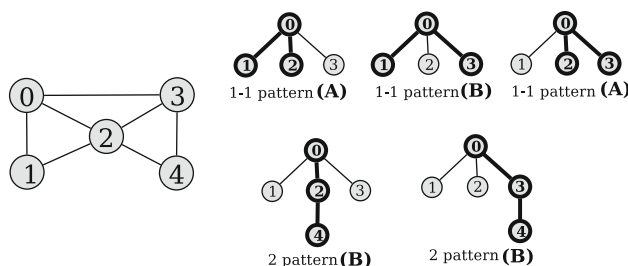


Fig. 3 *Kavosh* combinatorial search tree starting on node 0 leads to five different 3-subgraphs occurrences

will present on Sect. 4 since the underlying structure of both algorithms is very similar.

For each level of the enumeration tree, the algorithm places a probability of descending, meaning it will only go on exploring that branch with that particular probability. This results in only a fraction of all subgraphs occurrences being enumerated, where each occurrence is sampled with the same probability (we will address and prove this later). Thus, it is possible to have an unbiased estimator for the number of occurrences in each isomorphism class.

3 Exact subgraph census

Our proposal to address the subgraph census problem was presented in (Paredes and Ribeiro 2013). We tried to explore the underlying structure of networks to decrease the amount of computation needed to classify each occurrence in its isomorphism class. The idea is to separate all occurrences in intermediate classes that have two important properties: they can be calculated quickly and each occurrence in the same intermediate class is in the same isomorphism class. Following the complete enumeration of the subgraphs in the network, it is only necessary to compare a single representative subgraph per intermediate class, hence decreasing the number of isomorphism tests required.

To accomplish that, our algorithm, FaSE (from FAST Subgraph Enumeration), is composed of two processes, closely integrated with each other: enumeration and encapsulation. The former pertains to the fundamental process of actually finding each individual occurrence of a subgraph in the original network. This is required to be done by an incremental growth of a connected set of vertices. The encapsulation process is where the isomorphism classes are obtained by storing the topological features of the subgraph. Whenever a vertex is added to the current set of enumerated vertices, we generate a label that describes the relation of the newly added vertex to the already added ones. This corresponds to the partitioning in intermediate

classes mentioned above. To actually accomplish this, we use a generic process we called LS-Labeling that categorizes each subgraph intermediate class. The actual storage of the labels and subgraphs is done using a tree data structure that acts as a customized g-trie in which the LS-Labeling works as the divider, that is, it is responsible by the tree's edges. The following sections describe these techniques thoroughly.

3.1 Subgraph enumeration

The enumeration process is not constrained, it allows for different approaches. As long as it counts every occurrence of each subgraph once and only once and provided that it does so in an incremental fashion, meaning node by node, any process is allowed. The goal here is to enforce that the process transitions from state to state adding a single new node at a time. This permits that each enumerated subgraph is labeled according to the transitions it took to reach the final state.

Consequently, it is possible to use any modern enumeration algorithm. As described above, two of best that accomplish this task are ESU (Wernicke 2006) and Kavosh (Kashani et al. 2009) and they both can be integrated in FaSE since they follow the required behavior.

3.2 Encapsulating isomorphism information in a tree

As the enumeration process is running, we need to record the data collected. The reason to do so is to take advantage of the topology of subgraphs, which in practice is separating the subgraphs into said intermediate classes. Thus, a data structure that is adapted to the behavior of the enumeration step, but also compact and benefiting from the common topology given by the labels is required. Thus, a good candidate that follows these parameters and fits to the idea of hierarchical construction of the enumeration is a g-trie. The actual data structure used is based on our previous work with g-tries, which can be thought of as “prefix trees of graphs”, although FaSE's setup is somewhat altered. To avoid ambiguity, throughout the rest of this paper, we will use *nodes* to refer to tree nodes (in our g-trie) and *vertices* to network and subgraph vertices.

3.2.1 G-tries

The custom g-trie works as a tree whose nodes represent graphs. This is done in an order that respects the topology of the subgraphs, meaning if a certain node is parent of another node, then the graph represented by the former is a subgraph of the latter (in this particular case, with only

one additional vertex). Each node stores two pieces of information: a frequency, which is the number of subgraphs of the original network that are of that particular type; a label information regarding its topological structure. The idea is to start off with an empty graph and sequentially add new vertices. For each vertex added, a label that portrays its relation with the previous added vertices is calculated and used to determine its node on the g-trie. Each vertex addition follows a new node on the g-trie. Note that this is a deterministic process, meaning that if the same subgraph is added twice the resulting label is the same. In terms of the g-tree correspondence, the calculated label establishes the node to follow (and the due edge). If this node is nonexistent, both the node and the edge are created. As a result, if two different subgraphs are processed and end up on the same g-trie node, it is assured that they are isomorphic, thanks to the label requirements. An example g-trie can be visualized in Fig. 4.

Regarding how the g-trie actually accomplishes this, it works by keeping a current node that represents the partial subgraph being enumerated (partial since it is being enumerated), which is initially the root node (corresponding to the empty graph). It uses two procedures to progress: Deepen and Jump. The first one inserts a new vertex into the current graph by moving along the g-trie to the corresponding node, a process which lowers the current node (“deepening”). Additionally, it creates the new node and edge if they were previously nonexistent and augments the frequency count of that particular node by one. It uses the label generated for the added vertex, which is assigned to a determined edge, to decide where to go in the tree. This is implemented using a prefix tree (or “trie”) to ensure linear

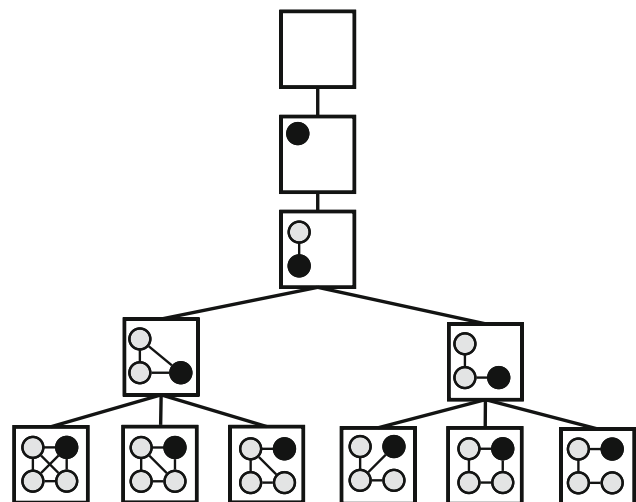


Fig. 4 An example g-trie with some graphs up to 4 vertices. Dark vertices newly added vertices

time search of the new node on the length of the label. Contrary to this, the `Jump` procedure sets the current vertex to its parent, thus going up in the *g*-trie.

To actually insert graphs into this *g*-trie, it is possible to take advantage of the common topologies inherent on the enumeration of the subgraphs. Whenever a new vertex is selected by the enumeration process, the labeling algorithm assigns a new label to this vertex in relation to the already selected ones and uses this information to perform a `Deepen` operation on the *g*-trie. After the recursive call made to enumerate all the subgraphs that exist from the current subgraph, a `Jump` call is performed to go back to the previous node in the *g*-trie. The reason this works (and why it is done) is since all subgraphs achieved from a particular state (corresponding to a node in the *g*-trie) will share a common topology related to the partial enumerated set (the state) and therefore share the same label information to that point.

Summarizing the previous paragraphs, it is possible to conclude that this setup is the one of a simple tree regulated by the labels assigned in each step. The consequence of this is that it ends up representing graphs simply because the label is designed in that way. Hence, this is a very general data structure adaptable to different labeling algorithms.

3.2.2 LS-labeling

The generic labeling algorithm is called LS-Labeling. As already mentioned, it is the core of the *g*-trie and it is also directly related with the branching factor of the tree since it governs the different edges, thus it is associated with both the algorithm running time and the used memory. It acts under a pair of conditions namely that it deterministically partitions the different subgraphs in a class created by the LS-Labeling are in the same isomorphism class, and that it does so incrementally (emulating the behavior of the enumeration step) using only information regarding the newly added vertex and its relationship with the already added ones. From these conditions, one could idealize that this labeling algorithm could simply be a procedure that actually calculated isomorphisms, thus rendering the point of the tree useless. However, as was said throughout the paper, this is a computationally hard problem and so its use is exactly what we are trying to avoid. Thus, it makes sense to ensure another condition: that the algorithm runs in polynomial time. This behavior sets up a trade off regarding the time spent labeling the various subgraphs and the time spent on the actual *g*-trie (which includes the final isomorphism test time).

In our past work, we described two intuitive labeling algorithms which are called the “adjacency list” label and

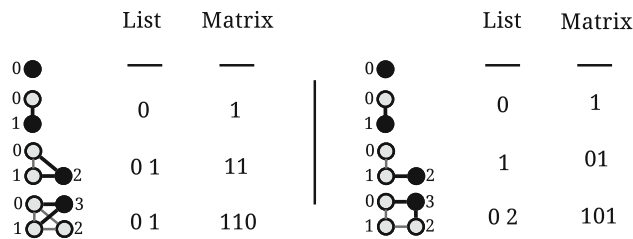


Fig. 5 Two different valid LS-Labeling schemes on two example graphs. *Dark vertices* are the ones being added

the “adjacency matrix” label, coming from the corresponding graph data structures. We show an example of both labels in Fig. 5. When a new vertex is added, the algorithms act on the current subgraph and the vertex to be added. For simplicity, we will consider the undirected case first when adding the *k*-th vertex and then distinguish the directed one. In the case of the adjacency list, the label corresponds to a ordered list of at most *k* – 1 integers where the value *i* ($0 < i < k$) is present if there is a connection from the newly added vertex to the *i*-th added vertex. Similarly, in the adjacency matrix case a list of *k* – 1 Boolean values is kept, each one indicating if there is a connection between the newly added vertex and each vertex added before in order of addition, which corresponds to a segment of the actual adjacency matrix of those vertices. This method scales pretty easily to the directed case, where instead of just keeping one list, in both cases we keep two, one pertaining to the ingoing connections and the other two the outgoing (in practice, a separator value is also used on the adjacency list case to separate the ingoing from the outgoing list). We show a visual representation of a *g*-trie with the labels associated with each edge using the “adjacency list” label in Fig. 6.

To prove the correctness of these two labels options, first notice that they are methodically equivalent and only change the way they represent the information. Thus, to prove its correctness, it suffices to show that two subgraphs labeled equally belong to the same isomorphism class. To show that we need to find a bijection between the two subgraphs. This is simple enough by following the order in which each vertex was enumerated, which is implicitly represented on the actual label, and map the vertex in each position of the order to one another. Hence, any two subgraphs labeled equally belong to the same isomorphism class and we have our correctness proof.

We are aware that there are more possibilities for this operation that we did not previously address. One of them is what we called the “*n*-th-neighbor” label, which instead of simply considering the connections between the already added vertices and the newly added, also considers the

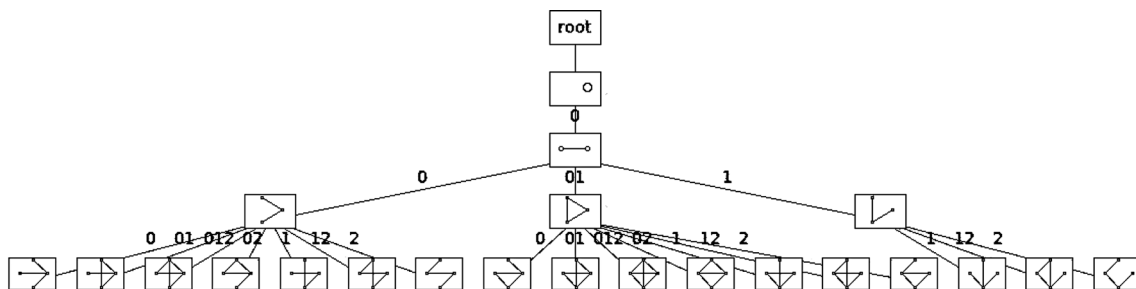


Fig. 6 An example g-trie with list LS-Labeling after searching for 4-subgraphs

connections from the nodes at distance of a maximum of n from the corresponding vertices. Obviously, these connections augment exponentially and if they are all considered it corresponds to a full isomorphic label. However, a simple “2nd-neighbor” label could, in some cases where the subgraph fingerprint is more heavily populated with certain subgraphs, decrease the run time and memory used. However, since this method is not so simple as the previous ones, it would probably have greater costs on the general case and thus we did not experiment with it.

Note also that since the LS-Labeling is being used as an intermediate classifier, the g-trie will end up having more leaves than there are different isomorphism classes. This could affect the overall run time (since we need to perform an isomorphism test per intermediate class), however, in the case of both the adjacency list and matrix label, the number of leaves is directly correlated to the different automorphisms of a same graph. Thus, it ends up being just a fraction of the total number of occurrences in any practical example and so there is a significant gain of computation time.

We conclude this section by highlighting the flexibility the LS-Labeling generic algorithm displays. Since it only enforces a small number of conditions, it allows for the trade off referred earlier to be adjusted by changing the type of LS-Labeling. Perhaps more importantly, it is adaptable to different formulations of the problem, as was possible to observe with the case of directed graphs. The algorithm is still the same, but the labeling is tuned to suit this particular instance. So it can be extended to other problem formulations such as colored graphs, weighted graphs or even multigraphs.

3.3 The FaSE algorithm

We present an overview of the whole FaSE in Algorithm 1. This incorporates the enumeration step, the g-trie and the LS-Labeling. We use the expression += to denote “increment by a value”.

Algorithm 1 The FaSE Algorithm

Input: A graph G and a subgraph size k
Result: Frequencies of all k -subgraphs of G

```

1: procedure FASE( $G, k$ )
2:    $EnumerateAll(G, k, \emptyset, 0)$ 
3:   for all  $n$  in  $GTrie.leaves()$  do
4:      $frequency[CanonicalLabel(n.Graph)] += n.count$ 

5: procedure ENUMERATEALL( $G, K, S, d$ )  $\triangleright S$ :subgraph;
    $d$ :depth
6:   if  $d = K$  then
7:      $GTrie.current.count += 1$ 
8:   else
9:     while  $nS \leftarrow EnumerateNext(S)$  do
10:       $w \leftarrow nS.NextNode()$ 
11:       $nL \leftarrow LSLabel(S, w)$ 
12:       $GTrie.Deepen(nL)$ 
13:       $nS.Subgraph \leftarrow nS.Subgraph \cup w$ 
14:       $EnumerateAll(G, K, nS, d + 1)$ 
15:       $GTrie.Jump()$ 

```

This algorithm puts together all the discussed parts of FaSE. The procedure `EnumerateAll()` iterates through all subgraphs of all sizes to K , incrementing the counter when the size is K . The frequencies are stored internally by the g-trie, however, since the LS-Labeling does not give the final classes, it is necessary to accumulate the results from each g-trie node and perform an isomorphism test to a representing graph. In the original implementation, we do so resorting to `nauty` (McKay 2012), a third-party efficient isomorphism toolkit, although any algorithms that create a canonical label (that is, a label that represents isomorphism classes) will work.

Note also that in our original implementation (and in any practical implementation) we hard coded the enumeration step into the `EnumerateAll()` function to increase efficiency and explore low-level features of the algorithm.

4 Approximating a subgraph census

In this section, we will explore this work’s contribution, an interesting feature of the FaSE algorithm, namely that it can be adapted to an approximation algorithm to estimate the frequency of each subgraph type in a network by obtaining a sample of subgraphs. It is possible to tune the algorithm to trade accuracy for time, which allows it to be run in a wider range of real networks, which are usually too large for a complete exact enumeration for higher subgraph sizes. The actual method we use is very similar to the one presented in Wernicke (2006) but we will provide our analysis and discussion.

Since each subgraph is enumerated once and only once in the exact version, we can use that to only find a sample of the total number. To do so, we will introduce a probability p_d at each depth d (d varies from 0 to $K - 1$, where K is the desired size of the enumerated subgraphs) of the enumeration (which can conceptually be easier to imagine in the g-trie). To clarify the previous sentence, the depth here is the order of the vertex being currently added to the partial set, which is equivalent to the size of the partially enumerated subgraph. The idea is to instead of always processing each newly enumerated vertex (which corresponds to lines 10–15 in Algorithm 1), do it with probability p_d at each level p_d .

We can easily observe that the probability of a particular subgraph on the network being sampled is the probability of the first vertex being chosen (at level 0) which is p_0 times the probability of the second vertex being chosen and so on, which equals $\prod_{0 \leq d < K} p_d$. We will call this value *sampling percentage*, and denote it as: $p_s = \prod_{0 \leq d < K} p_d$.

We denote the total number of subgraphs of size K (the leaves in the induced ESU search tree) in graph G by $T(G)$. It is possible to show that the average number of sampled subgraphs is $p_s \times T(G)$. To prove so, first note that each K -subgraph has the same probability of p_s of being sampled. Since there are $T(G)$ subgraphs and each one has a probability of p_s of being sampled, the average number of sampled subgraphs is $p_s \times T(G)$.

We will call $Fsample(G_k, G)$ to the frequency of subgraphs of G sampled by the algorithm that are from the same isomorphism class as G_k . This definition allows us to define an estimator for the value of $F(G_k, G)$ as follows:

$$\widehat{F}(G_k, G) = \frac{Fsample(G_k, G)}{p_s}$$

Note that since all the isomorphism classes are disjoint, to obtain an estimator for the total number of subgraphs it suffices to sum all the $\widehat{F}(G_k, G)$, one per different isomorphism class.

4.1 Uniform sampling

To start the theoretical discussion of the approximation, we will first prove the estimator is an unbiased estimator. To do so, observe that since the probability of sampling each subgraph is the same, p_s , the expected value of $Fsample(G_k, G)$ is simply $p_s \times F(G_k, G)$.

To calculate the expected value of $\widehat{F}(G_k, G)$, we observe that since the expected value is a linear operator, this corresponds to the previously calculated value divided by p_s . Plugging this into the formula of the estimator gives:

$$\mathbb{E}(\widehat{F}(G_k, G)) = \frac{\mathbb{E}(Fsample(G_k, G))}{p_s} = F(G_k, G)$$

Thus, we conclude that $\widehat{F}(G_k, G)$ is an unbiased estimator for $F(G_k, G)$.

Using this information, Algorithm 2 shows the adapted algorithm, which from now on we will call Rand-FaSE to distinguish from the exact version of FaSE.

Algorithm 2 The Rand-FaSE Algorithm

Input: A graph G and a subgraph size k

Result: Frequencies of all k -subgraphs of G

```

1: procedure FASE( $G, k$ )
2:   EnumerateAll( $G, k, \emptyset, 0$ )
3:   for all  $n$  in GTrie.leaves() do
4:     frequency[CanonicalLabel( $n$ .Graph)] +=  $n$ .count

5: procedure ENUMERATEALL( $G, K, S, d$ )  ▷  $S$ :subgraph;
    $d$ :depth
6:   if  $d = K$  then
7:     GTrie.current.count += 1
8:   else
9:     while  $nS \leftarrow$  EnumerateNext( $S$ ) do
10:      with probability  $p_d$  do
11:         $w \leftarrow nS$ .NextNode()
12:         $nL \leftarrow$  LSTLabel( $S, w$ )
13:        GTrie.Deepening( $nL$ )
14:         $nS$ .Subgraph ←  $nS$ .Subgraph ∪  $w$ 
15:        EnumerateAll( $G, K, nS, d + 1$ )
16:        GTrie.Jump()
17:
```

Note that in all practical implementations the actual probability call should be hard coded, since it can prevent some unneeded work done in the *EnumerateNext* () function.

4.2 Performance analysis

To continue, we will reason about the variance of the estimator and how the choice of each individual value of p_d affects it and thus the quality of the estimation.

First of all, notice that the number of subgraphs sampled of a certain type depends on the structure of the enumeration tree. If it were perfectly balanced and each subgraph type evenly distributed along the tree, then the individual values would not matter but only their product (what we called of sampling percentage). However, this is not the case in any of the presented enumeration algorithms. Even though for instance the ESU enumeration tree is naturally skewed since it enforces an order on the enumeration, it is highly unlikely that any algorithm generates a balanced enumeration tree since this is very input dependent.

Since the enumeration tree is not balanced, the choice of parameters influences the quality of the sample and run time. If lower values for p_d are chosen for levels of the tree nearer to the root, this will increase the variance of the results, since it is possible to branch out a sub-tree with more occurrences of a certain type. However, the run time of the algorithm is decreased in exchange for the augment of variance. This decrease is twofold: on one hand, the amount of subgraphs sampled has a higher variance, which results in fluctuations in run time; on the other hand, since a subgraph that is not going to be sampled is pruned earlier in the tree, we can avoid most work on its partial enumeration, which is costly since it involves traversing the g-trie, generating its label through the LS-Labeling and doing the actual enumeration.

A consequence of the unbalance of the enumeration tree is that even if given the values for p_d , calculating the variance is hard since it is highly dependent on the input network. It is possible to draw some conclusions though, the most important one being that the variance is higher in relative value for lower $F(G_k, G)$ values. To explain this recall that the average number of sampled subgraphs in $S(G_k, G)$ is: $p_s \times F(G_k, G)$. When this value is small (specially when it approaches 1 or is smaller than 1) since the number of sampled subgraphs is a discrete quantity, the actual value of $F_{sample}(G_k, G)$ is going to be rounded down or up. This means the variance will be higher in relative value, since for high values of $F(G_k, G)$ the continuous approach is a good approximation.

There are ways of decreasing the variance while keeping the estimator unbiased. In Wernicke (2006), the author suggests instead of simply continuing with a certain probability, from a node (of the enumeration tree) with x children at depth d randomly choose $x' = \lceil x \cdot p_d \rceil$ with probability $x \cdot p_d - \lfloor x \cdot p_d \rfloor$ or choose $\lfloor x \cdot p_d \rfloor$ with probability $(1 - (x \cdot p_d - \lfloor x \cdot p_d \rfloor))$. The idea is to choose a fixed number of children instead of taking each one with a certain probability, ensuring that there is always a collection of nodes that will be followed. The author also showed that this leads to a lower variance. In our

implementation, which we will discuss on the next section, we did not include this because even though this improves the quality of the sample on average, for lower values of $F(G_k, G)$ it can decrease, particularly when $\lceil x \cdot p_d \rceil$ rounds to 0, where depending on the input network, the algorithm would not sample any subgraphs of a certain isomorphism class.

In the next section, we will provide some experimental results that will help in understanding all the important features and behavior of the sampling algorithm, namely run time, performance and convergence.

4.3 Further discussion

To conclude the discussion about the sampling we will mention two important aspects regarding the sampling's application and how to improve it.

Naturally, the main purpose of doing a sample in place of a full enumeration is to use it in inputs that would take too much time to calculate using the latter approach. On these cases are included networks with high number of vertices and edges. Therefore, the data structure used to represent the network cannot be a simple adjacency matrix, since it would draw too much memory and thus would be unfeasible. The obvious substitute is an adjacency list, but due to the fact that FaSE requires a way of knowing if two certain vertices are connected (in the LS-Labeling and in the isomorphism test), the adjacency list will hurt time performance compared with the simple matrix (which implements this operation in constant time). To improve the operation of finding out if two vertices are connected, we experimented numerous alternatives like keeping the neighbor list of each vertex ordered and then performing a binary search to find if a vertex is in the list (we can do this since we never delete vertices from the graph). Another method we tried out turned out to give better results on all networks we tested, this method was to keep a hash table as a neighbor list with a simple hash function of taking the vertex label of each neighbor modulus a constant times the number of neighbors of each vertex. We could slightly improve the results by keeping a cache of a small number of recent queries and reporting the result immediately if positive.

Another aspect that could improve the quality of the sample would be to automatize the choice of the individual probabilities p_d . This could be achieved through an adaptive sample, that would start out with very low parameters and over multiple runs only explore the enumeration tree where needed. This works since for high values of $F(G_k, G)$ the estimator result converges rather quickly whereas for lower values it does not. So exploring this could significantly improve the sample.

5 Experimental results

To evaluate the performance of both proposed algorithms, *Rand-FaSE* was implemented in *C++* using *ESU* as the base enumeration algorithm. All tests were performed on a Linux machine with an Intel Core 2 6600 (2.4 GHz) and 2 GB of memory.

We implemented both the adjacency list and matrix *LS-Labeling* methods, but the two had very similar execution times, although the list method ended up having slightly better results most of the time, so we opted to only show the results obtained using it. As stated previously, we used the third-party tool *nauty* (McKay 2012) to efficiently perform the isomorphism classifications.

We used a varied set of undirected and directed networks. In all networks weights, self-loops and multiple edges were either ignored or nonexistent.

To provide a comparison measure, we included a summary of the experimental results obtained by the Exact approach in Paredes and Ribeiro (2013) and its analysis.

We will first present the results regarding the Exact approach and follow it with the Approximate approach results.

5.1 Exact approach results

To test the performance of the Exact approach, we ran *FaSE* with different subgraph sizes and different networks and compared the execution times to *ESU*, through its publicly available tool and *Kavosh*, through its original source code. We chose these algorithms since at the time they were the main previous approaches. The networks used are summarized in Table 1.

The time each algorithm took to perform a complete *K*-subgraph census on all networks was measured, with *K* varying from 3 to 9. Due to time constraints, we only show execution times up to 5 h. All the results as well as statistics about the number of subgraphs per network and how many leaves of *FaSE*'s *g*-trie used are shown in Table 2.

Analyzing the results, the general trend is that *FaSE* obtains better results in all setups than both *ESU* and

Kavosh, as was expected. Moreover, it was always an order of magnitude faster, except for a couple of outliers. Another observation in order is that there is a tendency for the speedup to increase as the *K* increases, which means there is a larger speedup in setups where the total execution time is higher, which are the ones where a faster algorithm is more critical. This is a sensible outcome, since the speedup comes mainly from the isomorphism tests avoided, which is directly related to the ratio between the total number of subgraphs and number of *g*-trie leaves and this is a quantity that generally increases for smaller subgraph sizes and larger networks (as is possible to observe in the results table). The actual values are very much network dependent and there is no "external" measure (number of nodes, edges...) that allows a prediction of the actual execution times in any order of accuracy, since it heavily depends on combinatorial features of the network.

It is also important to notice that the major bottleneck of a subgraph census is the isomorphism testing, which is what the algorithm aims to improve. To check that *FaSE* addresses this and is not a somehow faster implementation of the *ESU* algorithm it was ran without the *g*-trie functionality, simulating the actual *ESU* algorithm functioning. The result proved to be slightly better than *FanMod*, but was still roughly an order of magnitude slower than *FaSE*'s normal functioning. Furthermore, contribution of the enumeration process was compared to the final execution time by running the algorithm without the isomorphism tests, meaning only running the enumeration algorithm. Obviously, this does not allow to compute the actual census. The results indicate that the actual enumeration is only a tiny fraction of the whole execution time, confirming what we stated above.

The final aspect we want to highlight is that the number of leaves used by the *g*-trie has a heavy influence on the memory used by the algorithm. This implicates that it is impossible to run it with much larger subgraph sizes than the ones tested in this work. Even though the super exponential growth of the number of subgraph types makes it impossible to even store the individual frequencies of each type, this is still prohibitive and actually potentially slightly affects the execution time.

Table 1 Complex networks used in the exact tests

Network	Directed	Nodes	Edges	Avg. degree	Type	Source
StarWars	No	51	157	3.08	Social	Our own (Paredes and Ribeiro 2013)
Jazz	No	198	2742	13.85	Social	Arenas (Gleiser and Danon 2003)
Neural	Yes	297	2359	7.94	Biological	Newman (Watts and Strogatz 1998)
Foldoc	Yes	13,356	120,700	9.04	Semantic	Pajek (Batagelj and Mrvar 2006)

Table 2 Detailed experimental results for the four networks used for the exact setup

Network	K	Subgraphs found		FaSE		ESU		Kavosh	
		Types	Occurrences	Time (s)	Leaves	Time (s)	Speedup	Time (s)	Speedup
StarWars	3	2	1449	<0.01	3	<0.01	–	<0.01	–
	4	6	12,958	<0.01	17	0.04	23.5	0.03	17.6
	5	21	98,426	0.01	171	0.39	30.7	0.21	16.5
	6	106	630,369	0.08	2,406	3.12	38.0	1.90	23.1
	7	699	3,445,808	0.58	26,692	21.95	38.0	13.26	23.0
	8	5601	16,320,648	3.55	203,687	133.34	37.6	78.18	22.0
Jazz	3	2	67,414	<0.01	3	0.14	31.8	0.06	13.6
	4	6	1,833,618	0.15	17	4.24	28.9	2.55	17.4
	5	21	49,500,654	4.65	171	143.64	30.9	89.3	19.2
	6	112	1,266,953,062	140.84	3,113	3,630.00 ^b	25.8	2,912.43	20.7
	7	853	30,166,157,456	3,946.81	106,417	>5 h	–	>5 h	–
	8	5601	16,320,648	3.55	203,687	133.34	37.6	78.18	22.0
Neural	3	13	47,322	0.01	45	0.09	16.7	0.04	7.4
	4	197	1,394,259	0.13	1,846	2.21	17.5	1.71	13.5
	5	7072	43,256,069	4.73	76,214	102.14	21.6	91.03	19.3
	6	286,376	1,309,307,357	170.96	2,499,645	4,420.00 ^b	25.9	4,636.43	27.1
Foldoc	3	13	2,553,830	0.35	45	3.97	11.2	2.17	6.1
	4	198	228,272,189	27.80	2,304	903.39	32.5	308.78	11.1
	5	8345	29,621,881,964	3,735.20	141,115	>5 h	–	>5 h	–

^a FanMod accepts only 8 as the maximum subgraph size

^b Overflow problem in its own reported enumeration time and so we used elapsed time

5.2 Approximation approach results

We divided the tests for the Approximation approach into three sections with different aims. We first compared it with the Exact approach to assess the accuracy of the approximation for different sampling values. Then, we compared it with the previous work by testing how many subgraphs were sampled per second, so as to evaluate the time efficiency of the approximation. Finally, we tested how the approximation converges to the exact values by measuring the error and standard deviation displayed through various sampling percentages.

We used additional networks on the tests regarding the Approximation approach. We summarized the networks we used in Table 3. Note that we repeated some networks used in the previous section, but we included them in this table for completeness.

Since our algorithm requires choosing the multiple probabilities per level, p_d , we opted for the following three setups that explore the sampling properties differently:

High: $p_0 = 1, \dots, p_{K-3} = 1, p_{K-2} = p_s, p_{K-1} = 1$

Medium: $p_0 = 1, \dots, p_{K-4} = 1, p_{K-3} = \sqrt{p_s}, p_{K-2} = \sqrt{p_s}, p_{K-1} = 1$

Low: $p_0 = 1, p_1 = \sqrt[2]{p_s}, \dots, p_{K-2} = \sqrt[2]{p_s}, p_{K-1} = 1$

Note that we always considered p_0 and p_{K-1} to be 1 since due to the way we implemented our algorithm having $p_{K-1} \neq 1$ means it will do all the work enumerating a certain subgraph and then discard it with probability $1 - p_{K-1}$ and having $p_0 \neq 1$ means discarding a whole branch of the enumeration recursive tree, which means a whole isomorphism class could be discarded.

5.2.1 Comparison with the exact approach

To compare the Approximate approach with the exact one, we first ran Rand-FaSE with two different input networks, Yeast and Metabolic, to different sampling percentages. We used these two for this particular test since they are average sized directed and undirected networks and so allow us to perform more time demanding tests that would otherwise be unfeasible on larger networks. To measure the accuracy of the approximation, we calculated the percentage of isomorphism classes correctly estimated by the algorithm and considered the frequency of an isomorphism class to be correctly estimated when the approximated value is within 15 % of the real value (calculated through the Exact approach) for the three sampling setups described above (high, medium and

Table 3 Complex networks used in the approximation tests

Network	Directed	Nodes	Edges	Avg. degree	Type	Source
Jazz	No	198	2742	13.85	Social	Arenas (Gleiser and Danon 2003)
Yeast	No	2361	6646	2.81	Biological	Pajek (Batagelj and Mrvar 2006)
AstroPh	No	18,772	198,050	10.55	Social	SNAP (Leskovec et al. 2007)
Metabolic	Yes	453	2025	4.47	Biological	Arenas (Gleiser and Danon 2003)
Foldoc	Yes	13,356	120,700	9.04	Semantic	Pajek (Batagelj and Mrvar 2006)
Neural	Yes	297	2359	7.94	Biological	Newman (Watts and Strogatz 1998)

low). We did not consider isomorphism classes where the expected number of subgraphs sampled is smaller than 10, the reason being that in these cases the error associated would be too large to estimate the real value in any practical scenario.

The obtained results were graphed in a log plot displayed in Fig. 7. Excluding a few outliers, both plots are approximately a line (the first one eventually converges to 100 % correctness). Since this is a semi-log plot, this means that it is approximately a logarithmic function, that is, multiplying by 10 the number of samples should roughly double the correctness of the approximation. Of course this result is dependent on the way we measured correctness and thus is not fit for all scenarios.

Another observation to make is that, as expected, since the high setup places the probabilities in lower levels it should have a lower variance, which results in overall better results. Likewise, since the low setup distributes the probabilities more evenly it has the highest variance and obtains the overall worse results. There were a few outliers on the lower probabilities, but it was probably due to the fact that for lower sampling percentages the variance is obviously higher and so there are a lot more fluctuations in the results.

To have a better understanding of how the sampling works for individual isomorphism class sizes, we ran the algorithm with the network `foldoc` for different sampling percentages and measured the relative error to the real value. We used the `foldoc` network to showcase this since it is a rather dense network and thus for the particular subgraph size chosen, it has at least a subgraph of each existing type. Thus, it is clear how our algorithm behaves for lower and higher frequency subgraph classes. The results are showcased in Table 4. Taking a close look at the table confirms that the relative error for isomorphism classes with fewer subgraphs is higher, specially for the smaller sampling percentages. Furthermore, for the 0.1 % there were even isomorphism classes that did not get any subgraphs sampled at all.

5.2.2 Comparison with the previous approaches

Comparing our algorithm with the previous approaches was done by analyzing the speed performance. We compared our algorithm with `Rand-Esu`, the approximated version of the `ESU` algorithm. Since the functioning of `Rand-Esu` is conceptually similar to `Rand-FaSE` and

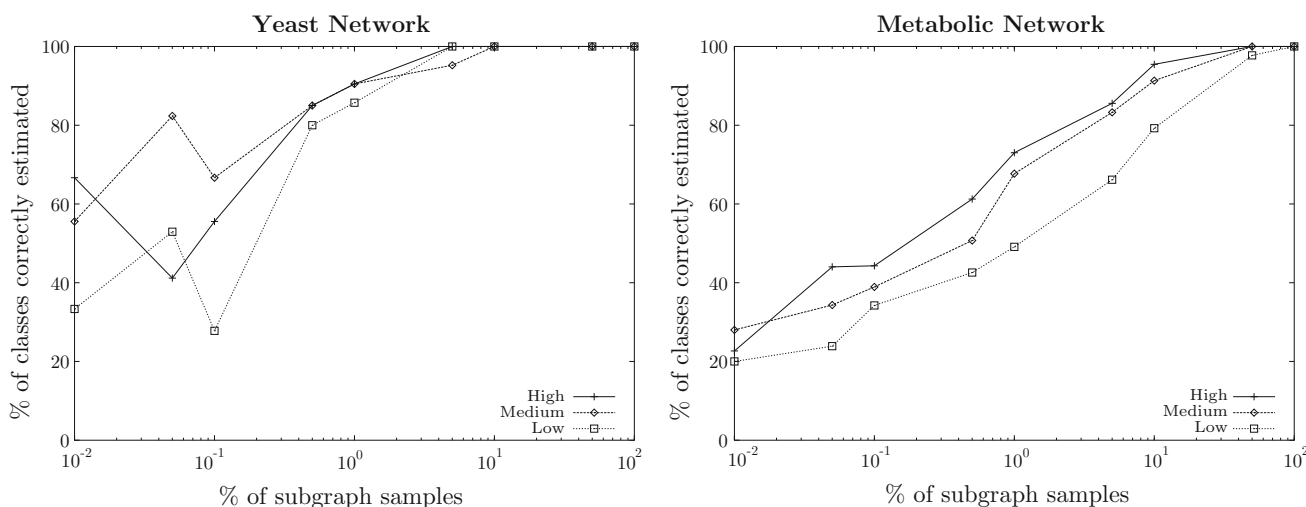















Fig. 7 Accuracy of Rand-FaSE for the undirected Yeast Network and directed Metabolic Network for size 5 subgraphs

Table 4 Results obtained for different sampling percentages for the directed `Foldoc` Network for size 3 subgraphs with setup *high*

Subgraph type	Number Exact	Number 50 % Sample	Error (%)	Number 10 % Sample	Error (%)	Number 1 % Sample	Error (%)	Number 0.1 % Sample	Error (%)
	178,812	179,364	0.3	177,400	0.8	186,500	4.3	184,000	2.9
	167,053	166,736	0.2	170,820	2.3	159,100	4.8	138,000	17.4
	420,580	423,762	0.8	437,710	4.1	371,400	11.7	311,000	26.1
	1,354,914	1,353,372	0.1	1,348,450	0.5	1,321,900	2.4	1,534,000	13.2
	30,118	30,448	1.1	29,420	2.3	27,400	9.0	29,000	3.7
	13,783	13,870	0.6	14,280	3.6	13,300	3.5	7,000	49.2
	65,626	65,616	0.0	64,570	1.6	57,700	12.1	62,000	5.5
	676	698	3.4	670	0.7	500	25.9	0	100.0
	2,254	2222	1.2	2180	3.1	1700	24.4	0	100.0
	262,620	263,238	0.2	270,730	3.1	268,500	2.2	237,000	9.8
	29,963	29,972	0.0	29,130	2.8	28,000	6.5	42,000	40.2
	7,401	7,484	1.2	7550	2.1	5500	25.6	5000	32.4
	20,030	19,942	0.4	19,940	0.4	19,500	2.6	22,000	9.8
Total	2,553,830	2,556,724	0.1	2,572,850	0.7	2,461,000	3.6	2,571,000	0.7

uses the idea of probabilities per level, when comparing with it we used the same probabilities in the same depths. Note however, that we did not enforce $p_{K-1} = 1$ in `Rand-Esu`'s tests since its implementation places the probability before performing the enumeration, contrary to how `FaSE` does it (as explained above). Even though in Sect. 5.1 we compared with `Kavosh`, it does not own an approximate version, as far as we know, so we did not consider it in this section.

We first ran our algorithm against the `Rand-Esu` in the `Jazz` network and the `Neural` network for a 10 % sampling percentage and recorded the number of subgraphs sampled per second. We used this instead of the raw execution time since the actual number of sampled subgraphs oscillates and thus it does not represent the quality of the algorithm speed-wise. We chose these two networks to both vary the type of tested networks and the average degree. The results obtained are plotted in Fig. 8.

The principal aspect to take note is that `Rand-FaSE` always outperforms `Rand-Esu`, being roughly an order of magnitude faster. This result is consistent with the one obtained in the exact approach. However, the speedup is expected to be slightly less in the sampling version, since the speedup derives from the number of enumerated subgraphs that do not require an isomorphism test, thus by reducing the number of subgraphs that are actually enumerated, the speedup will tend to decrease. Although, the results on these networks show that it is not a noticeable decrease.

Other important observation is that our algorithm, as well as `Rand-Esu`, appears to scale well with the increased subgraph size. As it increases a small drop is detectable, however it is a very subtle one.

To evaluate how the approximation speed compares to the exact value speed, we ran `Rand-FaSE` on the networks of Sect. 5.2.1 and plotted the execution time for various sampling percentages in relation to the time the

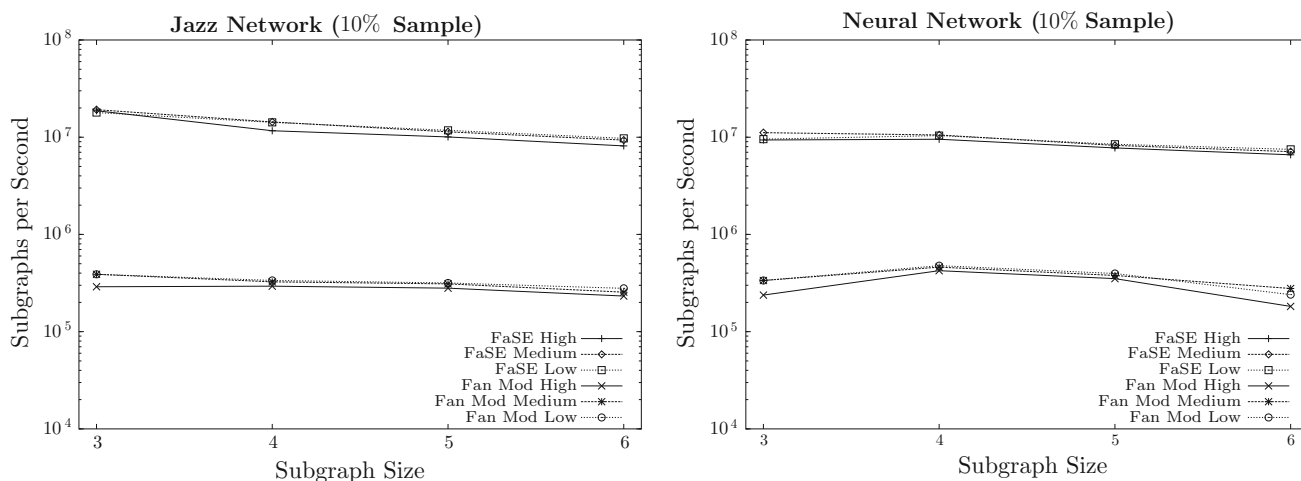


Fig. 8 Sampling speed comparison for a 10 % sample for the undirected Jazz Network and directed Neural Network

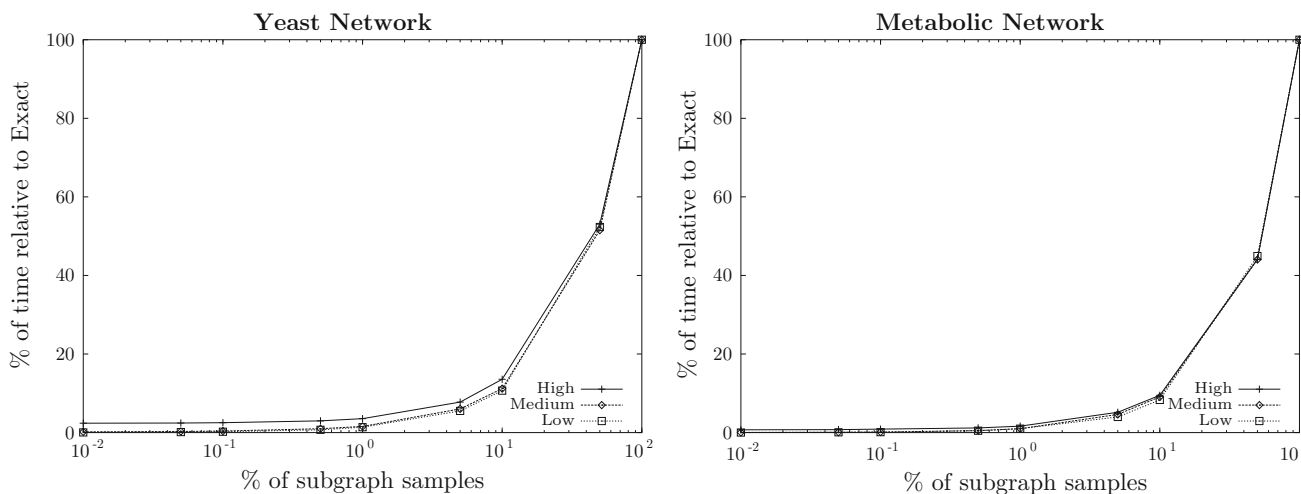


Fig. 9 Comparing the execution time of various sampling percentages with the Exact approach for the undirected Yeast Network and directed Metabolic Network for size 5 subgraphs

exact approach took, on a 5-subgraph census in Fig. 9. Using the same networks here as in the previous subsection, we get an idea of how speed performance compares with accuracy for the same setups. Note that the result displays a roughly linear growth behavior (since the graph is in a semi-log scale, the exponential represents a linear growth). However, even though for both networks a 50 % sample takes approximately 50 % of the time the exact approach takes, a 1 % sample takes 3 % of the time the exact approach, for the high setup. This effect worsens as the sampling percentage drops and ends up stabilizing at about 2 % of the time the exact approach takes, regardless of the sampling percentage. The reason for so is that thanks to the way the high setup is designed, it ends up enumerating all subgraphs up to size

$K - 1$. For the medium setup a similar effect is noticeable, but much subtler, since in this case we are enumerating all subgraphs to size $K - 2$. Obviously, the low setup does not display this behavior, but likewise, as the sampling percentage decreases the relation between time of the approximation and time of the exact deviates more from an exponential. For example, a 0.1 % sample takes approximately 0.2 % of the time the exact approach does.

Since the main goal of running an approximation algorithm is to apply it to a network where the exact approach is unfeasible, we tested our algorithm using the ideas discussed in Sect. 4.3 in an undirected network with 1,134,890 nodes and 2,987,624 edges that represent the network of a Youtube community taken from SNAP (Yang and Leskovec 2012). We ran a 0.1 % sample for 4-

subgraphs with setup *high*, which took about 20 min to complete. Based on tests on enumerations on 3-subgraphs of the same network and the results of this section, it should take about a full day to run the exact approach. The results allow us to have an idea of the total number of subgraphs as well as the distribution of the different subgraph types having run for only a very small fraction of what the exact approach is expected to.

5.2.3 Measuring convergence

To bring this section to an end, we performed some tests to assess the convergence of our algorithm. In Sect. 5.2.1, we could observe the percentage of correctly estimated values converging towards the optimal value so we consolidate this with a more detailed view over the percentage of error and the standard deviation. We ran Rand-FaSE with the *astroPh* network with setup *high* for various sampling percentages and measured both the relative error and the standard deviation normalized by the real value. We used this network since it is of a larger size and thus allows us to better observe the convergence and standard deviation evolution. The results are plotted in Fig. 10. Note that “Sub1” refers to the “L” shaped size 3 subgraph and “Sub2” refers to the triangle (size 3 complete graph). The number of subgraphs of type “Sub1” and “Sub2” in the *astroPh* network is of the same order of magnitude.

As expected, both the relative error and the standard deviation decrease towards 0. It is interesting to notice that above the 10 % sampling percentage the relative error and standard deviation stabilize and decrease very slowly, with little fluctuations.

6 Conclusion

In this paper, we presented both Rand-FaSE, an extension of FaSE that performs an approximate network-centric subgraph census. By making use of the common topology of the enumerated subgraphs and encapsulating this information in a tree structure called a g-trie, FaSE is able to discard most of the isomorphism tests required to correctly identify each subgraph type, which is the main bottleneck of this problem. Hence, it achieves much better results than any of the past approaches that tackle the same problem, which is shown by the results found by comparing all approaches. Furthermore, its sampling version, Rand-FaSE, acts as a logical extension and works by sampling only a percentage of the total number of subgraphs. By placing a probability in each depth and during the enumeration process only continuing the recursive step if a drawn random number is smaller than that depth’s probability, the algorithm gives an unbiased estimate of the real frequency of each subgraph type in the original network.

Thanks to FaSE’s use of LS-Labeling, the algorithm is very generic and allows for various different LS-Labeling functions. This means that the algorithm can be easily adapted for different scenarios such as colored graphs or multigraphs. In these more complex setups, network-centric approaches have a clear advantage over other approaches that require a pre-generated set of subgraphs as input since the addition of colors or multiedges vastly increments the number of possible subgraphs types. Network-centric algorithms naturally only enumerate the existing types, which are normally a very small fraction of the total number of possibilities. We note that there are some examples of applications on other graph setups, such

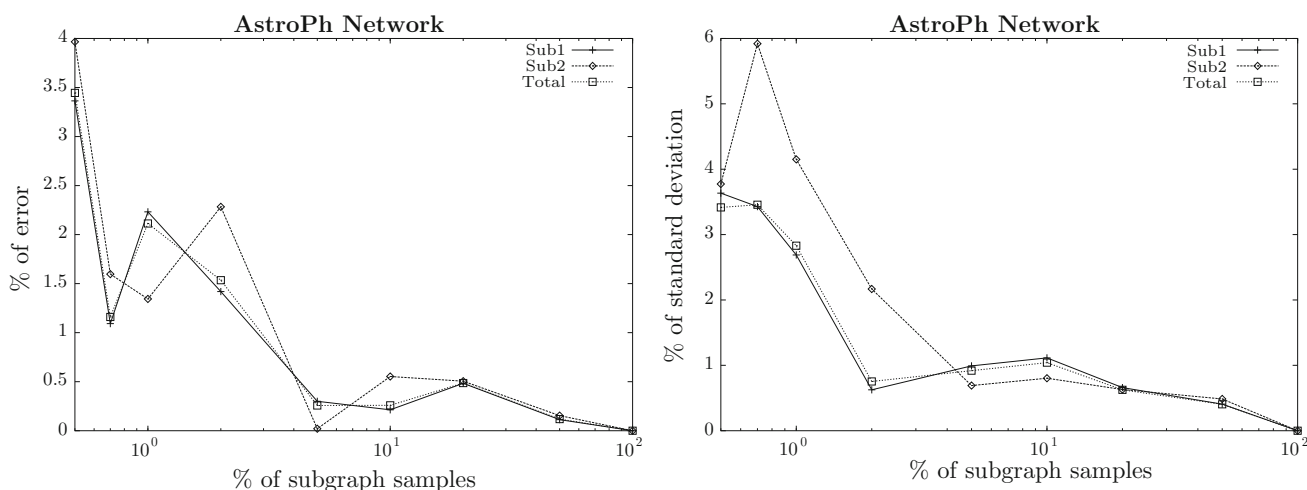


Fig. 10 Testing convergence through % of error to the real value and standard deviation (normalized by the real value)

as weighted graphs (Choobdar et al. 2012b) and colored graphs (Ribeiro and Silva 2014a).

The main drawback of the FaSE algorithm is the memory it spends to store the g-trie and all its leaves. We intend to tackle this issue by first noticing that since our g-trie only stores the intermediate classes generated by the LS-Labeling, there is still a lot of common substructures that we can take advantage of. The idea is to compress the g-trie so as to only store as many leaves as actual isomorphism classes during the actual enumeration process. Having this done, we intend to pre-calculate a whole g-trie and completely discard the isomorphism tests, as long as there is enough memory.

Regarding Rand-FaSE, we wish to experiment with the large-scale data structures to store huge networks. As stated in Sect. 4.3, we have tried different approaches and tested it real networks, as said in Sect. 5.2.2. However, there is still a lot of improvement that could be done by studying how the queries of connected vertices are performed to explore any patterns.

We are currently working on a parallel version of FaSE to take advantage of having multiple processors working. Our goal is to adapt it in a shared memory environment, by having multiple threads sharing the work.

Lastly, we wish to apply both algorithms in real complex networks in greater detail by both considering larger subgraph sizes and getting more accurate approximations (by having a larger sampling percentage).

Acknowledgments Pedro Ribeiro is funded by an FCT Research Grant (SFRH/BPD/81695/2011). This work is partly funded by NSRF and ERDF through the COMPETE and ON.2 Programmes and by National Funds through FCT, within projects FCOMP-01-0124-FEDER-037281 and NORTE-07-0124-FEDER-000059.

References

- Albert I, Albert R (2004) Conserved network motifs allow protein-protein interaction prediction. *Bioinformatics* 20(18):3346–3352. doi:10.1093/bioinformatics/bth402
- Batagelj V, Mrvar A (2006) Pajek datasets. <http://vlado.fmf.uni-lj.si/pub/networks/data/>
- Bhuiyan M, Rahman M, Rahman M, Hasan MA (2012) Guise: uniform sampling of graphlets for large graph analysis. In: IEEE international conference on data mining, ICDM, pp 91–100
- Choobdar S, Ribeiro P, Bugla S, Silva F (2012a) Co-authorship network comparison across research fields using motifs. In: IEEE/ACM international conference on advances in social networks analysis and mining, IEEE, pp 147–152. doi:10.1109/ASONAM.2012.34
- Choobdar S, Ribeiro P, Silva F (2012b) Motif mining in weighted networks. In: Data mining workshops (ICDMW), 2012 IEEE 12th international conference on, pp. 210–217. doi:10.1109/ICDMW.2012.111
- Cook SA (1971) The complexity of theorem-proving procedures. ACM Symposium on Theory of computing. ACM symposium on theory of computing (STOC). ACM, New York, NY, USA, pp 151–158
- Costa L, Oliveira O Jr, Travieso G, Rodrigues F, Boas P, Antequera L, Viana M, Da Rocha L (2011) Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Adv Phys* 60:329–412
- Fortunato S (2010) Community detection in graphs. *Phys Rep* 486(3–5):75–174
- Gleiser PM, Danon L (2003) Community structure in jazz. *Adv Complex Syst* 06(04), pp. 565–573. doi:10.1142/S0219525903001067
- Grochow J, Kellis M (2007) Network motif discovery using subgraph enumeration and symmetry-breaking. *Res Comput Mol Biol*, pp 92–106
- Iitzkovitz S, Levitt R, Kashtan N, Milo R, Iitzkovitz M, Alon U (2005) Coarse-graining and self-dissimilarity of complex networks. *Phys Rev E (Stat Nonlin Soft Matter Phys)* 71:016127
- Janssen E, Hurshman M, Kalyaniwalla N (2012) Model selection for social networks using graphlets. *Internet Math*
- Kashani Z, Ahrabian H, Elahi E, Nowzari-Dalini A, Ansari E, Asadi S, Mohammadi S, Schreiber F, Masoudi-Nejad A (2009) Kavosh: a new algorithm for finding network motifs. *BMC Bioinform* 10(1):318
- Kashtan N, Iitzkovitz S, Milo R, Alon U (2004) Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* 20(11):1746–1758
- Khakabimamaghani S, Sharafuddin I, Dichter N, Koch I, Masoudi-Nejad A (2013) Quatexelero: an accelerated exact network motif detection algorithm. *PLoS ONE* 8(7):e68073. doi:10.1371/journal.pone.0068073
- Kreher DL, Stinson DR (1999) Combinatorial algorithms: generation, enumeration, and search. *SIGACT News* 30(1):33–35
- Leskovec J, Kleinberg JM, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. *ACM Trans Knowl Discov From Data* 1(1). doi:10.1145/1217299.1217301
- Li X, Stones DS, Wang H, Deng H, Liu X, Wang G (2012) Netmode: network motif detection without nauty. *PLoS One* 7(12):e50093
- Marcus D, Shavitt Y (2010) Efficient counting of network motifs. In: ICDCS workshops, IEEE Computer Society, pp 92–98
- McKay B (2012) nauty. <http://cs.anu.edu.au/~bdm/nauty/>
- Milo R, Shen-Orr S, Iitzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. *Science* 298(5594):824–827
- Omidi S, Schreiber F, Masoudi-nejad A (2009) Moda: an efficient algorithm for network motif discovery in biological networks
- Paredes P, Ribeiro P (2013) Towards a faster network-centric subgraph census. In: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining, ACM, New York, NY, USA, ASONAM '13, pp 264–271. doi:10.1145/2492517.2492535
- Pržulj N (2010) Biological network comparison using graphlet degree distribution. *Bioinformatics* 26(6):853–854
- Ribeiro P, Silva F (2010) Efficient subgraph frequency estimation with g-tries. *International Workshop on algorithms in bioinformatics*, Springer, WABI, 6293:238–249
- Ribeiro P, Silva F (2014a) Discovering colored network motifs. In: Contucci P, Menezes R, Omicini A, Pongcel-Casasnovas J (eds) *Complex networks V*, Studies in computational intelligence, vol 549, Springer International Publishing, pp 107–118. doi:10.1007/978-3-319-05401-8_11
- Ribeiro P, Silva F (2014b) G-tries: a data structure for storing and finding subgraphs. *Data Min Knowl Discov* 28:337–377
- Ribeiro P, Silva F, Kaiser M (2009) Strategies for network motifs discovery. In: IEEE international conference on e-Science, e-Science, pp 80–87

- Schreiber F, Schwobbermeyer H (2004) Towards motif detection in networks: frequency concepts and flexible search. In: International workshop on network tools and applications in biology, NetTAB, pp 91–102
- Slota GM, Madduri K (2013) Fast approximate subgraph counting and enumeration. In: 42nd international conference on parallel processing (ICPP), pp 210–219
- Sporns O, Kötter R (2004) Motifs in brain networks. *PLoS Biol* 2:369
- Valverde S, Solé RV (2005) Network motifs in computational graphs: a case study in software architecture. *Phys Rev E* 72(026):107. doi:[10.1103/PhysRevE.72.026107](https://doi.org/10.1103/PhysRevE.72.026107)
- Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. *Nature* pp 440–442
- Wernicke S (2006) Efficient detection of network motifs. *IEEE/ACM Trans Comput Biol Bioinf*, pp 347–359
- Wu G, Harrigan M, Cunningham P (2011) Characterizing wikipedia pages using edit network motif profiles. In: 3rd International workshop on search and mining user-generated contents (SMUC), ACM, New York, NY, USA, pp 45–52
- Yang J, Leskovec J (2012) Defining and evaluating network communities based on ground-truth. In: Proceedings of the ACM SIGKDD workshop on mining data semantics, ACM, New York, NY, USA, MDS '12, pp 3:1–3:8. doi:[10.1145/2350190.2350193](https://doi.org/10.1145/2350190.2350193)