

A Survey on Subgraph Counting: Concepts, Algorithms and Applications to Network Motifs and Graphlets

PEDRO RIBEIRO¹, PEDRO PAREDES^{1,2}, MIGUEL E.P. SILVA^{1,3}, DAVID APARÍCIO^{1,4}, and FERNANDO SILVA¹

¹ DCC-FCUP & CRACS/INESC-TEC, University of Porto, Portugal

² Carnegie Mellon University, USA

³ University of Manchester, UK

⁴ Feedzai

Computing subgraph frequencies is a fundamental task that lies at the core of several network analysis methodologies, such as network motifs and graphlet-based metrics, which have been widely used to categorize and compare networks from multiple domains. Counting subgraphs is however computationally very expensive and there has been a large body of work on efficient algorithms and strategies to make subgraph counting feasible for larger subgraphs and networks.

This survey aims precisely to provide a comprehensive overview of the existing methods for subgraph counting. Our main contribution is a general and structured review of existing algorithms, classifying them on a set of key characteristics, highlighting their main similarities and differences. We identify and describe the main conceptual approaches, giving insight on their advantages and limitations, and provide pointers to existing implementations. We initially focus on exact sequential algorithms, but we also do a thorough survey on approximate methodologies (with a trade-off between accuracy and execution time) and parallel strategies (that need to deal with an unbalanced search space).

Keywords: Subgraphs, Subgraph Enumeration, Network Motifs, Graphlets, Analytical Algorithms, Approximate Counting, Sampling, Parallel Computation, Unbalanced Work Division

1 INTRODUCTION

Networks (or graphs) are a very flexible and powerful way of modeling many real-world systems. In its essence, they capture the interactions of a system, by representing entities as nodes and their relations as edges connecting them (e.g., people are nodes in social networks and edges connect those that have some relationship between them, such as friendships or citations). Networks have thus been used to analyze all kinds of social, biological and communication processes [35]. Extracting information from networks is therefore a vital interdisciplinary task that has been emerging as a research area by itself, commonly known as Network Science [15, 90].

One very common and important methodology is to look at the networks from a subgraph perspective, identifying the characteristic and recurrent connection patterns. For instance, network motif analysis [121] has identified the feed-forward loop as a recurring and crucial functional pattern in many real biological networks, such as gene regulation and metabolic networks [101, 208]. Another example is the usage of graphlet-degree distributions to show that protein-protein interaction networks are more akin to geometric graphs than with traditional scale-free models [138].

At the heart of these topologically rich approaches lies the subgraph counting problem, that is, the ability to compute subgraph frequencies. However, this is a very hard computational task. In fact, determining if one subgraph exists at all in another larger network (i.e., *subgraph isomorphism* [182]) is an NP-Complete problem [34]. Determining the exact frequency is even harder, and millions or even billions of subgraph occurrences are typically found even in relatively small networks.

Authors' contacts: Pedro Ribeiro (pribeiro@dcc.fc.up.pt), Pedro Paredes (preisben@cs.cmu.edu), Miguel E.P. Silva (miguel.silva@manchester.ac.uk), David Aparício (david.aparicio@feedzai.com), Fernando Silva (fds@dcc.fc.up.pt)
This work was made when all the authors were at DCC-FCUP.

Given both its usefulness and hard tractability, subgraph counting has been raising a considerable amount of interest from the research community, with a large body of published literature. This survey aims precisely to organize and summarize these research results, providing a comprehensive overview of the field. Our main contributions are the following:

- **A comprehensive review of algorithms for *exact* subgraph counting.** We give a structured historical perspective on algorithms for computing exact subgraph frequencies. We provide a complete overview table in which we employ a taxonomy that allows to classify all algorithms on a set of key characteristics, highlighting their main similarities and differences. We also identify and describe the main conceptual ideas, giving insight on their main advantages and possible limitations. We also provide links to existing implementations, exposing which approaches are readily available.
- **A comprehensive review of algorithms for *approximate* subgraph counting.** Given the hardness of the problem, many authors have resorted to approximation schemes, which allow trading some accuracy for faster execution times. As on the exact case, we provide historical context, links to implementations and we give a classification and description of key properties, explaining how the existing approaches deal with the balance between precision and running time.
- **A comprehensive review of *parallel* subgraph counting methodologies.** It is only natural that researchers have tried to harness the power of parallel architectures to provide scalable approaches that might decrease the needed computation time. As before, we provide an historical overview, coupled with classification on a set of important aspects, such as the type of parallel platform or availability of an implementation. We also give particular attention to how the methodologies tackle the unbalanced nature of the search space.

We complement this journey through the algorithmic strategies with a *clear formal definition of the subgraph counting problem* being discussed here, an *overview of its applications* and complete and a large number of *references to related work* that is not directly in the scope of this article. We believe that this survey provides the reader with an insightful and complete perspective on the field, both from a methodological and an application point of view.

The remainder of this paper is structured as follows. Section 2 presents necessary terminology, formally describes subgraph counting, and describes possible applications related subgraph counting. Section 3 reviews exact algorithms, divided between full enumeration and analytical methods. Approximate algorithms are described in Section 4 and parallel strategies are presented in Section 5. Finally, in Section 6 we give our concluding remarks.

2 PRELIMINARIES

2.1 Concepts and Common Terminology

This section introduces concepts and terminology related to subgraph counting that will be used throughout this paper. A *network* is modeled with the mathematical object *graph*, and the two terms are used interchangeably. Networks considered in this work are *simple labeled graphs*. Here we are interested in algorithms that count *small, connected, non-isomorphic subgraphs* on a single network.

Graph: A graph G is comprised of a set $V(G)$ of *vertices/nodes* and a set $E(G)$ of *edges/connections*. Nodes represent entities and edges correspond to relationships between them. Edges are represented as pairs of vertices of the form (u, v) , where $u, v \in V(G)$. In *directed* graphs, edges (u, v) are ordered pairs ($u \rightarrow v$) whereas in *undirected* graphs there is no order since nodes are always reciprocally connected ($u \rightleftharpoons v$). The *size* of a graph is the number of vertices in the graph and it is written as $|V(G)|$. A k -graph is a graph of size k . A graph is considered *simple* if it does not contain multiple edges (two or more edges connecting the same vertex pair) nor self-loops (an edge connecting a vertex to itself). Nodes are *labeled* from 0 to $|V(G)| - 1$, and $L(u) < L(v)$ means that u has a smaller label than v .

Neighborhood and Degree: The *neighborhood* of vertex $u \in V(G)$, denoted as $N(u)$, is composed by the set of vertices $v \in V(G)$ such that $(u, v) \in E(G)$. The *degree* of u , written as $deg(u)$, is the given by $|N(u)|$. The exclusive neighborhood $N_{exc}(u, S)$ are the neighbors of u that are not neighbors of any $v \in S$ with $u \neq v$.

Graph Isomorphism: A *mapping* of a graph is a bijection where each vertex is assigned a value. In the context of this work, since graphs are *labeled*, a mapping is a permutation of the node labels. Two graphs G and H are said to be isomorphic if there is a one-to-one mapping between the vertices of both graphs, such that there is an edge between two vertices of G if and only if their corresponding vertices in H also form an edge (preserving direction in the case of directed graphs). More informally, isomorphism captures the notion of two networks having the same edge structure – the same topology – if we ignore distinction between individual nodes. Figure 1 illustrates this concept. Despite looking different, the structure of the graphs is the same, and they are isomorphic. The labels in the nodes illustrate mappings that would satisfy the conditions given for isomorphism.

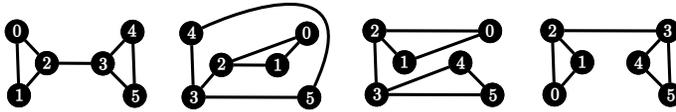


Fig. 1. Four isomorphic undirected graphs of size 6.

Subgraphs: A *subgraph* G_k of a graph G is a k -graph such that $V(G_k) \subseteq V(G)$ and $E(G_k) \subseteq E(G)$. A subgraph is *induced* if $\forall(u, v) \in E(G_k) \leftrightarrow (u, v) \in E(G)$ and is said to be *connected* when all pairs of vertices have a sequence of edges connecting them. *Graphlets* [138] are small, connected, non-isomorphic, induced subgraphs. Figure 2 presents all 4-node undirected graphlets.

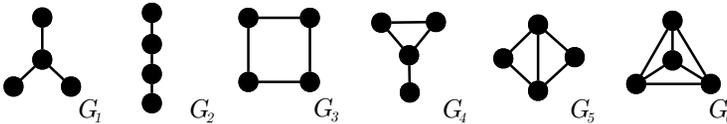


Fig. 2. All non-isomorphic undirected subgraphs (or graphlets) of size 4.

Orbit: The set of isomorphisms of a graph into itself is called the group of *automorphisms*: two vertices are said to be equivalent when there exists some automorphism that maps one vertex into the other. This equivalence relation partitions $V(G)$ into equivalence classes, which we refer to as *orbits*. Therefore, *orbits* are all the unique positions of a subgraph. For instance, a k -hub has k nodes but only 2 orbits: one *center-orbit* inhabited by a single node and a *leaf-orbit* where the remaining $k-1$ nodes are. Nodes at the same orbit are topologically equivalent. Figure 3 shows all different orbits of the graphlets from Figure 2.

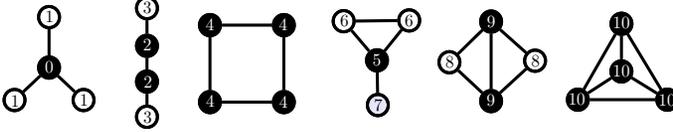


Fig. 3. The 10 orbits of all 4-node undirected graphlets.

Match and Frequency: A *match* of graph H in graph G occurs when there is a set of nodes from $V(G)$ that induce H . In other words, G_k is a subgraph of G that is isomorphic to H . Figure 4 shows the matches of three different subgraphs (A , B and C) on graph G . The *frequency* of H in G is the number of *different* $G_k \subseteq G$ that induce H . Two matches are considered different if they do not share all nodes and edges.

Graphlet-Degree Distribution : It is an extension of the node-degree distribution and both can be used for graph characterization and comparison. Notice that the node-degree can be seen as simply the orbit a in Figure 4. The graphlet-degree vector $G DV(v)$ is a feature vector of v specifying how many times it occurs in each orbit. The graphlet-degree distribution $G DD_G$ is a feature matrix of graph G where cell (i, j) indicates the number of nodes that appear i times in orbit j , and can be constructed from Fr_G , the frequency matrix where each line is the $G DV$ of a single node.

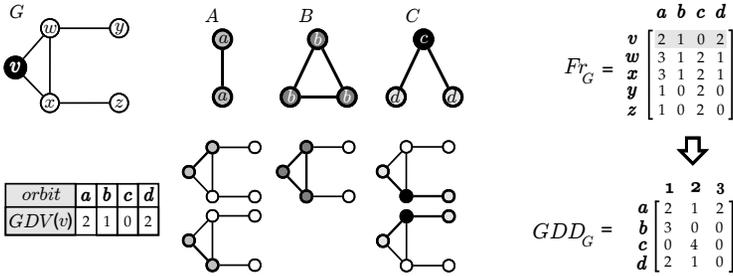


Fig. 4. $G DV(v)$ obtained by enumerating all undirected graphlet-orbits of sizes 2 and 3 (A , B and C) touching v , and resulting Fr_G and $G DD_G$ matrices for the complete 3-subgraph census

2.2 Problem statement

Making use of previous concepts and terminology, we now give a more formal definition of the problem tackled by this survey:

DEFINITION 2.1 (SUBGRAPH COUNTING). Given a set \mathcal{G} of non-isomorphic subgraphs and a graph G , determine the frequency of all induced matches of the subgraphs $G_s \in \mathcal{G}$ in G . Two occurrences are considered different if they have at least one node or edge that they do not share.

This problem is also known as *subgraph census*. In short, one wants to extract the occurrences of all subgraphs of a given size, or just a smaller set of "interesting" subgraphs, contained in a large graph G . Note how here the input is a single graph, in contrast with Frequent Subgraph Mining (FSM) where collections of graphs are more commonly used (differences between Subgraph Counting and FSM are discussed in Section 2.4.5).

Approaches diverge on which subgraphs are counted in G . *Network-centric* methods extract all k -node occurrences in G and then assess each occurrence's isomorphic type. On the other end of the spectrum, *subgraph-centric* methods first pick a isomorphic class and then only count occurrences matching that class in G . Therefore, subgraph-centric methods are preferable to network-centric algorithms when only one or a few different subgraphs are to be counted. *Set-centric* approaches are middle-ground algorithms that take as input a set of interesting subgraphs and only count those on G . This work is mainly focused on network-centric algorithms, while not limited to them, since: (a) exploring all subgraphs offers the most information possible when applying subgraph counting to a real dataset, (b) hand-picking a set of interesting subgraphs might be hard or impossible and could be heavily dependent on our knowledge of the dataset, (c) it is intrinsically the most general approach. It is obviously possible to use subgraph-centric methods to count all isomorphic classes, simply by executing the method once per isomorphic type. However, that option is only feasible for small subgraph sizes because larger k values produce too many subgraphs (see Table 1) and it is likely that a network only has a small subset of them, meaning that the method would spend a considerable amount of time looking for features that do not exist, while network-centric methods always do useful work since they count occurrences in the network.

Table 1. Number of different undirected and directed subgraphs (i.e., isomorphic classes), as well as their respective orbits, depending on the size of the graphlets.

k	Undirected		Directed	
	#Subgraphs	#Orbits	#Subgraphs	#Orbits
2	1	1	2	3 ($1.5 \times \text{\#Subgraphs}$)
3	2	3	15	30 ($2.0 \times \text{\#Subgraphs}$)
4	6	11	214	697 ($3.3 \times \text{\#Subgraphs}$)
5	21	58	9,578	44,907 ($4.7 \times \text{\#Subgraphs}$)
6	112	407	1,540,421	9,076,020 ($5.9 \times \text{\#Subgraphs}$)
7	823	4,306	872,889,906	$\approx 7 \times \text{\#Subgraphs}$
8	11,117	72,489	1,792,473,955,306	$\approx 8 \times \text{\#Subgraphs}$
9	261,080	2,111,013	13,026,161,682,466,252	$\approx 9 \times \text{\#Subgraphs}$

Here we are mainly interested in algorithms that count induced subgraphs, but non-induced subgraphs counting algorithms are also considered. Counting one or the other is equivalent since it is possible to obtain induced occurrences from non-induced occurrences, and vice-versa. However, we should note that, at the end of the counting process, induced occurrences need to be obtained by the algorithm. This choice penalizes non-induced subgraph counting algorithms since the transformation is quadratic on the number of subgraphs [52]. Some algorithms count orbits instead of subgraphs [62]. However, counting orbits can be reduced to counting subgraphs and, therefore, these algorithms are also considered.

We should note that we only consider the most common and well studied subgraph frequency definition, in which different occurrences might share a partial subset of nodes and edges, but there are other possible frequency concepts, in which this overlap is explicitly disallowed [47, 167].

2.3 Algorithms Not Considered

In this work we focus on practical algorithms that are capable of counting all subgraphs of a given size. Therefore, algorithms that only target specific subgraphs are not considered (e.g., triads [163], cliques [7, 51], stars [37, 54] or subtrees [91]). Furthermore, given our focus on generalizability, we do not consider algorithms that are only capable of counting subgraphs in specific graphs (e.g., bipartite networks [161], trees [36]), or that only count local subgraphs [38].

Graphs used throughout this work are simple, have a single layer of connectivity and do not distinguish the node or edge types with qualitative or quantitative features. Therefore we do not discuss here algorithms that use colored nodes or edges [53, 58, 148], and neither those that consider networks that are heterogeneous [57, 156], multilayer [23, 143], labelled/attributed [123], probabilistic [162] or any kind of weighted graphs [197].

Finally, the networks we consider are static and do not change their topology. We should however note that there has been an increasing interest in temporal networks, that evolve over time [66]. Some algorithms beyond the scope of this survey try to tackle temporal subgraph counting, either by considering temporal networks as a series of static snapshots [14, 70], by timestamping edges [129], or by considering a stream of small updates to the graph topology [27, 77, 166, 173].

2.4 Applications and Related Problems

2.4.1 Subgraph Isomorphism. Given two graphs G and H , the *subgraph isomorphism* problem is the computational task of determining if G contains a subgraph isomorphic to H . Although efficient solutions might be found for specific graph types (e.g., linear solutions exist for planar graphs [48]), this is a known NP-Complete problem for general graphs [34], and can be seen as much simpler version of counting, that is, determining if the number of occurrences is bigger than zero. This task is closely related to the *graph isomorphism* problem [107, 108], that is, the task of determining if two given graphs are isomorphic. Since many subgraph counting approaches rely on finding the subgraphs contained in a large graph and then checking to what isomorphic class the subgraphs found belong to, subgraph isomorphism can be seen as an integral part of them. The well known and very fast NAUTY tool [106] is used by several subgraph counting algorithms to assess the type of the subgraph found [130, 149, 196].

2.4.2 Subgraph Frequencies. The small patterns found in large graphs can offer insights about the networks. By considering the frequency of all k -subgraphs, we have a very powerful and rich feature vector that characterizes the network. There has been a long tradition on using the triad census on the analysis of social networks [191], and they have been used as early as in the 70s to describe local structure [65]. Examples of applications in this field include studying social capital features such as brokerage and closure [137], discovering social roles [42], seeing the effect of individual psychological differences on network structure [76] or characterizing communication [181] and social networks [28]. Given the ubiquity of graphs, these frequencies have also been used on many other domains, such as in biological [175], transportation [184] or interfirm networks [99].

2.4.3 Network Motifs. A subgraph is considered a *network motif* if it is somehow exceptional. Instead of simply using a frequency vector, motif based approaches construct a *significance profile* that associates an importance to each subgraph, typically related to how overrepresented it is. This concept first appeared in 2002 and it was first defined as subgraphs that occurred more often than expected when compared against a null model [121]. The most common null model is to keep the degree sequence and with this we can obtain characteristic network fingerprints that have been shown to be very rich and capable of classifying networks into distinct superfamilies [120]. Network motif analysis has since been in a vast range of applications, such as in the analysis of biological

networks (e.g., brain [177], regulation and protein interaction [204] or food webs [16]), social networks (e.g., co-authorship [32] or online social networks [43]), sports analytics (e.g., football passing [17]) or software networks (e.g., software architecture [183] or function-call graphs [199]).

In order to compute the significance profile of motifs in a graph G , most conceptual approaches rely on generating a large set of $R(G)$ of similar randomized networks that serve as the desired null model. Thus, subgraph counting needs to be performed both on the original network and on the set of randomized networks. If the frequency of a subgraph S is *significantly bigger* in G than its average frequency in $R(G)$, we can consider S to be a network motif of G [79]. Other approaches try to avoid exhaustive generation of random networks and, thus, avoid also counting subgraphs on them, by following a more analytical approach capable of providing estimations of the expected frequencies (e.g., using an expected degree model [117, 135, 165] or a scale-free model [178]). Nevertheless, there is always the need of counting subgraphs in the original network.

While network motifs are usually about induced subgraph occurrences [121, 198], there are some motif algorithms that count non-induced occurrences instead [92, 126]. Moreover, although most of the network motifs usages assume the previously mentioned statistical view on significance as overrepresentation, there are other possible approaches [200] such as using information theory concepts (e.g., motifs based on entropy [1, 33], subgraph covers [192], or minimum description length [22]). We should also note that some approaches try to better navigate the space of "interesting" subgraphs, so that reaching larger motif sizes can be reached not by searching all possible larger k -subgraphs, but instead by leveraging computations of smaller motifs [97, 134].

Finally, we should note that several authors use the term motif to refer to small subgraphs, even when it does not imply any significance value beyond simple frequency on the original network.

2.4.4 Orbit-Aware Approaches and Network Alignment. When authors use the term *graphlet*, they commonly take orbits into consideration, and use metrics such as the graphlet-degree distribution (GDD, see details in section 2.1), a concept that appeared in 2007 [138]. In this way, graphlet algorithms count how many times each node appears in each orbit. Unlike motifs, graphlets do not usually need a null model (i.e., networks are directly compared by comparing their respective GDDs). These orbit-aware distributions can be used for comparing networks. For instance, they have shown that protein interaction networks are more akin to random geometric graphs than to traditional scale-free networks [138]. Moreover, they are also used to compare nodes (using graphlet-degree vectors). This makes them useful for *network alignment* tasks, where one needs to establish topological similarity between nodes from different networks [118]. Several graphlet-based network alignment algorithms have been proposed and shown to work very well for aligning biological networks [11, 87, 88, 100, 179].

2.4.5 Frequent Subgraph Mining (FSM). FSM algorithms find subgraphs that have a *support* higher than a given threshold. The most prevalent branch of FSM takes as input a bundle of networks and finds which subgraphs appear in a vast number of them - referred to as *graph transaction based FSM* [74]. These algorithms [69, 125, 202] heavily rely on the Downward Closure Property (DCP) to efficiently prune the search space. Algorithms for subgraph counting, which is our focus, can not, in general, rely on the DCP since it is not possible to know if growing an infrequent k -node subgraph will result, or not, in a frequent $k + 1$ subgraph. Furthermore, we are not only interested in frequent subgraphs but in all of them, since rare subgraphs can also give information about the network's topology. A less prominent branch of FSM, *single graph based FSM*, targets frequent subgraphs in a single large network, much like our subgraph counting problem. However, they adopt various support metrics that allow for the DCP to be verified, which, as stated previously, is not the case in the general subgraph counting problem [74].

2.5 Other Surveys and Related Work

To the best of our knowledge there is no other comparable work to this survey in terms of scope, thoroughness and recency. Most of the already existing surveys that deal with subgraph counting are directly related to network motif discovery. Some of them are from before 2015 and therefore predate many of the most recent algorithmic advances [82, 105, 150, 180, 198], and all of them only present a small subset of the strategies discussed here. There are more recent review papers, but they all differ from our work and have a much smaller scope. **Al Hasan and Dave** [6] only consider triangle counting, **Xia et al.** [200] focus mainly on significance metrics, and finally, while we here present a structured overview of more than 50 exact, approximate and parallel algorithmic approaches, **Jain and Patgiri** [72] presents a much simpler description of 5 different algorithms.

3 EXACT COUNTING

As subgraph counting evolved over the years, a multitude of algorithms and methods were developed that address the problem in different ways and for distinct purposes. As such, it is useful, although not easy, to group strategies together in order to facilitate their understanding as well as learn why and how they came about. With this in mind we divided this section into two major groups of algorithms, namely enumeration and analytic approaches, which are further subdivided in their respective section. Table 2 summarizes our proposed taxonomy composed of six aspects, ordered by their publication year: (i) **approach** (enumeration or analytic), (ii) **type** (a subgroup of the underlying approach), (iii) **k -restriction** (does the method only work for certain subgraph sizes?), (iv) **orbit awareness** (does the method also count orbits?), (v) **directed** (is the method applicable to directed graphs?) and (vi) if code is **publicly available**. At the end of this section, we also present some related theoretical results that influenced some of the algorithms we discuss.

Table 2. Overview of all major exact algorithms.

	Year	Approach	Type	k -restriction	Orbit	Directed	Code
MFINDER [121]	2002	Enum.	Classical	None	✗	✓	[9]
ESU [193, 196]	2005	Enum.	Classical	None	✗	✓	[194]
ITZHACK [71]	2007	Enum.	Classical	≤ 5	✗	✓	✗
GROCHOW [56]	2007	Enum.	Single-subgraph	None	✗	✓	✗
KAVOSH [78]	2009	Enum.	Classical	None	✗	✓	[122]
GTRIES [147, 149]	2010	Enum.	Encapsulation	None	✓	✓	[144]
RAGE [102, 103]	2010	Analytic	Decomposition	≤ 5	✗	✓	[104]
NEMO [85]	2011	Enum.	Single-subgraph	None	✗	✓	[155]
NETMODE [92]	2012	Enum.	Encapsulation	≤ 6	✗	✓	[93]
SCMD [185]	2012	Enum.	Encapsulation	None	✗	✗	✗
ACC-MOTIF [110, 111]	2012	Analytic	Decomposition	≤ 6	✗	✓	[109]
ISMAGS [40, 68]	2013	Enum.	Single-subgraph	None	✗	✓	[133]
QUATEXELERO [80]	2013	Enum.	Encapsulation	None	✗	✓	[81]
FASE [130]	2013	Enum.	Encapsulation	None	✗	✓	[145]
ENSA [205]	2014	Enum.	Encapsulation	None	✗	✓	✗
ORCA [62, 63]	2014	Analytic	Matrix-based	≤ 5	✓	✗	[64]
HASH-ESU [75]	2015	Enum.	Encapsulation	None	✗	✓	✗
SONG [176]	2015	Enum.	Encapsulation	None	✗	✓	✗
ORTMANN [127, 128]	2016	Analytic	Matrix-based	≤ 4	✓	✓	✗
PGD [3, 5]	2016	Analytic	Decomposition	≤ 4	✓	✗	[2]
PATCOMP [61]	2017	Enum.	Encapsulation	None	✗	✓	✗
ESCAPE [136]	2017	Analytic	Decomposition	≤ 5	✓	✗	[168]
JESSE [112, 114]	2017	Analytic	Matrix-based	None	✓	✗	[113]

3.1 Enumeration approaches

A significant part of the history of practical subgraph counting algorithms is intertwined with network motif analysis. This is because when motifs were first proposed [121], they raised the interest and necessity for efficient subgraph counting, which has since been growing and establishing itself as a very important graph analysis primitive with multidisciplinary applicability.

Exact subgraph counting consists of *counting* and *categorizing* (i.e., determining the isomorphic class of) all subgraph occurrences. Early methods *first enumerate* all connected subgraphs with k -vertices and *only afterwards categorize* each subgraph found using a graph isomorphism tool like NAUTY [106]. We refer to these as **classical methods**.

Many methods followed this strategy, until new methods appeared that counted the frequency of a single-subgraph category instead, thus avoiding the categorization step necessary by the classical methods. This was done by only enumerating one particular subgraph of interest. Even though they were not the fastest methods for a network-centric application, they were an important milestone towards the methods that followed. We refer to these as **single-subgraph-search methods**.

The next step was to combine the two previous ideas into a more efficient approach: merge the enumeration and categorization steps together. This was achieved in different ways, such as using common topological features of subgraphs or pre-computing some information about subgraphs to avoid repeated computations of isomorphism. We refer to these as **encapsulation methods**.

The next sections thoroughly delve into the most well-known methods of each category, giving an historical perspective on each, in an effort to understand each method's breakthroughs and drawbacks, and how subsequent algorithms built upon them to reduce (or mitigate) their limitations.

3.1.1 Classical methods. In the seminal work, **Milo et al.** [121] first defined the concept of network motif and also proposed MFINDER, an algorithm to count subgraphs. MFINDER is a recursive backtracking algorithm, that is applied to each edge of the network. A given edge is initially stored on a set S , which is recursively grown using edges that are not in S but share one endpoint with at least one edge in S . When $|S| = k$, the algorithm checks if the subgraph induced by S has been found for the first time by keeping a hash table of subgraphs already found. If the subgraph was reached for the first time, the algorithm categorizes it and updates the hash table (otherwise, the subgraph is ignored).

Another very important work, by **Wernicke** [193], proposed a new algorithm called ESU, also known as FANMOD due to the graphical tool that uses ESU as its core algorithm [196]. This algorithm greatly improved on MFINDER by never counting the same subgraph twice, thus avoiding the need to store all subgraphs in a hash table. ESU applies the same recursive method to each vertex v of the input graph G : it uses two sets V_S and V_E , which initially are set as $V_S = \{v\}$ 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 $V_E = V_E \cup \{u \in N_{exc}(u, V_S) : L(u) > L(v)\}$ (where v is the original vertex to be added to V_S). The N_{exc} here makes sure we only grow the list of possibilities with vertices not already in V_S and the condition $L(u) > L(v)$ 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, which means V_S contains a single occurrence of a k -subgraph. At the end of the process, ESU performs isomorphism tests to assess the category of each subgraph occurrence, which is a considerable bottleneck.

Itzhack et al. [71] proposed a new algorithm that was able to count subgraphs using constant memory (in relation to the size of the input graph). Itzhack et al. did not name their algorithm, so we will refer to it as ITZHACK from here on. ITZHACK avoids explicitly computing the isomorphism class of each counted subgraph by caching it for each different adjacency matrix, seen as a bitstring. This strategy only works for subgraphs of k up to 5, since it would use too much memory for higher

values. Additionally, the enumeration algorithm is also different from ESU. This method is based on counting all subgraphs that include a certain vertex, then removing that node from the network and repeating the same procedure for the remaining nodes. For each vertex v , first the algorithm considers the tree composed of the k neighborhood of v , that is, a tree of all vertices at a distance of $k - 1$ or less from v . This is very similar to the tree obtained from performing a breadth-first search starting on v , with the difference that vertices that appear on previous levels of the tree are excluded if visited again. This tree can be traversed in a way that avoids actually creating it by following neighbors, and thus only using constant memory. To perform the actual search, the method uses the concept of *counting patterns*, which are different combinatorial ways of choosing vertices from different levels of the tree. 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. Itzhack et al. claimed that ITZHACK is over 1,000 times faster than ESU, however the author of ESU disputed this claim in [195], stating that the experimental setup was faulty and claimed that ITZHACK is only slightly faster than ESU (its speedup could be attributed mainly to the caching procedure).

Kashani et al. [78] proposed a new algorithm called KAVOSH. Like ESU and ITZHACK, 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. Its functioning is very similar to that of ITZHACK: it builds an implicit breadth-first search tree and then uses a similar concept to the counting patterns used by ITZHACK. However, it is a more general method since it does not perform any caching of isomorphism information, allowing the enumeration of larger subgraphs.

3.1.2 Single-subgraph-search methods. The idea that it is possible to obtain a very efficient method of counting a single subgraph category was first noticed by **Grochow and Kellis** [56]. Their base method consists on a backtracking algorithm that is applied to each vertex. It tries to build a partial mapping from the input graph to the target subgraph (the subgraph it is trying to count) by building all possible assignments based on the number of neighbours. Grochow and Kellis also suggested an improvement based on symmetry breaking, using the automorphisms of the target subgraph to build set of conditions, of the form $L(a) < L(b)$, to prevent the same subgraph from being counted multiple times. This symmetry breaking idea allowed for considerable improvements in runtime, specially for higher values of k . Grochow and Kellis did not name their algorithm, so we will refer to it as the GROCHOW algorithm from here on.

Koskas et al. [85] presented a new algorithm which they called NEMO. This method draws some ideas from GROCHOW, since it performs a backtrack based search with symmetry breaking in a similar fashion. Although, instead of using conditions on vertex labels, it finds the orbits of the target subgraph and forces an ordering between the labels of the vertices from the input graph that match vertices in the target subgraph with the same orbit. Additionally, it uses a few heuristics to prune the search early, such as ordering the vertices from the target graph such that for all $1 \leq i \leq k$, its first i vertices are connected.

ISMAGS, which is based on its predecessor ISMA [40], was proposed by **Houbraken et al.** [68]. The base idea of this method is similar to the one in GROCHOW, however, the authors use a clever node ordering and other heuristics to speedup the partial mapping procedure. Additionally, their symmetry breaking conditions are significantly improved by applying several heuristic techniques based on group theory.

3.1.3 Encapsulation methods. The ideas applied to GROCHOW introduced a way of escaping the classic setup of enumerating and then categorizing subgraphs, albeit focusing on a single subgraph.

The next step would be to extend this idea to a more general algorithm, which is appropriate to a full subgraph counting. This was first done by **Ribeiro and Silva** [147] using a new data-structure they called the *g-trie*, for *graph trie*. The *g-trie* is a prefix tree for graphs, each node represents a different graph, where the graph of a parent node has shared common substructures with the graph of its child node, which are characterized precisely by the vertices of the graph of the child node. The root represents the one vertex graph with one child, a node representing the edge graph, which in turn has two children representing the triangle graph and the 3-path, and so on. This tree can be augmented by giving each node symmetry breaking conditions similar to those from GROCHOW. The authors show how to efficiently build this data-structure and augment with the symmetry breaking conditions for any set of graphs. Also, they describe a subgraph counting algorithm based on using this data-structure along with an enumeration technique similar to that of GROCHOW. However, since this data-structure encapsulates the information of multiple graphs in an hierarchical order, it achieves a much faster full subgraph counting algorithm. The usage of this data-structure has been significantly extended since its original publication, such as a version for colored networks [148] or an orbit aware version [13]. A more detailed discussion of the data-structure and the subgraph counting algorithm is presented in [149]. Also, even though the subgraph counting algorithm was not named, we will refer to it as the GTRIE algorithm from here on.

GTRIE encapsulates common topological information of the subgraphs being counted, but there are other approaches, such as **Li et al.** [92], who developed NETMODE. It builds on KAVOSH, by using its enumeration algorithm, but instead of using NAUTY to perform the categorization step, it makes use of a cache to store isomorphism information and thus is able to perform it in constant time. This is very similar to what ITZHACK does, however, Li et al. suggested an improvement that allows NETMODE to scale to $k = 6$ without using too much memory. This improvement is based on the *reconstruction conjecture* [60], that states that two graphs with 3 or more vertices are isomorphic if their deck (the set of isomorphism classes of all vertex-deleted subgraphs of a graph) is the same. This is known to be false for directed graphs with $k = 6$, but there are very few counter-examples that can be directly stored such as in the $k \leq 5$ case, thus NETMODE applies the conjecture for all the remaining cases by building their deck, hashing its value and storing its count in a table.

Wang et al. [185] proposed a new method called SCMD that counts subgraphs in compressed networks. SCMD applies a symmetry compression method that finds sets of vertices that are in an homeomorphism to cliques or empty subgraphs, which have the additional property that any other vertex that connects to a vertex in the set is connected to all other vertices in the set. These sets of vertices form a partition of the graph that is obtained using a method published in [98], which is based on looking at vertices in the same orbit. This is a versatile method that can use algorithms like ESU or KAVOSH to enumerate all subgraphs of sizes from 1 to k in the compressed network. Finally, SCMD “decompresses” the results by looking at all the different enumerated subgraphs and calculating all the combinations that can form a decompressed subgraph. For example, for $k = 3$, if a compressed 2-subgraph is found containing two vertices: one compressed vertex representing a clique of 5 uncompressed vertices and a compressed vertex representing a single vertex from the uncompressed graph, it results in $\binom{5}{2} + \binom{5}{3}$ triangles from the uncompressed graph, obtained by taking two vertices from the clique vertex and one from the other vertex, which are all connected and thus form a triangle, $\binom{5}{2}$, plus taking three vertices from the clique vertex $\binom{5}{3}$. The authors argue that most complex networks exhibit high symmetries and thus are improved by the application of this technique. Even though their work only includes undirected graphs, the authors affirm it is easy to extend the same concepts to directed networks. **Xu et al.** described another algorithm that enumerates subgraphs on compressed networks, called ENSA [201, 205]. Their method is based

on an heuristic graph isomorphism algorithm, and they also discuss an optimization based on identifying vertices with unique degrees.

Following the ideas first applied in GTRIE, **Khakabimamaghani et al.** [80] proposed a new algorithm they called QUATEXELERO. QUATEXELERO is built upon any incremental enumeration algorithm, like ESU, and it implements a data structure similar to a quaternary tree. Each node in the tree represents a graph, that can be built by looking into the nodes from the path from it to the root of the tree. Additionally, all graphs represented by a single node belong to the same isomorphism class. To fill the tree, initially a pointer to the root of the tree is set. Whenever a new vertex is added to the partial enumeration map, QUATEXELERO looks into the existing edges between the newly added node and the previously existing nodes in the mapping and stores its information in the quaternary tree. For each vertex in the mapping, depending on whether there is no edge, an inedge, an outedge or a bidge between it and the newly added vertex, the pointer is assigned to one of its four children, creating it if it was nonexistent.

Parallel to the publishing of the work of QUATEXELERO, **Paredes and Ribeiro** [130] proposed FASE. The idea of FASE is similar to the one from QUATEXELERO, however, instead of using a quaternary tree, it uses a data-structure similar to the g-trie, albeit without the symmetry breaking condition augmentation. This data-structure has the same property as the quaternary tree that every node represents a graph and each node is built using the adjacency information of a newly added vertex in relation to the vertices present in its parent.

Other works that extend these ideas have been proposed subsequently. For example, **Jing and Cheng** [75] propose HASH-ESU, an algorithm based on the same idea from QUATEXELERO and FASE, but which hashes the adjacency information instead of storing it in a tree. Another example is the work by **Song et al.** [176]. They describe a method that starts by enumerating all $k = 3$ subgraphs using ESU and then use dynamic programming to grow connected sets and perform the counting. Their algorithm was not named, so we will refer to it as the SONG algorithm from here on.

Both QUATEXELERO and FASE have potential memory issues, since there may be several nodes representing the same graph, which is not a problem for GTRIE since it only stores one copy of each possible graph. To address this, **Himamshu and Jain** [61] proposed PATCOMP. Their method compresses the quaternary tree using a technique similar to a radix tree, however, their method is 2 to 3 times slower and only saves around 10% of the memory usage.

3.2 Analytic approaches

Since the overall goal of the problem we are aiming to solve is to count subgraphs, it is not necessary to explicitly enumerate each connected set of size k . Here lies the difference between *counting* and *enumerating* or *listing*. It was with this in mind that a new class of methods emerged striving to avoid enumerating all subgraphs in a graph. We can point two main approaches to this type of counting.

The first one tries to relate the frequency of each subgraph with the frequencies of other subgraphs of the same or smaller size. This permits constructing a matrix of linear equations between subgraphs frequencies that can be solved using traditional linear algebra methods. We refer to these as **matrix based methods**.

The second approach targets each subgraph individually by decomposing it in several smaller patterns of graph properties, like common neighbors, or triangles that touch two vertices. We refer to these as **decomposition methods**.

3.2.1 Matrix based methods. The first known method to apply a practical analytic approach based on matrix multiplication to subgraph counting was ORCA, a work by **Hočevár and Demšar** [62], which is based on counting orbits and not directly subgraphs. Their original work was targeted at

orbits in subgraphs up to 5 vertices and, because of that, they count induced subgraphs specifically, while most analytic approaches count non-induced occurrences. ORCA works by setting up a system of linear equations per vertex of the input graph that relate different orbit frequencies, which are the system's variables. This system of linear equations contains information about the input graph. By construction, the matrix has a rank equal to the number of orbits minus 1, thus to solve it one only need to find the value of one the orbit frequencies and use any standard linear algebra method to solve it. Usually, the orbit pertaining to the clique is chosen, since there are efficient algorithms to count this orbit and, for sparse enough networks, it is usually the one with the least occurrences, making it less expensive to count.

Later, the authors of ORCA extended their work by suggesting a way of producing equations for arbitrary sized subgraphs [63], although their available practical implementation is still limited to size 5 [64]. Another possible extension for ORCA was proposed by [112] with the JESSE algorithm, which was further complemented with a strategy for optimizing the computation by carefully selecting less expensive equations [114].

Similar to ORCA, but using a different strategy, **Ortmann and Brandes** [127] proposed a new method, which they further improved and better described in [128]. They also target orbits, but for subgraphs of size up to 4. Their approach is based on looking into non-induced subgraphs using them to build linear equations that are less expensive to compute. Additionally, they also apply an improved clique counting algorithm. **Ortmann and Brandes** [127] did not name their algorithm, so we will refer to it as the ORTMANN algorithm from here on.

3.2.2 Decomposition methods. Before ORCA was proposed, the first ever practical method that used an analytic approach to subgraph counting was RAGE, by **Marcus and Shavitt** [102, 103]. Their method is based on [55] which employs similar techniques but with a more theoretical focus. RAGE targets non-induced subgraphs and orbits of size 3 and 4. It does so by running a different algorithm for each of the 8 existing subgraphs. Each algorithm is based on merging the neighborhoods of pairs of vertices to ensure that a given quartet of vertices have the desired edges to form a certain subgraph.

ACC-MOTIF, which was proposed by **Meira et al.** [110] and then further improved in [111], was also one of the first methods to employ an analytic strategy, but stands out as the only known analytic method that also works for directed subgraphs. ACC-MOTIF also targets non-induced subgraphs and their latest version supports up to size 6 subgraphs.

Another method that followed this trend of decomposition methods is PGD, proposed by **Ahmed et al.** [3, 4]. This method builds on the classic triangle counting algorithm to count several primitives that are then used to obtain the frequency of each subgraph and orbit. It is currently one of the fastest methods, however it can only count undirected subgraphs of size 3 and 4. Additionally, as most analytic methods, it is highly parallelizable. Due to its versatile nature, PGD has been expanded to other frequency metrics and it stands out as one of the only available efficient methods that can count motifs incident to a vertex or edge of the graph [5], in what is called a "local subgraph count".

More recently, ESCAPE was proposed by **Pinar et al.** [136]. This method is based on a divide and conquer approach that identifies substructures of each counting subgraph to partition them into smaller patterns. It is a very general method, but with the correct choices for decomposition, it is possible to describe a set of formulas to compute the frequency of each subgraph. The original paper only describes the resulting formulas to subgraphs up to size 5, however larger sizes can be obtained with some effort. As of this writing, it is possibly the most efficient algorithm to count undirected subgraphs and orbits up to size 5.

3.3 Theoretical Results

Even though the focus of this work is to look at the proposed practical algorithms, it is important to note that some of the existing work drew inspiration from numerous more theoretical-oriented works. Thus, it is of relevance to briefly summarize some of the achievements in this area and we will do so with a special interest in those that directly influenced some of the algorithms discussed in this section.

The first interest in subgraph counting stemmed from the world of enumeration algorithms. The book “Enumeration in Graphs” [18] surveyed several methods to enumerate several different structures in a graph, such as cycles, trees or cliques. Even though these are specific subpatterns, they often represent the fundamental computation that needs to be done in order to enumerate any subgraph. These ideas were translated into works that count subgraphs by efficiently enumerating simpler substructures like these [71, 83]. Approximation schemes can also be developed with this in mind, which approximates the frequency of several subgraph families like cycles or paths and then generalize these for all size 4 subgraphs [55].

Another example of an initially purely theoretical technique is the work by **Kowaluk et al.** [86], which was one of the inspirations for the multitude of matrix based analytic algorithms for counting subgraphs. In fact, the most efficient algorithms are based on several theoretical foundations that allow a tighter analysis of runtime. Due to this interplay, it is worth mentioned a few more recent papers on subgraph counting and enumerating. There is an interest in finding efficient algorithms that are parameterized or sensitive to certain properties of the graph, such as independent sets [197] or its maximum degree [20]. Another current interest is in counting and enumerating subgraphs in a dynamic or online environment [94]. Finally, another active theoretical topic is to find optimal algorithms for enumeration, as in [50], as well as proving lower bounds on their time complexity, as **Björklund et al.** [21] does for triangle listing.

4 APPROXIMATE COUNTING

Despite the significant advances made towards faster subgraph counting algorithms, current state of the art algorithms that determine exact frequencies still take hours, if not days, for very large networks. With the ever increasing amount of data our society generates (e.g., in big social networks such as Twitter and Facebook new members/nodes join every second), it is unfeasible to count all possible subgraphs. To solve this problem, subgraph counting research drifted towards approximating these frequencies, making a trade-off between losing accuracy but gaining time. Additionally, in some applications, approximate subgraphs counts might be sufficient [79, 146].

Throughout this section we make a distinction between algorithms that estimate (i) subgraph frequencies or (ii) subgraph concentrations. Estimating subgraph frequencies is harder since the algorithm needs to know the magnitude of the values, whereas to estimate concentrations the algorithm only needs to know the different proportions of each subgraph in the network. Obtaining subgraph concentrations from subgraph frequencies is trivial but the reverse requires extra computational tasks.

We further split the approximate counting algorithms in five broad categories: **randomised enumeration**, **enumerate-generalize**, **path sampling**, **random walk**, and **colour coding**. In each subsection, we provide an algorithmic overview of each strategy and delve into the individual algorithms that implement it and how they differ between themselves.

Tables 3 and 4 summarize the algorithms we discuss in the section. We split the methods into algorithms where the full topology is assumed (Table 3) and algorithms tailored to networks with restricted access (Table 4). Although some algorithms from each category may work in the other setting, they excel for the task they were designed for and the distinction should be made clear. The

tables summarize our proposed taxonomy composed of five aspects, ordered by their publication year: (i) the type of **output** (frequencies or concentrations), (ii) **k -restrictions** (does the method only work for certain subgraph sizes?), (iii) **directed** (is the method applicable to directed graphs?), (iv) the **strategy** it employs, according to our taxonomy, and (v) if code is **publicly available**. Note that some authors do not have executable versions publicly available, but will be happy to share them through email. We mark these algorithms with a ✓ in the code column of the table.

Table 3. Algorithms for approximate subgraph counting.

	Year	Output	k -restriction	Directed	Strategy	Code
ESA [79]	2004	Conc.	None	✓	Random Walk	[9]
RAND-ESU [193]	2005	Freq.	None	✓	Rand. Enum.	[194]
TNP [139]	2006	Conc.	5	✗	Enum. - Generalize	✗
RAND-GTRIE [146]	2010	Freq.	None	✓	Rand. Enum.	[144]
GUISE [19]	2012	Conc.	5	✗	Random Walk	[141]
RAND-SCMD [185]	2012	Freq.	None	✓	Enum. - Generalize	✗
WEDGE SAMPLING [169]	2013	Freq.	3	✓	Path Sampling	[84]
GRAFT [142]	2014	Freq.	5	✗	Enum. - Generalize	[140]
PSRW & MSS [187]	2014	Conc.	None	✗	Random Walk	✗
MHRW [159]	2015	Conc.	None	✗	Random Walk	✓
RAND-FASE [131]	2015	Freq.	None	✓	Rand. Enum.	[132]
PATH SAMPLING [73]	2015	Freq.	4	✗	Path Sampling	✗
k -PROFILE SPARSIFIER [45, 46]	2016	Freq.	4	✗	Enum. - Generalize	[44]
MOSS [189]	2018	Freq.	5	✗	Path Sampling	[186]
SSRW [203]	2018	Freq.	7	✗	Random Walk	✗
CC [25]	2018	Freq.	None	✗	Color Coding	[24]

Table 4. Algorithms for approximate subgraph counting with restricted access.

	Year	Output	k -restriction	Directed	Strategy	Code
WRW [59]	2016	Conc.	None	✗	Random Walk	✗
IMPR [31]	2016	Freq.	5	✗	Random Walk	[29]
CSS & NB-SRW [30]	2016	Conc.	None	✗	Random Walk	✓
MINFER [188]	2017	Conc.	5	✓	Enumerate - Generalize	✗

4.1 Randomised Enumeration

These algorithms are adaptations of older enumeration algorithms that perform exact counting. They have the particularity that they all induce a tree-like search space in the computation, where the leaves are the subgraph occurrences, and thus perform the approximation in a similar manner. Each level of the search tree is assigned a value, p_i , which denotes the probability of transitioning from parent node to the child node in the tree. In this scheme, each leaf in this tree is reachable with probability $P = \prod_{i=1}^k p_i$ and the frequency of each subgraph is estimated using the number of samples obtained of that subgraph divided by P .

Figure 5 illustrates how probabilities are added to the search tree. In this specific example, which could be equivalent to searching subgraphs of size 4, the first two levels of the tree have probability 100%, so their successors are all explored. On the other hand, in the last two levels, the probability of exploring a node in the tree is only 80%, therefore some nodes, marked as grey, are not visited.

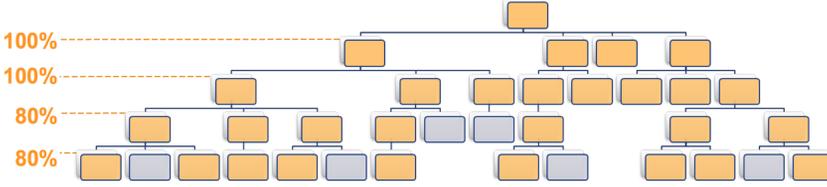


Fig. 5. Example of a tree-like search space induced by a Randomised Enumeration algorithm and a possible distribution of transition probabilities per tree level.

The first algorithm to implement this strategy was RAND-ESU by **Wernicke** [193], an approximate version of ESU (described in Section 3.1.1). Recall that ESU maintains two sets V_S and V_E , the set of vertices in the subgraph and the set of candidate vertices for extending the subgraph. When adding a vertex from V_E to V_S , this vertex is added with probability $p_{|V_S|}$, where $|V_S|$ is the depth of the search tree.

Using the more efficient *g-trie* data structure, **Ribeiro and Silva** [146] proposed RAND-GTRIE and **Paredes and Ribeiro** [131] proposed RAND-FASE. Each level of the *g-trie* is assigned a probability, p_i . When adding a new vertex to a subgraph of size d , corresponding to depth d in the *g-trie*, this is done with probability p_d .

4.2 Enumerate-Generalize

The general idea of these algorithms is to perform an exact count on a smaller network that was obtained from the original one (e.g., a sample, or a compressed network). From the frequencies of each subgraph in the smaller network, the frequencies in the original network are estimated. Algorithms vary on (i) how the smaller network is obtained and on (ii) which estimator they use.

The first example of an algorithm in this category is Targeted Node Processing (TNP) by **Pržulj et al.** [139]. This algorithm is specially tailored for protein-protein interaction, that, according to the authors, have a periphery that is sparser than the more central parts of the network. Using this information, it performs an exact count of the subgraphs in the periphery of the network and uses their frequencies to estimate the frequencies in the rest of the network. The authors claim that, due to the uniformity of the aforementioned networks, the distribution of the subgraphs in the fringe is representative of the distribution in the rest of the network.

SCMD by **Wang et al.** [185] (already covered in Section 3.1.3) allows the use of any approximate counting method in the compressed graph. There is no guarantee that subgraphs are counted uniformly in the compressed graph, introducing a bias that needs to be corrected. The authors give the example of this bias when using their method in conjunction with RAND-ESU. If each leaf (subgraph) of depth k in the search tree is reached with probability P and a specific subgraph in the compressed graph is sampled with probability ρ , then, to correct the sampling bias, the probability of decompressing the relevant k -subgraph is P/ρ .

In GRAFT, **Rahman et al.** [142] provide a strategy for counting undirected graphlets of size up to 5, using edge sampling. The algorithm starts by picking an edge e_g from each of the 29 graphlets and a set of edges sampled from the graph \mathcal{S} , without replacement. For each edge $e \in \mathcal{S}$ and for each graphlet g , the frequency of g is calculated such that e has the same position in g as e_g (e is said to be aligned with e_g). These frequencies are summed for all edges and divided by a

normalising factor, based on the automorphisms of each graphlet, which becomes the estimation for the frequency of that graphlet in the whole network. Note that if S is equal to $E(G)$, the algorithm outputs an exact answer.

Elenberg et al. create estimators for the frequency of size 3 [45] and 4 [46] subgraphs. A major difference from this work to previous ones is that **Elenberg et al.** estimate the frequencies of subgraphs that are not connected, besides the usual connected ones. The authors start by removing each edge from the network with a certain probability and computing the exact counts in this “sub-sampled” network. Then, they craft a set of linear equations that relate the exact counts on this smaller network to the ones of the original network. Using these equations, the estimation of the frequency of the subgraphs in the original network follows.

Wang et al. [188] introduce an algorithm that aims to estimate the subgraph concentrations of a network when only a fraction of its edges are known. They call this a “RESampled Graph”, obtained from the real network through random edge sampling, a common scenario on applications such as network traffic analysis. A key aspect of this algorithm is the number of non-induced subgraphs of a size k graphlet that are isomorphic to another size k graphlet, an example of this calculation can be found in Table 5. Using this number and the proportion of edges sampled to form the smaller network, the authors compute the probability that a subgraph in the “RESampled Graph” is isomorphic to another subgraph in the original graph. Then, an exact counting algorithm is applied to the “RESampled Graph” and by composing the results from this algorithm with the aforementioned probability, the subgraph concentrations in the original network are estimated.

4.3 Path Sampling

This family of algorithms relies on the idea of sampling path subgraphs to estimate the frequencies of the other subgraphs. Path subgraphs are composed by 2 exterior nodes and $k - 2$ interior nodes (where k is the size of the subgraph) arranged in a single line; the interior nodes all have degree of 2, while the exterior nodes have degree of 1. Examples of these are the subgraphs G_1 , G_3 and G_9 in Figure 6. The main idea for these algorithms, mainly for $k \geq 4$, is relating the number of non-induced occurrences of each subgraph of size k in the other size k subgraph. For example, when $k = 4$, there are 4 non-induced occurrences of G_3 in G_5 or 12 non-induced occurrences of G_3 in G_8 . Table 5 shows this full relationship when $k = 4$.

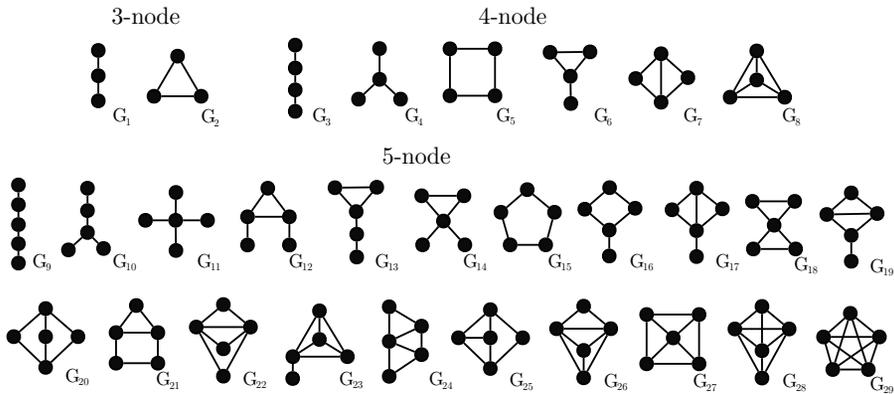


Fig. 6. The 29 isomorphic classes of undirected subgraphs between size 3 and 5.

Seshadhri et al. [169] introduced the idea of *wedge sampling*, where wedges denote size 3 path subgraphs. The premise of the algorithm is simple, they select a number of wedges uniformly at

	g_3	g_4	g_5	g_6	g_7	g_8
g_3	0	1	2	4	6	12
g_4	1	0	1	0	2	4
g_5	0	0	0	1	1	3
g_6	0	0	1	0	4	12
g_7	0	0	0	0	1	6
g_8	0	0	0	0	0	1

Table 5. Number of non-induced occurrences of each undirected graph of size 4 in each other. Position (i, j) in the table indicates the number of times that graph i occurs non-induced in graph j .

random and check whether they are closed or not. The fraction of closed wedges sampled is an estimation from for the clustering coefficient, from which the number of triangles can be derived.

Building on the idea of *wedge sampling*, **Jha et al. [73]** propose *path sampling* to estimate the frequency of size 4 graphlets. The main primitive of the algorithm is sampling non-induced occurrences of G_3 and determining which graphlet is induced by that sample. The estimator relies on both the number of induced subgraphs counted via the sampling and information contained in Table 5. Finally, the authors determine an equation to count the number of stars with 4 nodes (G_4) based on the frequencies of each other graphlet, since G_4 does not have any non-induced occurrence of G_3 .

Applying the same concepts to size 5 subgraphs, **Wang et al. [189]** present MOSS-5. For size 5, sampling paths is not enough to estimate the frequencies of all different subgraphs, as there are 3 subgraphs that do not have a non-induced occurrence of a path: G_{10} , G_{11} and G_{14} . On the other hand, G_{11} does not have a non-induced occurrence in 3 subgraphs as well (G_9 , G_{10} and G_{15}). Using this knowledge, the authors create an algorithm divided in two parts: first it samples non-induced size 5 paths (G_9), similarly to **Jha et al. [73]**, and then they repeat the procedure but sampling occurrences of G_{11} instead. Combining the results from these two sampling schemes, the authors are able to estimate the frequency of every size 5 subgraph.

To the best of our knowledge, MOSS-5 is the algorithm that achieves the best trade-off of accuracy and time to estimate the frequency of 5-subgraphs, as it is able to reach very small errors (magnitude 10^{-2}) with a very limited number of samples, even for big networks. However the ideas behind MOSS-5 are not easily extendable to directed subgraphs and larger sized undirected subgraphs due to the ever increasing number of dependencies between the number of non-induced occurrences, making it harder to use the information contained in a table similar to Table 5 for these cases.

4.4 Random Walk

A random walk in a graph G is a sequence of nodes, R , of the form $R = (n_1, n_2, \dots)$, where n_1 is the seed node and n_i the i th node visited in the walk. A random walk can also be seen as a Markov chain. We identify two main approaches to sample subgraphs using random walks. The first is incrementing the size of the walk until a sequence of k distinct nodes is drawn, forming a k -subgraph, which is then identified by an isomorphism test. The second approach is considering a graph of relationships between subgraphs, where two subgraphs are connected if one can be obtained from the other by adding or removing a node or by adding or removing an edge. A random walk is then performed on this graph instead of on the original one.

Kashtan et al. [79], in their seminal work commonly called ESA (Edge Sampling), implemented one of the first subgraph sampling methods in the MFINDER software. The authors propose to do a random walk on the graph, sampling one edge at a time until a set of k nodes is found, from which

the subgraph induced by that set of nodes is discovered. This method resulted in a biased estimator. To correct the bias, the authors propose to re-weight the sample, which takes exponential time in the size of the subgraphs.

Bhuiyan et al. [19] develop GUISE that computes the graphlet degree distribution for subgraphs of size 3, 4 and 5 in undirected networks. The algorithm is based on Monte Carlo Markov Chain (MCMC) sampling. It works by sampling a seed graphlet, calculating its neighbourhood (a set of other graphlets), picking one randomly and calculating an acceptance probability to transition to this new graphlet. This process is then repeated until a predefined number of samples is taken from the graph. The neighbourhood of a graphlet is similar to the graph of relationships previously mentioned, but to obtain a k -graphlet from another k -graphlet, a node from the original one is removed and, if the remaining $k - 1$ nodes are connected, their adjacency lists are concatenated and nodes are picked from there to form the new k -graphlet.

A similar approach to GUISE is used by **Saha and Al Hasan** [159], where MCMC sampling is also used to compute subgraph concentration. A difference to GUISE is that the size of graphlets is theoretically unbound and only a specific size k is counted, whereas GUISE counts graphlets of size 3, 4 and 5 simultaneously. They also suggest a modified version where the acceptance probability is always one (that is, there is always a transition to the new subgraph), which introduces a bias towards graphlets with nodes with a high degree. In turn, they propose an estimator that re-weights the concentration to remove this bias.

Wang et al. [187] propose a random walk based method to estimate subgraph concentrations that aims to improve on the approach taken by GUISE. The main improvement over GUISE is that no samples are rejected, avoiding a cost of sampling without any gain of information. The authors use a graph of relationships between connected induced subgraphs, where two k -subgraphs are connected if they share $k - 1$ nodes, but this graph is not explicitly built, reducing memory costs. The basic algorithm is just a simple random walk over this graph of relationships. The authors also present two improvements: *Pairwise Subgraph Random Walk* (PSRW), estimates size k subgraph by looking at the graph of relationships composed by $k - 1$ -subgraphs; *Mixed Subgraph Sampling* (MSS), estimates subgraphs of size $k - 1$, k and $k + 1$ simultaneously.

Han and Sethu [59] present an algorithm to estimate subgraph concentration based on random walks. Their algorithm, *Waddling Random Walk* (WRW), gets its name from how the random walk is performed, allowing sampling of nodes not only on the path of the walk, but also query random nodes in the neighbourhood. Let l be the number of vertices (with repetition) in the shortest path of a particular k -graphlet. The goal of the waddling is to reduce the number of steps the walk has to take to identify graphlets with $l > k$. While executing a random walk to identify a k -subgraph, the waddling approach limits the number of nodes explored to the size of the subgraph, k .

Chen and Lui [31] propose a random walk based algorithm to estimate graphlet counts in online social networks, which are often restricted and the entire topology is hidden behind a prohibitive query cost. With this context in mind, the authors introduced the concepts of *touched* and *visible* subgraphs. The former are subgraphs composed of vertices whose neighbourhood is accessible. The latter possess one and only one vertex with inaccessible neighbourhood. Their method, IMPR, works by generating $k - 1$ -node *touched* subgraphs via random walk and combining them with their node's neighbourhood for obtain k -node *visible* subgraphs, which form the k -node samples.

Chen et al. [30] introduce a new framework that incorporates PSRW as a special case. To sample k -subgraphs, the authors also use a graph of relationships between connected induced d -subgraphs, $d \in \{1, \dots, k - 1\}$, and perform a random walk over this graph. The difference to *PSRW* is that *PSRW* only uses d as $k - 1$, which becomes ineffective as k grows to larger sizes. The authors also augment this method of sampling with a different re-weight coefficient to improve estimation accuracy and

add non-backtracking random walks, which eliminates invalid states in the Markov Chain that do not contribute to the estimation.

Yang et al. [203] introduce another algorithm using random walks, *Scalable subgraph Sampling via Random Walk* (SSRW), able to compute both frequencies and concentrations of undirected subgraphs of size up to 7. The next nodes in the random walk are picked from the concatenation of the neighbourhoods of all nodes previously selected to be a part of the sampled subgraph. The authors present an unbiased estimator and compare it against **Chen et al.** [30] and **Han and Sethu** [59], getting better results than both for the single network tested.

4.5 Colour Coding

The technique of colour coding [8] has been adapted to the problem of approximating subgraph frequencies by **Zhao et al.** [206], **Zhao et al.** [207] and **Slota and Madduri** [174]. However, all these works focus on specific categories of subgraphs, for example, SAHAD [207] attempts to only find subgraphs that are in the form of a tree.

More recently, **Bressan et al.** [25] present a general algorithm using colour coding, that works for any undirected subgraph of size theoretically unbound. The algorithm works in two phases. The first, based on the original description of [8], is counting the number of non-induced trees, *treelets*, in the graph but with a particularity, the nodes were previously partitioned into k sets and attributed a label (a *colour*). These treelets then must be constituted solely of nodes with different colours. This part of the algorithm outputs counters $C(T, S, v)$, for every $v \in V(G)$, which are the number of treelets rooted in v isomorphic to T , whose colours span the colour set S .

The second phase of the algorithm is the sampling part, which is focused on sampling treelets uniformly at random. To pick a treelet with k nodes, the authors choose a random node v , a treelet T with probability proportional to $C(T, [k], v)$ and then pick one of the treelets that is rooted in v , is isomorphic to T and is coloured by $[k]$. Given a treelet T_k , the authors consider the graphlet G_k induced by the nodes of T_k and increment its frequency by $\frac{1}{\sigma(G_k)}$, where $\sigma(G_k)$ is the number of spanning trees of G_k .

5 PARALLEL STRATEGIES

By this point it should be clear that subgraph counting is a computationally hard problem. As discussed in Section 3, analytic approaches are much more efficient than enumeration algorithms; however, they are specific to certain sets of small subgraphs. Sampling strategies can produce results in a fraction of the time; but there's a trade-off between time and accuracy. Therefore, speeding up subgraph counting remains a crucial task. The availability of parallel environments, such as multicores, hybrid clusters, and GPUs gave rise to strategies that leverage on these resources. Here we follow a different organizational approach than Sections 3 and 4: we first give an historic overview of the parallel algorithms put forward throughout the years and then we discuss the strategies on a higher level. This is done because most parallel algorithms have a sequential counterpart (already described in previous sections) and many common aspects can be found between the parallel strategies. Table 6 summarizes our proposed taxonomy composed of seven aspects, ordered by their publication year: (i) their computational **platform**, (ii) the **initial work-units** (what part of the graph is divided initially), (iii) the **runtime work-units** (what part of the graph is divided during runtime), (iv) the **search traversal** strategy (how the graph is explored), (v) the **work division** strategy (how work-units are distributed), (vi) how **work sharing** is performed (if applicable) between workers (e.g., CPU processors, or CPU/GPU threads), and (vii) if code is **publicly available**.

Table 6. Parallel algorithms for subgraph counting.

	Year	Platform	Work-units		Search Traversal	Work Division	Work Sharing	Public Code
			Initial	Runtime				
PARWANG [190]	2005	DM	Vertices	✗	DFS	Static	✗	✗
DM-GROCHOW [164]	2008	DM	Isoclasses	Isoclasses	DFS	First-Fit	✗	✗
MPRF [96]	2009	MapReduce	Edges	Subgraphs	BFS	Static	✗	✗
DM-ESU [154]	2010	DM	Vertices	Subgraph-trees	DFS	Diagonal	M-W	✗
DM-GTRIES [151]	2010	DM	Vertices	Subgraph-trees	DFS	Diagonal	W-W	✗
SM-GTRIES [12]	2014	SM	Vertices	Subgraph-trees	DFS	Diagonal	W-W	[145]
SM-FASE [10]	2014	SM	Vertices	Subgraph-trees	DFS	Diagonal	W-W	[145]
SUBENUM [172]	2015	SM	Edges	Subgraphs	DFS	First-Fit	✗	[170]
GPU-ORCA [119]	2015	GPU	Vertices	Subgraphs	BFS	Static	✗	✗
LIN [95]	2015	GPU	Vertices	Subgraphs	BFS	Static	✗	✗
MRSUB [171]	2015	MapReduce	Edges	Subgraphs	BFS	Static	✗	✗
PGD [3]	2015	SM	Edges	✗	DFS	Static	✗	[2]
GPU-PGD [157]	2016	CPU+GPU	Edges	Subgraph-trees	BFS	First-Fit	W-W	✗
ELENBERG [46]	2016	DM	Vertices	Subgraphs	DFS	First-Fit	✗	[44]
MR-GTRIES [124]	2017	MapReduce	Vertices	Subgraph-trees	DFS	Timed	M-W	✗

5.1 Historical Overview

One key aspect necessary to achieve a scalable parallel computation is finding a balanced work division (i.e., splitting work-units *evenly* between workers – parallel processors/threads). A naive possibility for subgraph counting is to assign $\frac{|V(G)|}{|P|}$ nodes from network G to each worker $p \in P$. This egalitarian division is a poor choice since two nodes induce very different search spaces; for instance, *hub*-like nodes induce many more subgraph occurrences than nearly-isolated nodes. Instead of performing an egalitarian division, **Wang et al. [190]** discriminate nodes by their degree and distribute them among workers, the idea being that each worker gets roughly the same amount of *hard* and *easy* work-units. Despite achieving a more balanced division than the naive version, there is still no guarantee that the node-degree is sufficient to determine the actual complexity of the work-unit. Distributing work immediately (without runtime adjustments) is called a **static division**. Wang et al. did not assess scalability in [190], but they showed that their parallel algorithm was faster than MFINDER [121] in an E. Coli transcriptional regulation network. Since their method was not named, we refer to it as PARWANG henceforth.

The first parallel strategy with a **single-subgraph-search** algorithm at its core, namely GROCHOW [56], was by **Schatz et al. [164]**. Since the algorithm was not named, and it targets a **distributed memory (DM)** architecture (i.e., parallel cluster), we refer to it as DM-GROCHOW. In order to distribute query subgraphs (also called **isoclasses**) among workers they employed two strategies: naive and **first-fit**. The naive strategy is similar to PARWANG’s. In the first-fit model, each slave processor requests a subgraph type (or **isoclass**) from the master and enumerates all occurrences of that type (e.g., cliques, stars, chains). This division is **dynamic**, as opposed to static, but it is not balanced since different isoclasses induce very different search trees. For instance, in sparse networks k -cliques are faster to compute than k -chains. Using 64 cores, Schatz et al. obtained ≈ 10 -15x speedups over the sequential version on a yeast PPI network. They also tried another novel approach by partitioning the network instead of partitioning the subgraph-set. However, finding adequate partitions for subgraph counting is a very hard problem due to partition overlaps and subgraphs traversing different partitions, and no speedup was obtained using this strategy. We should note that parallel graph partitioning remains an active research problem to this day [26, 116], but is out of the scope of this work.

All parallel algorithms mentioned so far traverse occurrences in a **depth-first (DFS)** fashion, since doing so avoids having to store intermediate states. By contrast, **Liu et al. [96]** use a **breadth-first search (BFS)** where, at each step, all subgraph occurrences found in the previous one are expanded by one node. Their algorithm, MPRF, is implemented following a **MapReduce** model [39] which is intrinsically a BFS-like framework. In MPRF, mappers extend size k occurrences to size $k + 1$ and reducers remove repeated occurrences. At each BFS-level, MPRF divides work-units evenly among workers. We still consider this to be a static division since no adjustments are made in runtime. Thus, in our terminology, static divisions can be performed only once (at the start of computation in DFS-like algorithms) or multiple times (once per level in BFS-like algorithms). Overhead caused by reading and writing to files reduces MPRF's efficiency, but the authors report speedups of $\approx 7x$ on a 48-node cluster, when compared to the execution on a single-processor.

DFS-based algorithms discussed so far either perform a complete work-division right at the beginning (PARWANG), or they perform a partial work-division at the beginning and then workers request work when idle (DM-GROCHOW). In both cases, a worker has to finish a work-unit before proceeding a new one. Therefore, it is possible that a worker gets stuck processing a very computationally heavy work-unit while all the others are idle. This has to do with work-unit granularity: work-units at the top of the DFS search space have high (coarse) granularity since the algorithm has to explore a large search space. BFS-based algorithms mitigate this problem because work-units are much more fine grained (usually a worker only extends his work-unit(s) by one node). The work by **Ribeiro et al. [154]** was the first to implement **work sharing** during parallel subgraph counting, alleviating the problem of coarse work-unit granularity of DFS-based subgraph counting algorithms. Workers have a splitting threshold that dictates how likely it is to, instead of fully processing a work-unit, putting part of it in a global work queue. A work-unit is divided using **diagonal work splitting** which gathers unprocessed nodes at level k (i.e., nodes that are reached by expanding the current work-unit) and recursively goes up in the search tree, also gathering unprocessed nodes of level $k - i$, $i < k$, until reaching level 1. This process results in a set of finer-grained work-units that induces a more balanced search space than static and first-fit divisions. In [154] Ribeiro et al. use ESU as their core enumeration algorithm and propose a **master-worker (M-W)** architecture where a master-node manages a work-queue and distributes its work-units among slave workers. This strategy, DM-ESU, was the first to achieve near-linear speedups ($\approx 128x$ on a 128-node cluster) on a set of heterogeneous network. A subsequent version [151] used GTRIES as their base algorithm and implemented a **worker-worker (W-W)** architecture where workers perform work stealing. DM-GTRIES improves upon DM-ESU by using a faster enumeration algorithm (GTRIES) and having all workers perform subgraph enumeration (without wasting a node in work queue management). Similar implementations (based on W-W sharing and diagonal splitting) of GTRIES and FASE were also developed for **shared memory (SM) environments**, which achieved near-linear speedups in a 64-core machine [10, 12]. The main advantages of SM implementations is that work sharing is faster (since no message passing is necessary) and SM architectures (such as multicores) are a commodity while DM architectures (such as a cluster) are not.

Instead of developing efficient work sharing strategies, **Shahrivari and Jalili [172]** try to avoid the unbalanced computation induced by vertice-based work-unit division. SUBENUM is an adaptation of ESU which uses edges as starting work-units, achieving near-linear speedup ($\approx 10x$ on a 12-core machine). Using edges as starting work-units is also more suitable for the MapReduce model since edges are finer-grained work-units than vertices. In a follow-up work [171], Shahrivari and Jalili propose a MapReduce algorithm, MRSUB, which greatly improves upon [96], reporting a speedup of $\approx 34x$ on a 40-core machine. Like SUBENUM, MRSUB does not support work sharing between workers. A MapReduce algorithm with work sharing was put forward by **Naser-eddin and Ribeiro [124]**, henceforth called MR-GTRIES. Using work sharing with **timed redistribution**

(i.e., after a certain time, every worker stops and work is fully redistributed), they report a speedup of $\approx 26x$ on a 32-core machine. While MRSUB and MR-GTRIES efficiency is comparable ($\approx 80\%$), the latter has a much faster sequential algorithm at its core; therefore, in terms of absolute runtime, MR-GTRIES is the fastest MapReduce subgraph counting algorithm that we know of.

Graphics processing units (**GPUs**) are processors specialized in image generation, but numerous general purpose tasks have been adapted to them [49, 67, 115]. GPUs are appealing due to their large number of cores, reaching hundreds or thousands of parallel threads whereas commodity multicores typically have no more than a dozen. However, algorithms that rely on graph traversal are not best suited for the GPU framework due to branching code, non-coalesced memory accesses and coarse work-unit granularity [115]. **Milinković et al.** [119] were one of the firsts to follow a GPU approach (GPU-ORCA), with limited success. **Lin et al.** [95] put forward a GPU algorithm (henceforth referred to as LIN since it was unnamed) mostly targeted at network motif discovery but also with some emphasis on efficient subgraph enumeration. LIN avoids duplicate in a similar fashion to ESU [193] and auxiliary arrays are used to mitigate uncoalesced memory accesses. A BFS-style traversal is used (extending each subgraph 1 node at a time) to better balance work-units among threads. They compare LIN running on a 2496-core GPU (Tesla K20) against parallel CPU algorithms and report a speedup of $\approx 10x$ to a 6-core execution of the fastest CPU algorithm, DM-GTRIES.

Rossi and Zhou proposed the first algorithm that **combines multiple GPUs and CPUs** [157]. Their method dynamically distributes work between CPUs and GPUs, where unbalanced computation is given to the CPU whereas GPUs compute the more regular work-units. Since their method was not named, we refer to it as GPU-PGD. Their hybrid CPU-GPU version achieves speedups of $\approx 20x$ to $\approx 200x$ when compared to sequential PGD, depending largely on the network. As mentioned in Section 3, PGD is one of the fastest methods for sequential subgraph counting. As such, GPU-PGD is the fastest subgraph counting algorithm currently available as far as we know. However, GPU-PGD is limited to 4-node subgraphs, while DM-GTRIES is the fastest general approach.

5.2 Platform

Different parallel platforms offer distinct advantages and are more suited for particular strategies. Next we discuss the strategic differences between platforms.

5.2.1 Distributed Memory (DM). A parallel cluster offers the opportunity to use multiple (heterogeneous) machines to speedup computation. Clusters can have hundreds of processors and therefore, if speedup is linear, computation time is reduced from weeks to just a few hours. For work sharing to be efficiently performed on DM architectures one can either have a master-node mediating work sharing [154] or have workers directly steal work from each other [151, 153]. Usually DM approaches are implemented directly using MPI [151–153, 164, 190] but higher level software, such as GraphLab, can also be used [46]. DM has the drawback of workers having to send messages through the network, making network bandwidth a bottleneck.

5.2.2 Shared Memory (SM). SM approaches have the advantage in their underlying hardware being a commodity (multicore computers). Furthermore, workers in a SM environment do not communicate via network messages (since they can communicate directly in main memory), thus avoiding a bottleneck in the network bandwidth. However, the number of cores is usually very low when compared to DM, MapReduce, and GPU architectures. Algorithms on multicores tend to traverse the search space in a DFS fashion [3, 10, 12, 172] thus avoiding the storage of large number of subgraph occurrences in disk or main memory.

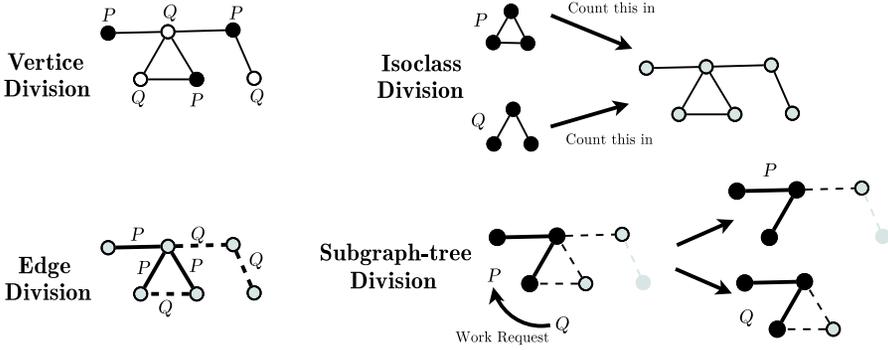


Fig. 7. Different base work-units, and their division to two workers P and Q .

5.2.3 MapReduce. The MapReduce paradigm has been successfully applied to problems where each worker executes very similar tasks, which is the case of subgraph counting. MapReduce is an inherently BFS method, whereas most subgraph counting algorithms are DFS-based. The biggest setback of using MapReduce is the huge amount of subgraph occurrences that are stored in files between each BFS-level iteration (corresponding to a node expansion) [96, 171]. To avoid this setback, one can instead store them in RAM when the number of occurrences fits in memory [124].

5.2.4 GPU. GPUs are very appealing due to their large amount of parallel threads. Despite linear speedups being rare in the GPU, since they have such a large number of cores the gains can still be substantial. However, they are not well-suited for graph traversal algorithms. One of current best pure BFS algorithms [115] on the GPU only achieve a speedup of $\approx 8x$ (on a 448-core NVIDIA C2050) when compared to a 4-core CPU BFS algorithm [89]. By contrast, Monte Carlo calculations on a NVIDIA C2050 GPU achieve a speedup of $\approx 30x$ [41] when compared to a 4-core CPU implementation. This is mainly due to branching problems, uncoalesced memory accesses and coarse work-unit granularity, sometimes leading to almost non-existent speedups in subgraph counting [119]. Using additional memory to efficiently store neighbors and smart work division help achieve some speedup [95]. Another approach is to combine CPUs and GPUs: CPUs handle unbalanced computation while GPUs execute regular computation [157].

5.3 Work-units

When a program can be split into a series of (nearly) independent tasks, efficient parallelism greatly reduces execution times. Each *worker* (be it a thread, CPU-core, or GPU-core) is assigned *work-units* (parallel tasks), either at the start of the computation (*initial work-units*) or during runtime (*runtime work-units*). Work-units in subgraph counting can be either (a) *vertices*, (b) *edges*, (c) *subgraphs*, (d) isomorphic classes (or *isoclasses* for short), or (e) *subgraph-trees*, and each option is described next. Figure 7 illustrates each base work-unit and gives an example of how each can be divided.

5.3.1 Vertices. One possibility is to consider each vertex $v \in V(G)$ as a work-unit and split them among workers. A worker p then computes all size- k subgraph occurrences that contain vertex v . Naive approaches have different workers finding repeated occurrences that need to be removed [190], but efficient sequential algorithms have canonical representations that eliminate this problem [78, 149, 193], making each work-unit independent. Using vertices as work-units has the drawback of creating very coarse work-units: different vertices induce search spaces with very different computational costs. For instance, counting all the subgraph occurrences that start (or eventually reach) a hub-like node is much more time-consuming than counting occurrences of

a nearly isolated node. For algorithms with vertices as work-units to be efficient they can either try to find a good initial division [190] or enable work sharing between workers [10, 12, 152, 153]. Each of these work division strategies is discussed in Section 5.5.

5.3.2 Edges. Due to the unbalanced search tree induced by vertex division, some algorithms use edges as work-units [3, 96, 171, 172]. The idea is similar to vertex division: distribute all $e(v_i, v_j) \in E(G)$ evenly among the workers. An initial edge division guarantees that all workers have an equal amount of 2-node subgraphs, which is not true for vertex division. However, for $k \geq 3$ this strategy offers no guarantees in terms of workload balancing. Therefore, in regular networks (i.e. networks where all nodes have similar clustering coefficients) this strategy achieves a good speedup, but it is not scalable in general. Some methods [157, 164] perform dynamic first-fit division (discussed in Section 5.5.2) instead the simple static division described.

5.3.3 Subgraphs. At the start of computation, only vertices and edges from the network are known. As the k -subgraph counting process proceeds, subgraphs of sizes $k - i, i < k$ are found. Thus, the work-units divided among threads can be these intermediate states (incomplete subgraphs). Some BFS-based algorithms [95, 96, 119, 171] begin with either edges or vertices as initial work-units and, at the end of each BFS-level, intermediate subgraphs are found and divided among workers. DFS-based methods expand each subgraph work-unit by one node until they reach a k -subgraph [46, 171].

5.3.4 Isoclass. Instead of partitioning the graph, like the previous three strategies do, one can instead chose to partition the set of isoclasses being enumerated [164]. Work-units split in this fashion have similar problems to the previously discussed: isomorphic classes do not induce computationally equivalent search spaces. For instance, in sparse networks it is much more time-consuming to enumerate chains or hubs than cliques.

5.3.5 Subgraph-trees. This approach is applicable only for DFS-like algorithms where, since the search tree is explored in a depth-first fashion, a work-tree is implicitly built during enumeration: when the algorithm is at level k of the search, unexplored candidates of stages $\{k - 1, k - 2, \dots, 1\}$ were previously generated. Then, instead of splitting top vertices from stage 1 only (as described in Section 5.3.1), the search-tree is split among sharing processors [10, 12, 151, 152] (more details on this in Section 5.5.3). Subgraph-trees are *expected* to be similar since both coarse- and fine-grained work-units are generated. Nevertheless, it is not guaranteed that work-units from the same level of the search tree induce similar work. This strategy also incurs the additional complexity of building the candidate-set of each level and splitting them among workers.

5.4 Search Traversal

Discounting analytic approaches presented in Section 3.2, subgraph counting algorithms typically count occurrences by traversing the graph. How graph traversal is performed greatly influences the parallel performance and is dependent on the platform, as is discussed next.

5.4.1 Breadth-First Search. Algorithms that adopt this strategy are typically MapReduce methods [96, 124, 171] or GPU [95, 119, 157] approaches. MapReduce works intrinsically in BFS fashion, and GPUs are very inefficient when work is unbalanced and contains branching code. BFS starts by (i) splitting edges among workers, (ii) the processors compute the patterns of size-3 from each edge (size-2 subgraphs), (iii) the patterns of size-3 are themselves split among processors and (iv) this process is repeated until the desired size- k patterns are obtained. The idea of BFS is to give large amounts of fine-grained work-units to each worker, thus making work division more balanced since these work-units induce similar work, making this approach more suitable for methods that

require regular data. However, the main drawback is that these algorithms need to store partial results (which grow exponentially as k increases) and synchronize at the end of each BFS-level.

5.4.2 Depth-First Search. To avoid the cost of synchronization and of storing partial results, most subgraph counting algorithms traverse the search space in a depth-first fashion [3, 10, 12, 56, 151–153, 172, 190]. This means that the algorithm starts with $V_{sub} = \{v\}$ and incrementally adds a new node to V_{sub} until it obtains a match of the desired size, and backtracks to find new matches. This strategy leads to unbalanced search spaces, caused by coarse-grained work-units, that need to be controlled.

5.5 Work Division

Splitting work is obviously essential for a parallel approach. Work can be divided at two moments: (i) an initial work division before subgraph counting starts and/or (ii) divisions during runtime.

5.5.1 Static. The simplest form of work division is to produce an initial distribution of work-units and proceed with the parallel computation, without ever spending time dividing work during runtime. Trying to obtain an estimation of the work beforehand [157, 190] is valuable but limited: if the estimation is done quickly but is not very precise (such as using node-degrees or clustering coefficients to estimate work-unit difficulty) little guarantees are offered that the work division is balanced, and obtaining a very precise estimation is as computationally expensive as doing subgraph enumeration itself. Following a BFS approach [96, 119, 171] helps balancing out the work-units and a static work division at each BFS-level is usually sufficient to obtain good results. However, those strategies have limitations as discussed in Section 5.4.1. Some analytic works, which do not rely on explicit subgraph enumeration, do not need advanced work division strategies because their algorithm is almost embarrassingly parallel [158].

5.5.2 Dynamic: First-fit. Instead of trying to estimate a good division one can generate work on-demand during runtime. One isomorphic class [164] or small portions of the graph [172] can be initially given at each processor and, when that computation is done, idle processors request more work. This strategy has the penalty of maintaining a global queue of work-units to be processed. Furthermore, the last $|P|$ work-units (where $|P|$ is the number of workers) can have different granularities (and thus computational cost), so the speedup is largely dependent on how well-balanced they are.

5.5.3 Dynamic: Diagonal Work Splitting. Algorithms that employ this strategy [10, 12, 151, 152] perform an initial static work division. They do not need a sophisticated criteria to choose to whom work-units are assigned because work will be dynamically redistributed during runtime: whenever workers are idle, some work will be relocated from busy workers to them. Furthermore, instead of simply giving half of their top-level work-units away and keeping the other half, a busy worker fully splits its work tree. The main idea is to build work-units of both fine- and coarse-grained sizes, and this is particularly helpful in cases where a worker becomes stuck managing a very complex initial work-unit; this way, that work-unit is split in half, and it can be split iteratively to other workers if needed. These work-units can then either be stored in a global work queue, which a master worker is responsible of managing [151, 152], or sharing is conducted between worker threads themselves [10, 12] (more details on Sections 5.6.1 and 5.6.2, respectively).

5.5.4 Dynamic: Timed Redistribution. Timed Redistribution is a way to avoid estimating work during runtime while guaranteeing that every worker has work (after a while). Workers first receive work and try to process as much as they can. After a certain time, they all stop and work is redistributed. This strategy is specially useful when worker communication is not practical,

such as in a MapReduce environment [124] on in the GPU. Setting an adequate threshold for work redistribution has a great impact: redistributing work too quickly has the drawback of wasting too much time in work division, while redistributing work too late has the drawback of having idle workers. One solution is to use an adaptive threshold [124]: if workers are too often without work, the threshold of the next iteration is lower, if workers are too often with much work left to compute, the threshold of the next iteration is higher.

5.6 Work Sharing

Since work is unbalanced for enumeration algorithms, work sharing can be used in order to balance work during runtime.

5.6.1 Master-Worker (M-W). This type of work sharing is mostly adopted in distributed memory (DM) environments since workers do not share positions of memory that they can easily access and use to communicate. A master worker initially splits the work-units among the workers (slaves) and then manages load balancing. Load balancing can be achieved by managing a global queue where slaves put some of their work, to be later redistributed by the master [154]. This strategy implies that the master is not being used the enumeration and that there is a need communication over the network.

5.6.2 Worker-Worker (W-W). Shared memory (SM) environments allow for direct communication between workers, therefore a master node is redundant. In this strategy, an idle worker asks a random worker for work [10, 12]. One could try to estimate which worker should be polled for work (which is computationally costly) but random polling has been established as an efficient heuristic for dynamic load balancing [160]. After the sharing process, computation resumes with each worker evolved in the exchange computing their part of the work. Computation ends when all workers are polling for work. This strategy achieves a balanced work-division during runtime, and the penalty caused by worker communication is negligible [10, 12]. Most implementations of W-W sharing are built on top of relatively homogeneous systems, such as multiworkered CPUs [172] or clusters of similar processors [164]. In these systems, since all workers are equivalent, it is irrelevant which ones get a specific easy (or hard) work-unit, thus only load balancing needs to be controlled. Strategies that combine CPUs with GPUs, for instance, can split tasks in a way that takes advantage of both architectures: GPUs are very fast for regular tasks while CPUs can deal with irregular ones. For instance, a shared deque can be kept where workers, either GPUs or CPUs, put work on or take work from [157]; the queue is ordered by complexity: complex tasks are placed at the front, and simple tasks at the end. The main idea is that CPUs handle just a few complex work-units from the front of the deque while GPUs take large bundles of work-units from the back.

6 CONCLUDING REMARKS

Over the last twenty years, subgraph counting has been under increased focus in the network science community, specially since the introduction of networks motifs [121], and its status as an important tool for network analysis, as well as graphlets [138] which are now established measures for network alignment.

In this survey we explored existing practical methods to solve the subgraph counting problem from three perspectives: (i) algorithms that efficiently perform exact counting, which is an intrinsically computationally hard task, (ii) algorithms that perform an approximation of subgraph frequencies, making the process faster but taking into account the accuracy of their estimation, and (iii) algorithms that efficiently exploit parallel architectures despite the unbalanced nature of subgraph counting. We showed that all three of these categories are still attracting new work, with new methods still emerging in an attempt to improve previous work.

The aim of this work was precisely to describe and classify the major algorithmic ideas in each of these three categories, offering a structured and thorough review of how they work and what are their advantages and disadvantages. Moreover, we provided more than two hundred references that allow further exploration of any aspects that might be of particular interest to the reader, including direct links to the existing practical implementations of the methods.

We feel that this survey provides valuable insight both from a more practical point of view, offering solutions and application ideas for those who view subgraph counting as a tool for network analysis, and from a more methodological angle, being not only a very strong starting point for new researchers joining the area, but also a very useful and comprehensive summary of recent research results for more established researchers.

REFERENCES

- [1] Christoph Adami, Jifeng Qian, Matthew Rupp, and Arend Hintze. 2011. Information content of colored motifs in complex networks. *Artificial Life* 17, 4 (2011), 375–390.
- [2] Nesreen K. Ahmed. 2018. A Parallel Graphlet Decomposition Library for Large Graphs. <https://github.com/nkahmed/PGD>. Accessed: 2019-10-09.
- [3] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 1–10.
- [4] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, Nick G Duffield, and Theodore L Willke. 2017. Graphlet decomposition: Framework, algorithms, and applications. *Knowledge and Information Systems* 50, 3 (2017), 689–722.
- [5] Nesreen K Ahmed, Theodore L Willke, and Ryan A Rossi. 2016. Estimation of local subgraph counts. In *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 586–595.
- [6] Mohammad Al Hasan and Vachik S Dave. 2018. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 2 (2018), e1226.
- [7] Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. 2018. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica* 80, 2 (2018), 668–697.
- [8] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *Journal of the ACM (JACM)* 42, 4 (1995), 844–856.
- [9] Uri Alon. 2018. Network Motif Software. <https://www.weizmann.ac.il/mcb/UriAlon/download/network-motif-software>. Accessed: 2019-10-09.
- [10] David Aparicio, Pedro Paredes, and Pedro Ribeiro. 2014. A Scalable Parallel Approach for Subgraph Census Computation. In *European Conference on Parallel Processing*. Springer, 194–205.
- [11] David Aparicio, Pedro Ribeiro, Tijana Milenković, and Fernando Silva. 2019. Temporal network alignment via GoT-WAVE. *Bioinformatics* 35, 18 (2019), 3527–3529.
- [12] David Aparicio, Pedro Ribeiro, and Fernando Silva. 2014. Parallel subgraph counting for multicore architectures. In *Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on*. IEEE, 34–41.
- [13] David Aparicio, Pedro Ribeiro, and Fernando Silva. 2016. Extending the Applicability of Graphlets to Directed Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2016).
- [14] David Aparicio, Pedro Ribeiro, and Fernando Silva. 2018. Graphlet-orbit Transitions (GoT): A fingerprint for temporal network comparison. *PLoS one* 13, 10 (2018), e0205497.
- [15] Albert-László Barabási et al. 2016. *Network science*. Cambridge university press.
- [16] Jordi Bascompte and Carlos J Melián. 2005. Simple trophic modules for complex food webs. *Ecology* 86, 11 (2005), 2868–2873.
- [17] Joris Bekkers and Shaunak Dabodghao. 2017. Flow Motifs in Soccer: What can passing behavior tell us? *Journal of Sports Analytics* Preprint (2017), 1–13.
- [18] MA Bezem and Jan van Leeuwen. 1987. *Enumeration in graphs*. Vol. 87. Unknown Publisher.
- [19] Mansurul A Bhuiyan, Mahmudur Rahman, and M Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, 91–100.
- [20] Andreas Bjorklund, Thore Husfeldt, Petteri Kaski, Mikko Kalle Henrik Koivisto, et al. 2018. Counting connected subgraphs with maximum-degree-aware sieving. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- [21] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. 2014. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*. Springer, 223–234.
- [22] Peter Bloem and Steven de Rooij. 2017. Large-scale network motif learning with compression. *CoRR arXiv* 1701 (2017).

- [23] Hanjo D Boekhout, Walter A Kusters, and Frank W Takes. 2019. Efficiently counting complex multilayer temporal motifs in large-scale networks. *Computational Social Networks* 6, 1 (2019), 1–34.
- [24] Marco Bressan. 2018. Motif Counting Beyond Five Nodes. <https://github.com/Steven--/graphlets>. Accessed: 2019-10-09.
- [25] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 4 (2018), 48.
- [26] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. Recent advances in graph partitioning. In *Algorithm Engineering*. Springer, 117–158.
- [27] Robrecht Cannoodt, Joeri Ruysinck, Jan Ramon, Katleen De Preter, and Yvan Saeys. 2018. IncGraph: Incremental graphlet counting for topology optimisation. *PLoS one* 13, 4 (2018), e0195997.
- [28] Raphaël Charbey and Christophe Prieur. 2019. Stars, holes, or paths across your Facebook friends: A graphlet-based characterization of many networks. *Network Science* (2019), 1–22.
- [29] Xiaowei Chen. 2018. Mining Graphlet Counts in Online Social Networks. <https://github.com/xwchen666/GraphletCountOSN>. Accessed: 2019-10-09.
- [30] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John Lui. 2016. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment* 10, 3 (2016), 253–264.
- [31] Xiaowei Chen and John CS Lui. 2016. Mining Graphlet Counts in Online Social Networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 71–80.
- [32] Sarvenaz Choobdar, Pedro Ribeiro, Sylwia Bugla, and Fernando Silva. 2012. Comparison of co-authorship networks across scientific fields using motifs. In *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*. IEEE, 147–152.
- [33] Sarvenaz Choobdar, Pedro Ribeiro, and Fernando Silva. 2012. Motif Mining in Weighted Networks. In *2nd IEEE ICDM Workshop on Data Mining in Networks*. IEEE, 210–217. <https://doi.org/10.1109/ICDMW.2012.111>
- [34] Stephen A Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 151–158.
- [35] Luciano da Fontoura Costa, Osvaldo N Oliveira Jr, Gonzalo Travieso, Francisco Aparecido Rodrigues, Paulino Ribeiro Villas Boas, Lucas Antiquiera, Matheus Palhares Viana, and Luis Enrique Correa Rocha. 2011. Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics* 60, 3 (2011), 329–412.
- [36] Éva Czabarka, László A Székely, and Stephan Wagner. 2018. On the number of nonisomorphic subtrees of a tree. *Journal of Graph Theory* 87, 1 (2018), 89–95.
- [37] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 589–598.
- [38] Vachik S Dave, Nesreen K Ahmed, and Mohammad Al Hasan. 2017. E-CLoG: counting edge-centric local graphlets. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 586–595.
- [39] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
- [40] Sofie Demeyer, Tom Michoel, Jan Fostier, Pieter Audenaert, Mario Pickavet, and Piet Demeester. 2013. The index-based subgraph matching algorithm (SMA): fast subgraph enumeration in large networks using optimized search trees. *PLoS one* 8, 4 (2013), e61183.
- [41] Aiping Ding, Tianyu Liu, Chao Liang, Wei Ji, Mark S Shephard, X George Xu, and Forrest B Brown. 2011. Evaluation of speedup of Monte Carlo calculations of two simple reactor physics problems coded for the GPU/CUDA environment. (2011).
- [42] Derek Doran. 2014. Triad-based role discovery for large social systems. In *International Conference on Social Informatics*. Springer, 130–143.
- [43] Alexandra Duma and Alexandru Topirceanu. 2014. A network motif based approach for classifying online social networks. In *2014 IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, 311–315.
- [44] Ehtna R. Elenberg. 2016. GraphLab PowerGraph implementation of 4-profile counting. <https://github.com/eelenberg/4-profiles>. Accessed: 2019-10-09.
- [45] Ethan R Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G Dimakis. 2015. Beyond triangles: A distributed framework for estimating 3-profiles of large graphs. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 229–238.
- [46] Ethan R Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G Dimakis. 2016. Distributed estimation of graph 4-profiles. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 483–493.
- [47] Rasha Elhessa and Tamer Kahveci. 2016. Identification of large disjoint motifs in biological networks. *BMC bioinformatics* 17, 1 (2016), 408.

- [48] David Eppstein. 2002. Subgraph isomorphism in planar graphs and related problems. In *Graph Algorithms and Applications I*. World Scientific, 283–309.
- [49] Wenbin Fang, Ka Keung Lau, Mian Lu, Xiangye Xiao, Chi K Lam, Philip Yang Yang, Bingsheng He, Qiong Luo, Pedro V Sander, and Ke Yang. 2008. Parallel data mining on graphics processors. *Hong Kong Univ. Sci. and Technology, Hong Kong, China, Tech. Rep. HKUST-CS08-07* (2008).
- [50] Rui Ferreira. 2013. Efficiently Listing Combinatorial Patterns in Graphs. *arXiv preprint arXiv:1308.6635* (2013).
- [51] Irene Finocchi, Marco Finocchi, and Emanuele G Fusco. 2015. Clique counting in MapReduce: algorithms and experiments. *Journal of Experimental Algorithmics (JEA)* 20 (2015), 1–7.
- [52] Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. 2015. Induced subgraph isomorphism: Are some patterns substantially easier than others? *Theoretical Computer Science* 605 (2015), 119–128.
- [53] Ali Gholami Rudi, Saeed Shahrivari, Saeed Jalili, and Zahra Razaghi Moghadam Kashani. 2013. RANGI: a fast list-colored graph motif finding algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 10, 2 (2013), 504–513.
- [54] Mira Gonen, Dana Ron, and Yuval Shavitt. 2011. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Discrete Mathematics* 25, 3 (2011), 1365–1411.
- [55] Mira Gonen and Yuval Shavitt. 2009. Approximating the number of network motifs. *Internet Mathematics* 6, 3 (2009), 349–372.
- [56] Joshua A Grochow and Manolis Kellis. 2007. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Annual International Conference on Research in Computational Molecular Biology*. Springer, 92–106.
- [57] Shawn Gu, John Johnson, Fazle E Faisal, and Tijana Milenković. 2018. From homogeneous to heterogeneous network alignment via colored graphlets. *Scientific reports* 8, 1 (2018), 12524.
- [58] Sylvain Guillemot and Florian Sikora. 2013. Finding and counting vertex-colored subtrees. *Algorithmica* 65, 4 (2013), 828–844.
- [59] Guyue Han and Harish Sethu. 2016. Waddling Random Walk: Fast and Accurate Mining of Motif Statistics in Large Graphs. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 181–190.
- [60] Frank Harary. 1974. A survey of the reconstruction conjecture. In *Graphs and combinatorics*. Springer, 18–28.
- [61] Himamshu and Sarika Jain. 2017. Impact of Memory Space Optimization Technique on Fast Network Motif Search Algorithm. In *Advances in Computer and Computational Sciences*. Springer, 559–567.
- [62] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.
- [63] Tomaž Hočevar and Janez Demšar. 2017. Combinatorial algorithm for counting small induced graphs and orbits. *PLoS one* 12, 2 (2017), e0171428.
- [64] Tomaž Hočevar and Janez Demšar. 2018. Orca. <http://www.biolab.si/supp/orca/>. Accessed: 2019-10-09.
- [65] Paul W Holland and Samuel Leinhardt. 1976. Local structure in social networks. *Sociological methodology* 7 (1976), 1–45.
- [66] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.
- [67] Sungpack Hong, Tayo Oguntebi, and Kunle Olukotun. 2011. Efficient parallel graph exploration on multi-core CPU and GPU. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*. IEEE, 78–88.
- [68] Maarten Houbraeken, Sofie Demeyer, Tom Michoel, Pieter Audenaert, Didier Colle, and Mario Pickavet. 2014. The Index-based Subgraph Matching Algorithm with General Symmetries (ISMAGS): exploiting symmetry for faster subgraph enumeration. *PLoS one* 9, 5 (2014), e97896.
- [69] Jun Huan, Wei Wang, and Jan Prins. 2003. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Third IEEE International Conference on Data Mining*. IEEE, 549–552.
- [70] Yuriy Hulovatyy, Huili Chen, and T Milenković. 2015. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinformatics* 31, 12 (2015), i171–i180.
- [71] Royi Itzhack, Yelena Mogilevski, and Yoram Louzoun. 2007. An optimal algorithm for counting network motifs. *Physica A: Statistical Mechanics and its Applications* 381 (2007), 482–490.
- [72] Deepali Jain and Ripon Patgiri. 2019. Network Motifs: A Survey. In *International Conference on Advances in Computing and Data Sciences*. Springer, 80–91.
- [73] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 495–505.
- [74] Chuntao Jiang, Frans Coenen, and Michele Zito. 2013. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review* 28, 01 (2013), 75–105.
- [75] Zhao Jing and Zhong Cheng. 2015. HashESU: Efficient Algorithm for Identifying Motifs in Biological Networks. *Journal of Chinese Computer Systems* 9 (2015), 024.

- [76] Yuval Kalish and Garry Robins. 2006. Psychological predispositions and network structure: The relationship between individual predispositions, structural holes and network closure. *Social networks* 28, 1 (2006), 56–84.
- [77] John Kallaugher, Michael Kapralov, and Eric Price. 2018. The sketching complexity of graph and hypergraph counting. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 556–567.
- [78] Zahra Razaghi Moghadam Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari-Dalini, Elnaz Saberi Ansari, Sahar Asadi, Shahin Mohammadi, Falk Schreiber, and Ali Masoudi-Nejad. 2009. Kavosh: a new algorithm for finding network motifs. *BMC bioinformatics* 10, 1 (2009), 318.
- [79] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. 2004. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* 20, 11 (2004), 1746–1758.
- [80] Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter, Ina Koch, and Ali Masoudi-Nejad. 2013. QuateXelero: an accelerated exact network motif detection algorithm. *PLoS one* 8, 7 (2013), e68073.
- [81] Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter, Ina Koch, and Ali Masoudi-Nejad. 2018. QuateXelero – Fast Motif Detection algorithm. <http://apps.cytoscape.org/apps/ismags>. Accessed: 2019-10-09.
- [82] Wooyoung Kim, Martin Diko, and Keith Rawson. 2013. Network motif detection: Algorithms, parallel and cloud computing, and related tools. *Tsinghua science and technology* 18, 5 (2013), 469–489.
- [83] Ton Kloks, Dieter Kratsch, and Haiko Müller. 2000. Finding and counting small induced subgraphs efficiently. *Inform. Process. Lett.* 74, 3-4 (2000), 115–121.
- [84] Tamara Kolda, Ali Pinar, and C. Seshadhri. 2018. Triadic Measures on Graphs: The Power of Wedge Sampling. <http://www.sandia.gov/~tgkolda/feastpack/>. Accessed: 2019-10-09.
- [85] Michel Koskas, Gilles Grasseau, Etienne Birmelé, Sophie Schbath, and Stéphane Robin. 2011. NeMo: Fast count of network motifs. *Book of Abstracts for Journées Ouvertes Biologie Informatique Mathématiques (JOBIM)* (2011), 53–60.
- [86] Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. 2013. Counting and detecting small subgraphs via equations. *SIAM Journal on Discrete Mathematics* 27, 2 (2013), 892–909.
- [87] Oleksii Kuchaiev, Tijana Milenković, Vesna Memišević, Wayne Hayes, and Nataša Pržulj. 2010. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society Interface* 7, 50 (2010), 1341–1354.
- [88] Oleksii Kuchaiev and Nataša Pržulj. 2011. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics* 27, 10 (2011), 1390–1396.
- [89] Charles E Leiserson and Tao B Schardl. 2010. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 303–314.
- [90] Ted G Lewis. 2011. *Network science: Theory and applications*. John Wiley & Sons.
- [91] Guanghui Li, Jiawei Luo, Zheng Xiao, and Cheng Liang. 2018. MTMO: an efficient network-centric algorithm for subtree counting and enumeration. *Quantitative Biology* 6, 2 (2018), 142–154.
- [92] Xin Li, Douglas S Stones, Haidong Wang, Hualiang Deng, Xiaoguang Liu, and Gang Wang. 2012. Netmode: Network motif detection without nauty. *PLoS one* 7, 12 (2012), e50093.
- [93] Xin Li, Douglas S Stones, Haidong Wang, Hualiang Deng, Xiaoguang Liu, and Gang Wang. 2016. NetMODE SourceForge.net. <https://sourceforge.net/projects/netmode/>. Accessed: 2019-10-09.
- [94] Min Chih Lin, Francisco J Soullignac, and Jayme L Szwarcfiter. 2012. Arboricity, h-index, and dynamic algorithms. *Theoretical Computer Science* 426 (2012), 75–90.
- [95] Wenqing Lin, Xiaokui Xiao, Xing Xie, and Xiao-Li Li. 2015. Network motif discovery: A GPU approach. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 831–842.
- [96] Yang Liu, Xiaohong Jiang, Huajun Chen, Jun Ma, and Xiangyu Zhang. 2009. Mapreduce-based pattern finding algorithm applied in motif detection for prescription compatibility network. In *International Workshop on Advanced Parallel Processing Technologies*. Springer, 341–355.
- [97] Jiawei Luo, Lv Ding, Cheng Liang, and Nguyen Hoang Tu. 2018. An efficient network motif discovery approach for co-regulatory networks. *IEEE Access* 6 (2018), 14151–14158.
- [98] Ben D MacArthur, Rubén J Sánchez-García, and James W Anderson. 2008. Symmetry in complex networks. *Discrete Applied Mathematics* 156, 18 (2008), 3525–3531.
- [99] Ravindranath Madhavan, Devi R Gnyawali, and Jinyu He. 2004. Two’s company, three’s a crowd? Triads in cooperative-competitive networks. *Academy of Management Journal* 47, 6 (2004), 918–927.
- [100] Noël Malod-Dognin and Nataša Pržulj. 2015. L-GRAAL: Lagrangian graphlet-based network aligner. *Bioinformatics* 31, 13 (2015), 2182–2189.
- [101] Shmoolik Mangan and Uri Alon. 2003. Structure and function of the feed-forward loop network motif. *Proceedings of the National Academy of Sciences* 100, 21 (2003), 11980–11985.
- [102] Dror Marcus and Yuval Shavitt. 2010. Efficient counting of network motifs. In *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*. IEEE, 92–98.

- [103] Dror Marcus and Yuval Shavitt. 2012. Rage—a rapid graphlet enumerator for large networks. *Computer Networks* 56, 2 (2012), 810–819.
- [104] Dror Marcus and Yuval Shavitt. 2018. NeMo R Package (CRAN archive). <http://www.eng.tau.ac.il/~shavitt/RAGE/Rage.htm>. Accessed: 2019-10-09.
- [105] Ali Masoudi-Nejad, Falk Schreiber, and Zahra Razaghi Moghadam Kashani. 2012. Building blocks of biological networks: a review on major network motif discovery algorithms. *IET systems biology* 6, 5 (2012), 164–174.
- [106] Brendan D McKay. 2003. *nauty user's guide (version 2.2)*. Technical Report. Technical Report TR-CS-9002, Australian National University.
- [107] Brendan D McKay et al. 1981. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, USA.
- [108] Brendan D McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60 (2014), 94–112.
- [109] Luís AA Meira, Vinicius R. Máximo, Álvaro L Fazenda, and Arlindo F da Conceição. 2018. acc-Motif: Accelerated Motif Detection. <https://www.ft.unicamp.br/docentes/meira/accmotifs/>. Accessed: 2019-10-09.
- [110] Luis AA Meira, Vinicius R Maximo, Alvaro L Fazenda, and Arlindo F da Conceicao. 2012. Accelerated motif detection using combinatorial techniques. In *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*. IEEE, 744–753.
- [111] Luis AA Meira, Vinicius R Máximo, Álvaro L Fazenda, and Arlindo F Da Conceição. 2014. Acc-motif: accelerated network motif detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 11, 5 (2014), 853–862.
- [112] Ine Melckenbeek, Pieter Audenaert, Didier Colle, and Mario Pickavet. 2017. Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. *Bioinformatics* 34, 8 (11 2017), 1372–1380.
- [113] Ine Melckenbeek, Pieter Audenaert, Thomas Van Parys, Yves Van De Peer, Didier Colle, and Mario Pickavet. 2019. Jesse - Tree-based algorithm to calculate graphlet densities of nodes in a graph using equations. <https://github.com/biointec/jesse>. Accessed: 2019-10-09.
- [114] Ine Melckenbeek, Pieter Audenaert, Thomas Van Parys, Yves Van De Peer, Didier Colle, and Mario Pickavet. 2019. Optimising orbit counting of arbitrary order by equation selection. *BMC bioinformatics* 20, 1 (2019), 27.
- [115] Duane Merrill, Michael Garland, and Andrew Grimshaw. 2012. Scalable GPU graph traversal. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 117–128.
- [116] Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2017. Parallel graph partitioning for complex networks. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2625–2638.
- [117] Giovanni Micale, Rosalba Giugno, Alfredo Ferro, Misael Mongiovi, Dennis Shasha, and Alfredo Pulvirenti. 2018. Fast analytical methods for finding significant labeled graph motifs. *Data Mining and Knowledge Discovery* 32, 2 (2018), 504–531.
- [118] Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. 2010. Optimal network alignment with graphlet degree vectors. *Cancer informatics* 9 (2010), 121.
- [119] Aleksandar Milinković, Stevan Milinković, and L Lazić. [n. d.]. A contribution to acceleration of graphlet counting. In *Infoteh Jahorina Symposium*, Vol. 14. 741–745.
- [120] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. 2004. Superfamilies of evolved and designed networks. *Science* 303, 5663 (2004), 1538–1542.
- [121] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.
- [122] Shahin Mohammadi. 2014. Kavosh: a new algorithm for finding network motifs. <https://github.com/shmohammadi86/Kavosh>. Accessed: 2019-10-09.
- [123] Misael Mongiovi, Giovanni Micale, Alfredo Ferro, Rosalba Giugno, Alfredo Pulvirenti, and Dennis Shasha. 2018. gLabTrie: A Data Structure for Motif Discovery with Constraints. In *Graph Data Management*. Springer, 71–95.
- [124] Ahmad Naser-eddin and Pedro Ribeiro. 2017. Scalable subgraph counting using MapReduce. In *Proceedings of the Symposium on Applied Computing*. ACM, 1574–1581.
- [125] Siegfried Nijssen and Joost N Kok. 2005. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science* 127, 1 (2005), 77–87.
- [126] Saeed Omid, Falk Schreiber, and Ali Masoudi-Nejad. 2009. MODA: an efficient algorithm for network motif discovery in biological networks. *Genes & genetic systems* 84, 5 (2009), 385–395.
- [127] Mark Ortmann and Ulrik Brandes. 2016. Quad census computation: simple, efficient, and orbit-aware. In *Proceedings of the 12th International Conference and School on Advances in Network Science-Volume 9564*. Springer-Verlag New York, Inc., 1–13.
- [128] Mark Ortmann and Ulrik Brandes. 2017. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied Network Science* 2, 1 (2017), 13.

- [129] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 601–610.
- [130] Pedro Paredes and Pedro Ribeiro. 2013. Towards a faster network-centric subgraph census. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*. IEEE, 264–271.
- [131] Pedro Paredes and Pedro Ribeiro. 2015. Rand-FaSE: fast approximate subgraph census. *Social Network Analysis and Mining* 5, 1 (2015), 17.
- [132] Pedro Paredes and Pedro Ribeiro. 2018. FaSE - Fast Subgraph Enumeration. <https://github.com/ComplexNetworks-DCC-FCUP/fase>. Accessed: 2019-10-09.
- [133] Thomas V Parys and Ine Melckenbeeck. 2016. ISMAGS - Enumerate all instances of a motif in a graph, making optimal use of the motif’s symmetries. <http://apps.cytoscape.org/apps/ismags>. Accessed: 2019-10-09.
- [134] Sabyasachi Patra and Anjali Mohapatra. 2018. Motif discovery in biological network using expansion tree. *Journal of bioinformatics and computational biology* 16, 6 (2018), 1850024–1850024.
- [135] Franck Picard, J-J Daudin, Michel Koskas, Sophie Schbath, and Stephane Robin. 2008. Assessing the exceptionality of network motifs. *Journal of Computational Biology* 15, 1 (2008), 1–20.
- [136] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1431–1440.
- [137] Christina Prell and John Skvoretz. 2008. Looking at social capital through triad structures. *Connections* 28, 2 (2008), 4–16.
- [138] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23 (2007), 177–183.
- [139] N Pržulj, Derek G Corneil, and Igor Jurisica. 2006. Efficient estimation of graphlet frequency distributions in protein–protein interaction networks. *Bioinformatics* 22, 8 (2006), 974–980.
- [140] Mahmudur Rahman, Mansurul Bhuiyan, and Mahmuda Rahman. 2018. GRAFT: an approximate graphlet counting algorithm for large graph analysis. <https://github.com/DMGroup-IUPUI/GRAFT-Source>. Accessed: 2019-10-09.
- [141] Mahmudur Rahman, Mansurul Bhuiyan, Mahmuda Rahman, and Mohammad Al Hasan. 2018. GUISE: Uniform Sampling of Graphlets for Large Graph Analysis. <https://github.com/DMGroup-IUPUI/GUISE-Source>. Accessed: 2019-10-09.
- [142] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2466–2478.
- [143] Yuanfang Ren, Aisharjya Sarkar, Ahmet Ay, Alin Dobra, and Tamer Kahveci. 2019. Finding Conserved Patterns in Multilayer Networks. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. ACM, 97–102.
- [144] Pedro Ribeiro. 2018. gtrieScanner - Quick Discovery of Network Motifs. <http://www.dcc.fc.up.pt/gtries/>. Accessed: 2019-10-09.
- [145] Pedro Ribeiro, David Aparício, Pedro Paredes, and Fernando Silva. 2017. GTScanner - Quick Discovery of Network Motifs. <http://www.dcc.fc.up.pt/~daparicio/software>. Accessed: 2019-10-09.
- [146] Pedro Ribeiro and Fernando Silva. 2010. Efficient subgraph frequency estimation with g-tries. *Algorithms in Bioinformatics* (2010), 238–249.
- [147] Pedro Ribeiro and Fernando Silva. 2010. G-tries: an efficient data structure for discovering network motifs. In *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 1559–1566.
- [148] Pedro Ribeiro and Fernando Silva. 2014. Discovering colored network motifs. In *Complex Networks V*. Springer, 107–118.
- [149] Pedro Ribeiro and Fernando Silva. 2014. G-Tries: a data structure for storing and finding subgraphs. *Data Mining and Knowledge Discovery* 28, 2 (2014), 337–377.
- [150] Pedro Ribeiro, Fernando Silva, and Marcus Kaiser. 2009. Strategies for network motifs discovery. In *2009 Fifth IEEE International Conference on e-Science*. IEEE, 80–87.
- [151] Pedro Ribeiro, Fernando Silva, and Luís Lopes. 2010. Efficient parallel subgraph counting using g-tries. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*. IEEE, 217–226.
- [152] Pedro Ribeiro, Fernando Silva, and Luís Lopes. 2010. A parallel algorithm for counting subgraphs in complex networks. In *International Joint Conference on Biomedical Engineering Systems and Technologies*. Springer, 380–393.
- [153] Pedro Ribeiro, Fernando Silva, and Luís Lopes. 2012. Parallel discovery of network motifs. *J. Parallel and Distrib. Comput.* 72, 2 (2012), 144–154.
- [154] Pedro Ribeiro, Fernando MA Silva, and Luís MB Lopes. 2010. Parallel Calculation of Subgraph Census in Biological Networks.. In *BIOINFORMATICS*. 56–65.
- [155] Stéphane Robin, Etienne Birmelé, Michel Koskas, Gilles Grasseau, and Sophie Schbath. 2018. RAGE - graphlet enumeration algorithm. <https://cran.r-project.org/src/contrib/Archive/NeMo/>. Accessed: 2019-10-09.

- [156] Ryan A Rossi, Nesreen K Ahmed, Aldo Carranza, David Arbour, Anup Rao, Sungchul Kim, and Eunyeek Koh. 2019. Heterogeneous network motifs. *arXiv preprint arXiv:1901.10026* (2019).
- [157] Ryan A Rossi and Rong Zhou. 2016. Leveraging Multiple GPUs and CPUs for Graphlet Counting in Large Networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 1783–1792.
- [158] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. 2017. Estimation of graphlet statistics. *arXiv preprint arXiv:1701.01772* (2017).
- [159] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding Network Motifs Using MCMC Sampling. In *CompleNet*. 13–24.
- [160] Peter Sanders. 1994. A detailed analysis of random polling dynamic load balancing. In *Parallel Architectures, Algorithms and Networks, 1994.(ISPAN), International Symposium on*. IEEE, 382–389.
- [161] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2150–2159.
- [162] Aisharjya Sarkar, Yuanfang Ren, Rasha Elhesha, and Tamer Kahveci. 2018. A new algorithm for counting independent motifs in probabilistic networks. *IEEE/ACM transactions on computational biology and bioinformatics* (2018).
- [163] Thomas Schank and Dorothea Wagner. 2005. Finding, counting and listing all triangles in large graphs, an experimental study. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 606–609.
- [164] Michael Schatz, Elliott Cooper-Balis, and Adam Bazinet. 2008. Parallel network motif finding. *Technical report, University of Maryland Insitute for Advanced Computer Studies* (2008).
- [165] Sophie Schbath, Vincent Lacroix, and Marie-France Sagot. 2008. Assessing the exceptionality of coloured motifs in networks. *EURASIP Journal on Bioinformatics and Systems Biology* 2009, 1 (2008), 616234.
- [166] Benjamin Schiller, Sven Jager, Kay Hamacher, and Thorsten Strufe. 2015. Stream-A stream-based algorithm for counting motifs in dynamic graphs. In *International Conference on Algorithms for Computational Biology*. Springer, 53–67.
- [167] Falk Schreiber and Henning Schwöbbermeyer. 2005. Frequency concepts and pattern detection for the analysis of motifs in networks. In *Transactions on computational systems biology III*. Springer, 89–104.
- [168] C Seshadhri. 2017. Escape (Bitbucket). <https://bitbucket.org/seshadhri/escape>. Accessed: 2019-10-09.
- [169] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. 2013. Triadic measures on graphs: The power of wedge sampling. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 10–18.
- [170] Saeed Shahrivari. 2016. GraphLab PowerGraph implementation of 4-profile counting. <https://github.com/eelenberg/4-profiles>. Accessed: 2019-10-09.
- [171] Saeed Shahrivari and Saeed Jalili. 2015. Distributed discovery of frequent subgraphs of a network using MapReduce. *Computing* 97, 11 (2015), 1101–1120.
- [172] Saeed Shahrivari and Saeed Jalili. 2015. Fast parallel all-subgraph enumeration using multicore machines. *Scientific Programming* 2015 (2015), 6.
- [173] Miguel EP Silva, Pedro Paredes, and Pedro Ribeiro. 2017. Network motifs detection using random networks with prescribed subgraph frequencies. In *International Workshop on Complex Networks*. Springer, 17–29.
- [174] George M Slota and Kamesh Madduri. 2013. Fast approximate subgraph counting and enumeration. In *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 210–219.
- [175] Ricard V Solé and Sergi Valverde. 2007. Spontaneous emergence of modularity in cellular networks. *Journal of The Royal Society Interface* 5, 18 (2007), 129–133.
- [176] Xiaoli Song, Changjun Zhou, Bin Wang, and Qiang Zhang. 2015. A Method of Motif Mining Based on Backtracking and Dynamic Programming. In *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*. Springer, 317–328.
- [177] Olaf Sporns and Rolf Kötter. 2004. Motifs in brain networks. *PLoS biology* 2, 11 (2004), e369.
- [178] Clara Stegehuis, Remco van der Hofstad, and Johan SH van Leeuwen. 2019. Variational principle for scale-free network motifs. *Scientific reports* 9, 1 (2019), 6762.
- [179] Yihan Sun, Joseph Crawford, Jie Tang, and Tijana Milenković. 2015. Simultaneous optimization of both node and edge conservation in network alignment via WAVE. In *International Workshop on Algorithms in Bioinformatics*. Springer, 16–39.
- [180] Ngoc Tam L Tran, Sominder Mohan, Zhuoqing Xu, and Chun-Hsi Huang. 2014. Current innovations and future challenges of network motif detection. *Briefings in bioinformatics* 16, 3 (2014), 497–525.
- [181] Shahadat Uddin and Liaquat Hossain. 2013. Dyad and triad census analysis of crisis communication network. *Social Networking* (2013).
- [182] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.
- [183] Sergi Valverde and Ricard V Solé. 2005. Network motifs in computational graphs: A case study in software architecture. *Physical Review E* 72, 2 (2005), 026107.

- [184] Sebastian Wandelt and Xiaoqian Sun. 2015. Evolution of the international air transportation country network from 2002 to 2013. *Transportation Research Part E: Logistics and Transportation Review* 82 (2015), 55–78.
- [185] Jianxin Wang, Yuannan Huang, Fang-Xiang Wu, and Yi Pan. 2012. Symmetry compression method for discovering network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 9, 6 (2012), 1776–1789.
- [186] Pinghui Wang. 2018. MOSS-5: Fast Method of Approximating Counts of 5-Node Graphlets in Large Graphs. <http://nskeylab.xjtu.edu.cn/dataset/phwang/code/mosscode.zip>. Accessed: 2019-10-09.
- [187] Pinghui Wang, John Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 2 (2014), 8.
- [188] Pinghui Wang, Yiyan Qi, John CS Lui, Don Towsley, Junzhou Zhao, and Jing Tao. 2017. Inferring Higher-Order Structure Statistics of Large Networks From Sampled Edges. *IEEE Transactions on Knowledge and Data Engineering* (2017).
- [189] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2018. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2018), 73–86.
- [190] Tie Wang, Jeffrey W Touchman, Weiye Zhang, Edward B Suh, and Guoliang Xue. 2005. A parallel algorithm for extracting transcriptional regulatory network motifs. In *Bioinformatics and Bioengineering, 2005. BIBE 2005. Fifth IEEE Symposium on*. IEEE, 193–200.
- [191] Stanley Wasserman and Katherine Faust. 1994. *Social network analysis: Methods and applications*. Vol. 8. Cambridge university press.
- [192] Anatol E Wegner. 2014. Subgraph covers: an information-theoretic approach to motif analysis in networks. *Physical Review X* 4, 4 (2014), 041026.
- [193] Sebastian Wernicke. 2005. A faster algorithm for detecting network motifs. In *WABI*, Vol. 3692. Springer, 165–177.
- [194] Sebastian Wernicke. 2006. FANMOD: a tool for fast network motif detection. <http://theinf1.informatik.uni-jena.de/motifs/>. Accessed: 2019-10-09.
- [195] Sebastian Wernicke. 2011. Comment on 'An optimal algorithm for counting networks motifs'. *Physica A: Statistical Mechanics and its Applications* 390 (2011), 143–145.
- [196] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: a tool for fast network motif detection. *Bioinformatics* 22, 9 (2006), 1152–1153.
- [197] Virginia Vassilevska Williams and Ryan Williams. 2013. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.* 42, 3 (2013), 831–854.
- [198] Elisabeth Wong, Brittany Baur, Saad Quader, and Chun-Hsi Huang. 2012. Biological network motif detection: principles and practice. *Briefings in bioinformatics* 13, 2 (2012), 202–215.
- [199] Peng Wu, Junfeng Wang, and Bin Tian. 2018. Software homology detection with software motifs based on function-call graph. *IEEE Access* 6 (2018), 19007–19017.
- [200] Feng Xia, Haoran Wei, Shuo Yu, Da Zhang, and Bo Xu. 2019. A Survey of Measures for Network Motifs. *IEEE Access* 7 (2019), 106576–106587.
- [201] Yuan Xu, Qiang Zhang, and Changjun Zhou. 2014. A new method for motif mining in biological networks. *Evolutionary bioinformatics online* 10 (2014), 155.
- [202] Xifeng Yan and Jiawei Han. 2002. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings*. IEEE, 721–724.
- [203] Chen Yang, Min Lyu, Yongkun Li, Qianqian Zhao, and Yinlong Xu. 2018. SSRW: A Scalable Algorithm for Estimating Graphlet Statistics Based on Random Walk. In *International Conference on Database Systems for Advanced Applications*. Springer, 272–288.
- [204] Esti Yeger-Lotem, Shmuel Sattath, Nadav Kashtan, Shalev Itzkovitz, Ron Milo, Ron Y Pinter, Uri Alon, and Hanah Margalit. 2004. Network motifs in integrated cellular networks of transcription–regulation and protein–protein interaction. *Proceedings of the National Academy of Sciences* 101, 16 (2004), 5934–5939.
- [205] Qiang Zhang and Yuan Xu. 2014. Motif mining based on network space compression. *BioData mining* 8, 1 (2014), 29.
- [206] Zhao Zhao, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. 2010. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 594–603.
- [207] Zhao Zhao, Guanying Wang, Ali R Butt, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. 2012. Sahad: Subgraph analysis in massive networks using hadoop. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 390–401.
- [208] Dongxiao Zhu and Zhaohui S Qin. 2005. Structural comparison of metabolic networks in selected single cell organisms. *BMC bioinformatics* 6, 1 (2005), 1.