An efficient approach for counting occurring induced subgraphs

Luciano Grácio, Pedro Ribeiro

CRACS & INESC-TEC
DCC-FCUP, Universidade do Porto, Portugal
lgracio@fc.up.pt, pribeiro@dcc.fc.up.pt

Abstract. Counting subgraph occurrences is a hard but very important task in complex network analysis, with applications in concepts such as network motifs or graphlet degree distributions. In this paper we present a novel approach for this task that takes advantage of knowing that a large fraction of subgraph types does not appear at all on real world networks. We describe a pattern-growth methodology that is able to iteratively build subgraph patterns that do not contain smaller non-occurring subgraphs, significantly pruning the search space. By using the g-trie data structure, we are able to efficiently only count those subgraphs that we are interested in, reducing the total computation time. The obtained experimental results are very promising allowing us to avoid the computation of up to 99.78% of all possible subgraph patterns. This showcases the potential of this approach and paves the way for reaching previously unattainable subgraph sizes.

Keywords: Subgraph Counting, Pattern-Growth, Induced Subgraphs, Occurring Subgraphs, Network Motifs

1 Introduction

Many real-world systems can be modeled and analyzed using complex networks. Being able to extract information from these networks is therefore a vital task with applications in a multitude of domains [2]. One very important network mining primitive is the ability to count the occurrences of induced subgraphs. The frequency in which different subgraph types appear inside a network provides a rich characterization of its topology and lies at the core of concepts such as *network motifs* [10] and *graphlet degree distributions* [13].

Counting subgraphs in general graphs is however a computationally very hard task. Even just knowing if a subgraph appears at all inside another graph, that is, determining if its frequency is higher than zero, is already an NP-complete problem [3]. This limits the applicability since the computation time grows exponentially on the size of both the subgraphs and the networks containing them.

Current approaches are typically focused on computing the frequency of all possible subgraph types of a given size k. As k increases, the amount of different types increases exponentially. For instance, there are only two different undirected subgraph types of size 2 (a triangle and a chain), but more than 10^5 of

size 8 and more than 10^9 of size 11. However, in real-world networks, many of these subgraphs do not occur at all (*i.e.*, many types have a frequency of zero). In fact, as we show in section 4, as k increases we might have less than 1% of occurring subgraphs.

Present counting methodologies are generally oblivious to the absence of many subgraph types and do not take advantage of this. In this work we propose an approach that fully exploits this characteristic of the subgraphs frequency distribution in order to improve the efficiency and reduce the computation time. We adopt a pattern-growth strategy in which we start by computing the frequency of subgraphs of a smaller initial size k_i. We then iteratively increase this size up to a desired size k by growing only the occurring subgraphs. We do so by making sure that we are only generating subgraph types that do not contain smaller non-occurring subgraphs inside themselves, which greatly reduces the possible subgraph search space. As the basis counting algorithm we use g-tries [14], a data structure that can efficiently store and count any given custom set of subgraphs. This allows the use of information about subgraphs that do not appear, guiding the computation towards subgraphs that do occur and avoiding unnecessary work.

The experimental results obtained with our proof of concept implementation are very promising and show the potential of the approach. We were always able to avoid computing the vast majority of the subgraphs, up to 99.5% in some networks, effectively avoiding the computation time. We also show that as we increase the size k, the fraction of non-occurring subgraphs also keeps increasing and appears to follow a logistic function, which further emphasizes the gains of our methodology and its capability to reach previously unfeasible sizes.

The remainder of this paper is organized as follows. Section 2 discusses background concepts, defines the problem we are solving and gives a brief overview of related work. Section 3 presents our proposed methodology in detail. In Section 4 we provide experimental results, showcasing the gains obtained by our approach. We finally give concluding remarks in Section 5.

2 Background

2.1 Notation

We first review the main graph concepts used, establishing a coherent terminology to be used throughout this paper. A graph G can be defined as a set of nodes (or vertices) $V(G)$, and a set of edges, $E(G)$, each connecting two nodes. An n-graph is a graph of size n, that is, a graph with n nodes. For the sake of simplicity, we will only consider *undirected* graphs, in which edges do not express direction, but our results expand naturally to directed graphs.

Two graphs G and H are *isomorphic* if there is a one-to-one mapping between the nodes of both graphs and there is an edge between two nodes of G if and only if their corresponding vertices in H also form an edge. An *automorphism* is an isomorphism of a graph into itself. The equivalence classes of the nodes under

the action of the automorphisms are called *orbits*, that is, two nodes belong to the same orbit if when mapped to one another, they give origin to isomorphic graphs. Figure 1 exemplifies this concept where, for instance, the star shaped `T4` pattern has two orbits: the white center node and the black periphery nodes.

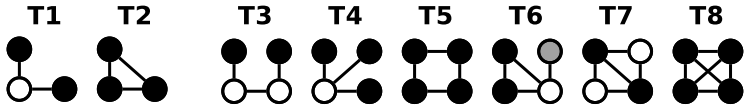

Fig. 1: All possible subgraph types of sizes 3 and 4, and their corresponding orbits (nodes with the same color have the same orbit in the corresponding subgraph).

H is a *subgraph* of G if $V(H) \subset V(G)$ and $E(H) \subset E(G)$. This subgraph is said to be *induced* if $(a, b) \in E(H) \iff (a, b) \in E(G)$. We will only be considering induced subgraphs. Note that the non-induced occurrences are also implicitly counted when finding the induced occurrences. If G contains H as a subgraph, we say that it is a *supergraph* of H. The set of all supergraphs of H is denoted as $super(H)$.

The *subgraph types* of a given size k are the set of all possible connected and non-isomorphic k-graphs. The *frequency* of a subgraph type is the number of times it occurs inside another graph. For the purposes of this paper we consider the classical and most used definition of frequency, which allows overlapping of nodes and edges between different occurrences (for other possible and less used frequency concepts we refer the reader to [16]). The set of all the subgraph types of size k with a frequency higher than zero in a graph G is denoted as k-$O(G)$ (the "O" stands for occurring). For the non-occurring set of k-subgraphs we use k-$NO(G)$. Figure 2 illustrates these concepts, showing the occurrences of all subgraphs of sizes 3 and 4. For instance, the frequency of subgraph type `T1` is 4. Note how supergraphs do not necessarily have a smaller or equal frequency than its subgraphs. For example, the single occurrence of the triangle `T2` induces two occurrences of the pattern `T6`, which contains inside it the triangle. This has implications on the computation tractability, as there is no downward closure property on the frequencies, and separates our problem from a more classical frequent subgraph mining task [8].

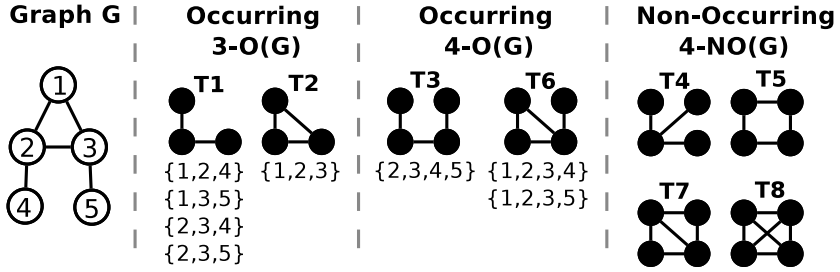

Fig. 2: Example graph and its corresponding 3 and 4-subgraph occurrences.

2.2 Problem Definition

Definition 1 (The occurring induced subgraph counting problem). Given a graph G and an integer k, determine the exact frequencies of the subgraphs in k-$O(G)$, that is, count all induced occurrences of k-subgraphs that occur at least once in G. Two occurrences are considered different if they have at least one node or edge that they do not share. Other nodes and edges can overlap.

Note that k-$NO(G)$ is just the complement of k-$O(G)$ and is also implicitly calculated by solving this problem. Furthermore, for most of the applications the interest is in the positive frequencies. For instance, network motifs can be thought of as overrepresented subgraphs [10], and the first step towards its discovery implies computing the frequencies of subgraphs that do appear on the given network.

2.3 Related Work

Current approaches for exact counting of general subgraphs typically follow one of three different strategies. Some methods are *network centric*, in the sense that given a number k, they compute the frequency of all possible subgraphs of size k. They generally work by enumerating all connected sets of k nodes and by identifying the isomorphic class of each occurrence. Examples of this are `ESU` [19] and `FaSE` [11]. They differ from our work because they do not explicitly take into advantage information about subgraphs that are certain to not exist. Other methods are *subgraph-centric*, meaning that they only count occurrences of a single individual subgraph type. Examples of this are `Grochow` [5] and `ISMAGS` [7]. In principle they could be coupled with our approach as counting algorithms. However, they do not use information from previously computed frequencies. Finally, *set-centric* methods lie in-between the two previous described approaches. They allow for counting custom sets of subgraphs, larger than a single subgraph, but also smaller than all possible k-subgraphs. The best-known example for this approach is the g-trie data structure [14] and the associated counting algorithms. We leverage precisely this capability by integrating it in our workflow as described in Section 3 and we add the capability of only searching for subgraphs that do not contain non-occurring smaller subgraphs.

Besides these general and exact approaches there are a multitude of algorithms. Some are geared towards specific types of subgraphs (for example star shaped subgraphs [4] or undirected subgraphs up to size five [6]). Others, such as `MOSS` [18] or `Rand-FaSE` [12], trade accuracy for speed and provide only approximate results.

We should also mention that several works exploit parallelism to further improve the computation time. Here we are more focused on the core methodological approach, and our proof of concept implementation is sequential, but our work can be adapted to use parallelism. In fact, there is an almost direct way of benefiting from this since there already exists distributed-memory [15] and shared-memory [1] parallel implementations of g-tries that we use as our base counting algorithm.

3 Proposed Methodology

3.1 Overview

We propose an incremental approach as detailed in the following algorithm:

Algorithm 1: High-level overview of our proposed methodology

Input : A graph G and an integer $k \geq 4$
Output: The exact frequencies of all subgraphs in k-$O(G)$

```
1  C_3 ← all 3-subgraph types            // current candidate subgraphs
2  B ← ∅                                 // blacklisted subgraphs
3  for i ← 3 to k do
4  |  F_i ← frequencies of all i-subgraphs ∈ C_i in graph G
5  |  if i < k then
6  |  |  B ← B ∪ all i-subgraphs ∈ C_i with frequency = 0
7  |  |  C_{i+1} ← (i+1)-super(C_i) that don not have any subgraph in B
8  |  end
9  end
10 return F_k
```

We assume that $k \geq 4$ since 2-subgraphs are just edges and computing 3-subgraph frequencies would be equivalent to simply counting without the opportunity to use a blacklist to limit the search. Note also that since we are considering simple undirected subgraphs, the initial candidate list C_3 consists on exactly two possible subgraphs: a triangle and a chain, corresponding to patterns `T1` and `T2` in Figure 1. In the following sections we give more detail on how to achieve each of the steps described by this algorithm and how the whole process results in an efficient computation of the occurring k-subgraphs.

3.2 Counting sets of subgraphs

The entire approach relies on keeping a list of current candidate subgraphs C_i which should not include subgraphs that are sure to not occur, hence pruning the search space. We then naturally need the ability to compute the frequency of all subgraphs in this candidate list (line 4 in Algorithm 1), both for adding more non-occurring subgraphs to our blacklist, and for generating the next set of candidates with one more node.

Computing these frequencies efficiently is not a trivial task. Taking advantage of the mutual information between the candidates is essential, even more since we are ruling out non-occurring subgraphs, which also highlights the topological characteristics that the occurring subgraphs should not have. With that purpose in mind we use the g-tries, previously developed by us, which are efficient data structures for storing and finding sets of subgraphs. In the same way that a prefix-tree can efficiently store strings by storing only once common prefixes between them, a g-trie takes advantage of the similarities between subgraphs and represents them with the aim to minimize redundancy. Figure 3 exemplifies the concept. Note how descendants of a g-trie node share the same subgraph.

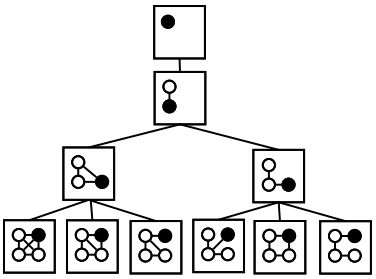

Fig. 3: An example g-trie containing all undirected 3 and 4-subgraphs.

After inserting the candidates (C_i) in a g-trie, we perform a census operation that computes their frequencies in G. Given the space constraints, we refer the reader to [14] for an in-depth explanation on how g-tries do this very efficiently, including how the g-trie itself is built given a set of subgraphs and how the usage of symmetry breaking conditions are able to constrain the search.

3.3 Blacklisting subgraphs

In the process of counting occurrences of a set of subgraphs, not only do we gather information about the subgraphs that occur in G but also about the ones that are non-occurring. This section describes how we use that information. The two main properties we exploit are the following:

1. **If a subgraph G' does not occur in G, no element of super(G') occurs in G.** When we find a non-existing subgraph $G' \in NO(G)$, we can also rule out the possibility that any subgraph that contains G' exists in G.
2. **Any (sub)subgraph of a (sub)graph contained in the blacklist has a positive frequency.** If they were non-occurring, then they would have been put into the blacklist during previous iterations, break property (1). This means that the blacklist is in its essence *minimal*, storing the smallest possible subgraphs that do not occur in G.

We now describe how to implement and use the blacklist.

Creation Once we have computed all i-subgraphs $\in C_i$ which do not occur in G, we insert them into a new g-trie, which we append to the blacklist B, currently implemented as an array of g-tries (line 6 of Algorithm 1).

Usage The blacklist B is used every time we consider adding a new candidate subgraph S (line 7 of Algorithm 1). This process is quite similar to the occurrence counting solution described above, but in this case we are searching for the blacklisted subgraphs in S and the process is interrupted once any of the

subgraphs is found. We iterate through all the g-tries in B, from the smallest to the largest size. If we discover an occurrence in S of any of the subgraphs in any of the g-tries, the process is interrupted and we know that we do not need to append S to the next iteration's candidates C_{i+}. We denote this as *is_blacklisted*(S). Keep in mind that there is also no need to append S to B, since it already contains a subgraph that is blacklisted. In fact, the lack of need to keep these discarded candidates in memory is one of the main advantages of this approach. In the case that no occurrences were found in any of the g-tries, we append the candidate to C_{i+1}.

3.4 Candidate generation

In the candidate generation phase of the iterative process, C_{i+1} is created (line 7 of Algorithm 1). In order to do this we start by considering a limited set of candidate subgraphs, by generating *super*(i-$O(G)$), that is, the set of graphs of size $i+1$ that contains at least one subgraph of an occurring subgraph of C_i. Note that we still need to pass the generated subgraphs through the blacklist filter, since it is most likely that some will contain blacklisted subgraphs. For instance, consider the subgraph types exemplified in Figure 1 and imagine a situation where the chain `T1` occurs and the triangle `T2` does not (and hence is included in the blacklist). The subgraph type `T6` would be generated as an expansion of `T1`, but would still be filtered out by the blacklist, since it contains `T2`.

We start by presenting a naive solution to this problem of supergraph generation, in which given an i-subgraph H, we want to generate all the $(i+1)$-subgraphs in *super*(H).

Exhaustive Node Enumeration: we add a new node and try to connect it to all possible subsets of the existing i nodes. This results in $2^i - 1$ possible connected $(i+1)$-subgraphs (the -1 term comes from excluding an empty set of current nodes, which would result in an unconnected new node).

The main problem with this exponential solution is that many subgraphs of the same type are redundantly created. In fact, connecting the new node to any two subsets that exhibit the same orbit classes will result in isomorphic subgraphs. Recall that the concept of orbits is exemplified in Figure 1. For instance, connecting a new node to any single node in the triangle `T1` (with only one orbit), will always result in subgraph `T6`. To address this problem, we developed a smarter method for generating supergraphs, that we now describe.

Orbit-Aware Enumeration: we only connect the new node to all possible orbit combinations. For instance, if a 4-subgraph has two nodes in orbit a and two nodes in orbit b (an example of this is subgraph `T7`), instead of naively trying all possible $2^4 - 1 = 15$ sets, we only connect to 8 subsets of existing orbits: $\{a\}, \{a, a\}, \{b\}, \{b, a\}, \{b, a, a\}, \{b, b\}, \{b, b, a\}$ and $\{b, b, b, a\}$. In order to implement this strategy we use a very efficient third-party software (nauty [9]) to compute the automorphisms and orbit classes.

While our orbit-aware strategy avoids duplicates growing from the same subgraph, it might still happen that duplicate $(i+1)$-subgraphs emerge from different $i-$subgraphs, due to the effect already described when considering the usage of the blacklist. For instance, consider Figure 1 and subgraph type `T6`, which would be generated as a supergraph of `T1`, but also as a supergraph of `T2`. To avoid having these duplicates and only keep one copy of each isomorphic subgraph class, we insert all candidates in a g-trie, which efficiently checks if the subgraph was already inserted.

4 Experimental Results

All tests were executed in a laptop with an Intel i7-6700HQ CPU, and 16GB of RAM. We used three different undirected real-world networks with varied topological characteristics, as described in Table 1.

Name	Size	Nr. Edges	Description
`karate`	34	78	social ties between members of a karate club [20]
`circuit`	252	399	electronic circuit (s_{420}) [10]
`euroroad`	1174	1417	europe main roads network [17]

Table 1: Real-world networks used in our experiments.

In order to test and explore our methodology, we ran a thorough experiment in which we kept increasing the subgraph size k being computed and we took note of the following values:

- **occurring:** number of occurring subgraphs of size k in G;
- **blacklisted:** number of subgraphs kept in the blacklist B after that iteration;
- **blocked:** number of subgraphs types that were generated but were not inserted into the candidates because they contained blacklisted subgraphs;
- **all:** number of all possible undirected subgraphs types of size k;
- **avoided:** percentage of **all** whose generation was avoided by our solution;
- **time:** execution time, measured as elapsed time between launching the program up to computing the k-frequencies, including everything in-between.

We now present three tables with the obtained results:

size k	occurring	blacklisted	blocked	all	avoided	time (s)
4	6	0	0	6	0.00%	0.00
5	21	0	0	21	0.00%	0.00
6	89	23	0	112	0.00%	0.03
7	476	81	297	853	2.22%	0.14
8	2612	832	6821	11117	10.34%	1.73
9	11569	3272	153286	261080	37.28%	33.28
10	40069	7886	2348283	11716571	79.89%	405.25

Table 2: Results obtained for the `karate` network.

size k	occurring	blacklisted	blocked	all	avoided	time (s)
4	5	1	0	6	0.00%	0.00
5	11	7	3	21	3.70%	0.00
6	33	14	56	112	14.39%	0.02
7	89	37	486	853	33.37%	0.05
8	293	61	3891	11117	63.80%	0.20
9	1001	142	31123	261080	88.03%	1.20
10	3659	379	259851	11716571	97.79%	8.44
11	13462	1203	2174907	1006700565	99.78%	66.77

Table 3: Results obtained for the `circuit` network.

size k	occurring	blacklisted	blocked	all	avoided	time (s)
4	5	1	0	6	0.00%	0.00
5	13	5	3	21	3.70%	0.00
6	41	15	51	112	10.07%	0.02
7	137	38	501	853	25.90%	0.06
8	511	105	4830	11117	53.41%	0.33
9	1937	299	47290	261080	81.61%	2.42
10	7428	909	453838	11716571	96.12%	19.23
11	28190	2521	4143557	1006700565	99.59%	175.27

Table 4: Results obtained for the `euroroad` network.

The results are very promising. The first main aspect to notice is that in all networks we are able to avoid computing the frequency of the vast majority of the entire set of all possible k-subgraph types (column *avoided*), with values reaching up to 99.78% with $k = 11$ in the `euroroad` network. Although the exact percentage may vary, it seems to follow a logistic distribution in the tested networks, as shown in Figure 4. This provides further evidence that our strategy has the potential to avoid most of the computation by making use of the information of smaller sized subgraphs. Furthermore, we do this while being able to provide exact results, assuring accuracy and always knowing all subgraph types that occur at least once.

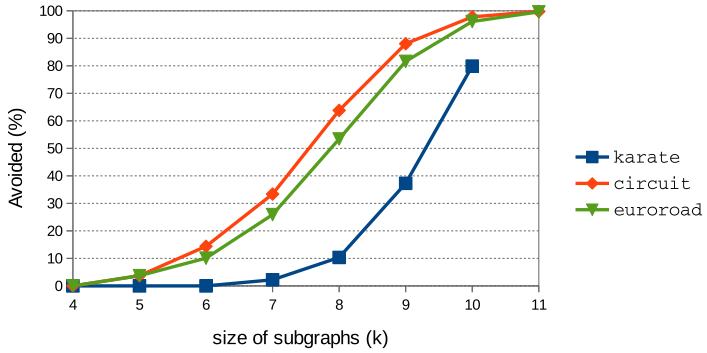

Fig. 4: Fraction of avoided subgraphs with increasing size k.

By looking at the blacklist size (column *blacklisted*), we can have a better notion on how we are filtering out non-occurring subgraphs. Gains are more immediate and larger when smaller subgraphs already have zero frequency. For instance, the clique of size 4 - type `T8` in Figure 1 - does not occur in the `euroroad` network. As a result we blacklist it in the first iteration, avoiding the computation of all its supergraphs. However, even in the cases where we start blacklisting later (for instance, the `karate` network contains all possible size 4 and 5 subgraph types and the blacklist only contains subgraphs of size ≥ 6), the quantity of blacklisted nodes still steadily grows and the quantity of avoided subgraphs seems to follow a similar shape. This gives empirical evidence that it might take longer, but as soon as we discover subgraphs that do not occur, they quickly give origin to many other related non-occurring subgraph types. Further preliminary experiments with other networks also support this observation. We should also indicate the really reduced size of the blacklist when compared with all possible subgraphs, showcasing our capability of having a very compact way of representing patterns known to not occur.

Another aspect to mention is that the number of subgraph types that appear at least once (column *occuring*) is always a small fraction of all possible types. Furthermore, this fraction of occurring subgraphs keeps shrinking as k increases, which indicates it might scale to larger values of k.

In what concerns execution time, even with just our preliminary non-optimized implementation we are already competitive up to size 9 when compared with the set-centric and very efficient g-trie base method starting with a g-trie containing all possible k-subgraphs (we achieve slightly slower times but still on the same order of magnitude). For sizes 10 and 11, this base approach is not even applicable, as the number of all possible subgraphs is so large that it is impossible to store them all on a g-trie. This showcases the need for a methodology like the one we propose.

As a final remark we would like to comment on the potential of our approach. Previous approaches that rely on knowing beforehand all possible subgraph types

are severely limited in time and memory, as the number of possible subgraph types grows superexponentially (column *all*). This is the case for general set-centric or subgraph-centric algorithms when we do not have a way of filtering subgraphs. It is also the case for network-centric methods, which typically provide a large memory footprint that might even grow exponentially as k increases (an example of this is `FaSE` [11]). Even less memory hungry analytic approaches might rely on knowing all types, as in the case of `ORCA` [6], which builds an intricate system of equations that relate all different subgraph types to accelerate the computation and cannot directly compute only a small fraction of these. Our approach provides a more efficient way of exploring the search space and can be combined with any base approach takes advantage of knowing which subgraph types we are looking for. Furthermore, the memory requirements are only bounded by the size of the occurring and blacklisted subgraphs, greatly enhancing the limits of applicability to much larger sizes.

5 Conclusion

Being able to count the frequency of subgraphs is a fundamental graph mining task with many applications. In this paper we present a novel approach for computing the frequency of all k-subgraphs that appear at least once. We propose a pattern-growth approach that is able to prune the search space. For this purpose, we keep a minimal list of subgraphs that we know that do not occur and we use it as a blacklist to filter out non-occurring subgraphs. With this, we are able to iteratively keep increasing the subgraph size with a very compact list of viable candidate subgraphs. The results obtained are very promising and we are able to avoid the computation of up to 99.78% of all possible subgraphs. The experiments showcase the capability of our approach and indicate that we can contribute towards increasing the feasible subgraph sizes.

For future work, we plan to further optimize our implementation and provide the community with an open-source tool. Furthermore, we intend to make a more systematic and thorough experimentation on a large set of both synthetic and real-world networks, gaining more insight on the distributions of occurring and blacklisted subgraphs, and also on the avoided fraction of subgraphs. We would also like to better explore the candidate generation algorithm and to explore other pruning options besides only avoiding non-occurring subgraphs.

Acknowledgements

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013.

References

1. Aparício, D.O., Ribeiro, P.M.P., da Silva, F.M.A.: Parallel subgraph counting for multicore architectures. In: Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on. pp. 34–41. IEEE (2014)
2. Costa, L.d.F., Oliveira Jr, O.N., Travieso, G., Rodrigues, F.A., Villas Boas, P.R., Antiqueira, L., Viana, M.P., Correa Rocha, L.E.: Analyzing and modeling real-world phenomena with complex networks: a survey of applications. Advances in Physics 60(3), 329–412 (2011)
3. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. In: Graph Algorithms And Applications I, pp. 283–309. World Scientific (2002)
4. Gonen, M., Ron, D., Shavitt, Y.: Counting stars and other small subgraphs in sublinear-time. SIAM Journal on Discrete Mathematics 25(3), 1365–1411 (2011)
5. Grochow, J.A., Kellis, M.: Network motif discovery using subgraph enumeration and symmetry-breaking. In: Annual International Conference on Research in Computational Molecular Biology. pp. 92–106. Springer (2007)
6. Hočevar, T., Demšar, J.: A combinatorial approach to graphlet counting. Bioinformatics 30(4), 559–565 (2014)
7. Houbraken, M., Demeyer, S., Michoel, T., Audenaert, P., Colle, D., Pickavet, M.: The index-based subgraph matching algorithm with general symmetries (ismags): exploiting symmetry for faster subgraph enumeration. PloS one 9(5), e97896 (2014)
8. Jiang, C., Coenen, F., Zito, M.: A survey of frequent subgraph mining algorithms. The Knowledge Engineering Review 28(1), 75–105 (2013)
9. McKay, B.D., Piperno, A.: Practical graph isomorphism, ii. Journal of Symbolic Computation 60, 94–112 (2014)
10. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: simple building blocks of complex networks. Science 298(5594), 824–827 (2002)
11. Paredes, P., Ribeiro, P.: Towards a faster network-centric subgraph census. In: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. pp. 264–271. ACM (2013)
12. Paredes, P., Ribeiro, P.: Rand-fase: fast approximate subgraph census. Social Network Analysis and Mining 5(1), 17 (2015)
13. Pržulj, N.: Biological network comparison using graphlet degree distribution. Bioinformatics 23(2), e177–e183 (2007)
14. Ribeiro, P., Silva, F.: G-tries: a data structure for storing and finding subgraphs. Data Mining and Knowledge Discovery 28(2), 337–377 (2014)
15. Ribeiro, P., Silva, F., Lopes, L.: Efficient parallel subgraph counting using g-tries. In: 2010 IEEE Int. Conference on Cluster Computing. pp. 217–226. IEEE (2010)
16. Schreiber, F., Schwöbbermeyer, H.: Frequency concepts and pattern detection for the analysis of motifs in networks. In: Transactions on computational systems biology III, pp. 89–104. Springer (2005)
17. Šubelj, L., Bajec, M.: Robust network community detection using balanced propagation. The European Physical Journal B 81(3), 353–362 (2011)
18. Wang, P., Zhao, J., Zhang, X., Li, Z., Cheng, J., Lui, J.C., Towsley, D., Tao, J., Guan, X.: Moss-5: A fast method of approximating counts of 5-node graphlets in large graphs. IEEE Tran. on Knowledge and Data Engineering 30(1), 73–86 (2018)
19. Wernicke, S.: Efficient detection of network motifs. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 3(4), 347–359 (2006)
20. Zachary, W.W.: An information flow model for conflict and fission in small groups. Journal of anthropological research 33(4), 452–473 (1977)