GRAFT: An Approximate Graphlet Counting Algorithm for Large Graph Analysis *

Mahmudur Rahman, Mansurul Bhuiyan, and Mohammad Al Hasan
Dept. of Computer and Info. Science, Indiana University–Purdue University, Indianapolis, USA
mmrahman@iupui.edu, mbhuiyan@iupui.edu, alhasan@cs.iupui.edu

ABSTRACT

Graphlet frequency distribution (GFD) is an analysis tool for understanding the variance of local structure in a graph. Many recent works use GFD for comparing, and characterizing real-life networks. However, the main bottleneck for graph analysis using GFD is the excessive computation cost for obtaining the frequency of each of the graphlets in a large network. To overcome this, we propose a simple, yet powerful algorithm, called GRAFT, that obtains the approximate graphlet frequency for all graphlets that have upto 5 vertices. Comparing to an exact counting algorithm, our algorithm achieves a speedup factor between 10 and 100 for a negligible counting error, which is, on average, less than 5%; For example, exact graphlet counting for ca-AstroPh takes approximately 3 days; but, GRAFT runs for 45 minutes to perform the same task with a counting accuracy of 95.6%.

Categories and Subject Descriptors

I [**Computing Methodologies**]: Miscellaneous; J [**Computer Applications**]: Miscellaneous

Keywords

approximate graphlet counting, graph analysis, graphlet frequency distribution

1. INTRODUCTION

Structural analysis of networks is an important research task that has received the due attention by researchers in various disciplines, such as social sciences [2], system sciences [4], and bioinformatics [8]. Such analyses lead to the discovery of various non-random properties in large, real-life networks; examples include scale-free-ness [1], small diameter [10], and graph densification with shrinking diameter [6]. Various graph generation models are also discovered for generating synthetic graphs having properties alike to the real-life graphs. Though these discoveries enhance our understanding of network structure in general, they are hardly useful in solving practical problems related to graphs and networks. The reason behind this is that the properties discussed in the above existing works are agnostic to the details and effects that arise when dealing with real networks. Further, they exhibit the global behaviors of a network, whereas the detailed sketch of the local structure around a node (or an edge) is required when answering questions that are too specific to that node (or edge). Existing network models are unable to provide such detailed information.

In a large network, a sketch of the local structure can be obtained by collecting the topological context in which each of the nodes resides. N. Przulj's group [8, 7] has done some interesting research works along this direction; in these works, the authors find the position of each vertex in a graphlet-space [1], which they use for solving various tasks that are related to biological networks. Some of these tasks are: compare structures of different biological networks [8], characterize biological networks using graphlet degree distribution [8], and obtain a structural to functional mapping for biological networks [7]. All these works require counting the graphlet frequencies, which is computationally expensive. There exists a software, called GraphCrunch [5], which counts the frequencies of all graphlets that have upto five vertices; however, we find that it is practically infeasible to use this software for counting graphlets in large networks that are available in the domains of social and information networks. For example, we ran GraphCrunch on Enronemail data set (38,692 vertices, 367,664 edges) and slashdot data set (77,357 vertices, 516,675 edges); neither of the counting processes finish after 5 days of running on a typical desktop computer.

An alternative to exact graphlet counting is to adopt algorithms for approximate counting that offer significant speedup with a negligible counting error; this direction has become popular in some of the recent researches for counting triangles [9, 3]. Since, the cost of graphlet counting is much higher than the cost of triangle counting, an approximate counting algorithm for the former will be more useful from a practical standpoint. Also for many practical usages of graphlets, such as for the construction of graphlet frequency distribution (GFD), approximate graphlet counting can be used in places of exact counting without any visible loss; although counting errors prevails by adopting an approximate counting, the effect of this error on GFD is negligible,

*This research is supported by an NSF CAREER Award (IIS-1149851)

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

[1] a graphlet is a small connected non-isomorphic induced subgraph of the given network. We provide a formal definition of graphlet in subsequent section.

because the latter compares the relative counts of various graphlets in a logarithm scale. Unfortunately, no algorithm exists that performs approximate counting of graphlets.

In this research, we propose a method called GRAFT [2] to perform the task of approximate counting of graphlets that have upto five vertices; in Figure 1 we show all such graphlets (modulo isomorphism). GRAFT samples a small number of edges uniformly and for each of the sampled edges it obtains a partial count for a graphlet such that the graphlet uses that edge in one of its induced embedding; GRAFT then uses the partial count associated with the sampled edges for approximating the total count of that graphlet. Experiments show that by sampling between 5% and 10% of the edges, we can easily obtain more than 95% of accuracy in graphlet counting for a speedup factor between 20 and 10; for larger graphs, the sampling factor can be reduced to 1% (or less) to achieve similar accuracy and even higher speedup.

2. BACKGROUND

Assume, $G(V, E)$ is a graph, then V is the set of vertices and E is the set of edges. Each edge $e \in E$ can be represented by a pair of vertices (v_i, v_j) where, $v_i, v_j \in V$. A graph is called simple, if it does not contain a self loop, and at most one edge exists between two of its vertices. In this work, we consider simple, connected, and undirected graphs.

A graph $G' = (V', E')$ is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$. A graph $G' = (V', E')$ is a **vertex-induced subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$ and $\{e = (v_a, v_b) : v_a, v_b \in V', e \in E, e \notin E'\} = \phi$. A vertex-induced subgraph is a subset of the vertices of a graph G together with any edges whose both endpoints are in this subset. In this paper we will refer to vertex-induced subgraph as "induced subgraph". Two graphs G and G' are **isomorphic**, denoted by $G \cong G'$, if there exists a structure-preserving (both adjacency and non-adjacency preserving) bijection $f : V \to V'$; such a function f is called an isomorphism from G to G'. An **embedding** of a graph G' in another graph G is a subgraph S of G, such that S and G' are isomorphic; when the subgraph S is a vertex-induced subgraph of G, the embedding is called an **induced embedding**. In Figure 1, g_5 is a subgraph, but not an induced subgraph, of g_{19}; On the other hand, g_7 is an induced subgraph of g_{19}.

Graphlets can be defined as small, non-isomorphic, connected, induced subgraphs of a large network. In this study, we work with all possible graphlets having k vertices; where, $k \in \{3, 4, 5\}$. We refer a graphlet with k vertices, as *k-Graphlet*; Note that, 1-*Graphlet* is simply a vertex, and 2-*Graphlet* is an edge. A *k-Graphlet* is called a **tree graphlet** if it is a tree, i.e., it has $k - 1$ edges. A graphlet that is not a tree graphlet is called a **cyclic graphlet**. Figure 1 shows all the graphlets that have between 3 and 5 vertices; there are 29 graphlets in this set. They are referred as g_i, for i from 1 to 29. Among them, $g_1, g_3, g_4, g_9, g_{10}$, and g_{11} are tree graphlets, and the remaining are cyclic graphlets. For each graphlet, we identify each of its vertices by an English small letter, such as, a, b, c, etc. as shown in Figure 1.

The task of **Graphlets counting** over an input graph G is to find the counts of all distinct induced embedding of each of the graphlets having upto k vertices. To distinguish an embedding, we assign integer identifiers to each of the vertices in G, starting from 1 to $|V|$. Then an induced

[2]GRAFT is an anagram of the bold letters in **A**pproximate **GR**aphlet **F**requency coun**T**ing

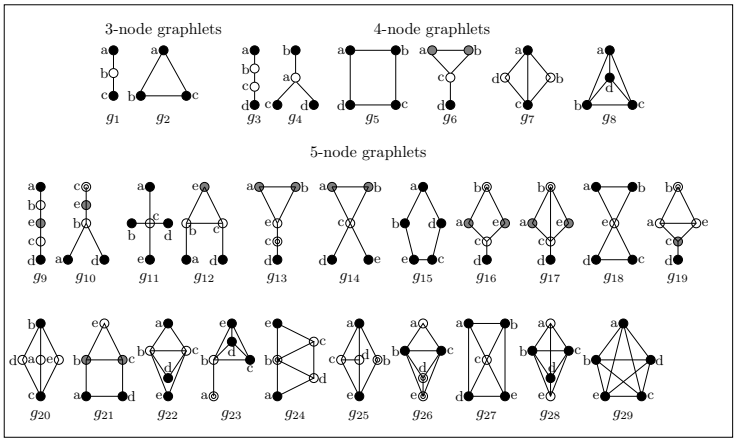

Figure 1: All 3,4,5-node graphlets

embedding of a *k-Graphlet* is denoted simply by a set of k vertex identifiers in the graph G, such that the subgraph induced by those vertices is isomorphic to that graphlet. Due to the "induced" constraint, at most one *k-Graphlet* is embedded in a given set of vertices of size k. In Figure 1, the induced-subgraph consisting with the vertices $\{a, b, c, e\}$ of g_{19} is an induced embedding of g_7; no other graphlet of size 4 is embedded in the above set of vertices of g_{19}.

An isomorphism from a graph $G(V, E)$ to itself is called an **automorphism**. Thus, an automorphism π of a graph G is a structure-preserving permutation π_V on V along with a consistent permutation π_E on E. The total number of automorphism of a graph is defined as $|\mathbf{Aut}(G)|$. Also, any permutation can be represented as a product of disjoint cycles, the vertices that belong to the same cycle under an automorphism form an equivalence class, which is called a vertex-orbit. Similarly the equivalence classes of the edges are called edge-orbits. In Figure 1, each vertex-orbit of a graphlet is represented by drawing the vertices of the orbit by same color. For example, graphlet g_{14}'s automorphism count is 4; it has three vertex-orbits $(a, b), (c), (d, e)$ and three edge-orbits $(ab), (ac, bc), (cd, ce)$.

3. METHOD

GRAFT works as an EDGEITERATOR algorithm; in such an algorithm, the counting process iterates over the edges of the input graph, $G(V, E)$. For an edge $e \in E$, it finds the count of all induced embeddings of a graphlet g with the constraint that the edge e is part of the embedding; we call this count a *partial count* of the graphlet with respect to the edge e. The partial count can be summed over all the edges to obtain a total count of the graphlet g in the input graph. However, in the above process, a distinct graphlet will be counted multiple times, by being accounted in different partial counts, so the above count needs to be corrected by dividing it with an appropriate normalization factor. Such a method yields an exact graphlet counting algorithm. GRAFT obtains an approximate graphlet counting algorithm by iterating over a random subset of edges instead of all the edges of the input graph. The fraction of edges in the random subset with respect to all the edges is called the *sampling factor* of GRAFT. The lower the sampling factor, the faster GRAFT runs. On the other hand, the higher the sampling factor, the better the accuracy of GRAFT. For a sampling factor of 1, GRAFT returns an exact count. In Figure 2, we show a pseudo-code of GRAFT.

GRAFT($G(V,E), g, p$):
comment: G is the large network,
g is a Graphlet
p is the sampling factor
1. choose an specific edge e_g in g
2. $count = 0$
3. Ep = sample p fraction of edges from E (without replacement)
4. **for** each edge $e \in Ep$
5. align e_g with e
6. enumerate all induced embeddings of g in G, where e and e_g are aligned, x is the total number of embeddings found
7. $count = count + x$
8. $count = count/normalization\_factor$
9. $count = count/p$
10. **return** $count$

Figure 2: GRAFT Algorithm

3.1 Partial graphlet count

We first discuss, how to obtain the partial count of a graphlet g that is associated with an edge e of the input graph (Line 5-6 in Figure 2). The first step for this task is to choose an specific edge e_g in the graphlet g, which will be aligned with the edge e in the large graph G. We will call the edge e_g the *first aligned edge* (FAE). Though, the choice of FAE can be arbitrary for exact counting, it is not the same for approximate counting; for the latter, a poor choice can drop the counting accuracy significantly. We will discuss more on this in Section 4.

Once the FAE is chosen, the next task is to enumerate all the embeddings of a graphlet g with the constraint that in those embeddings, e_g is aligned with the edge e (Line 6 in Figure 2). The size of the set containing all the embeddings is the partial count of the graphlet g associated with the edge e. The enumeration process of the embeddings differs based on whether g is a tree graphlet or a cyclic graphlet.

3.1.1 Tree graphlet enumeration

Enumeration process is simpler for a tree graphlet. From Figure 1, there are one (g_1), two (g_3, g_4), and three (g_9, g_{10} and g_{11}) tree graphlets with 3, 4, and 5 vertices, respectively.

We first explain the enumeration of g_3. Suppose, we are given the graph G shown in Figure 3(a) and we want to enumerate the embedding of graphlet g_3 in G. For this, GRAFT chooses the edge (b, c) of graphlet g_3 as FAE, and aligns it with the edge (id_1, id_2) of graph G (Figure 3(b)). Then step by step, it embeds the graphlet g_3 on the graph G over vertices $\{id_1, id_2, id_3$ and $id_4\}$ (From Figure 3(b) to 3(d)). At the end, we get an induced embedding of g_3 consisting with the vertices $\{id_1, id_2, id_3$ and $id_4\}$ (Figure 3(d)). By iterating over the adjacency lists of id_1 and id_2, GRAFT enumerates all possible embeddings of a and d. Note that, GRAFT only enumerates (counts) the induced embedding for g_3. For example, the embedding for g_3 consisting with the vertices $\{id_1, id_2, id_3$ and $id_5\}$ (Figure 3(e)) is not an induced embedding, and GRAFT will not enumerate it while counting the graphlet g_3.

It is easy to see that $|\mathbf{Aut}(g_3)|$ is 2, as a chain (g_3 graphlet) can be embedded at most in two ways (forward and backward) over the same set of vertices in G; so, the normalization factor for g_3 is 2. However, by applying the constraint that the edge (b, c) is mapped to edge (id_i, id_j), only if $id_i < id_j$, we can ensure that g_3 is mapped to an embedding only once, and then the normalization factor for this enumeration becomes 1.

Another example of tree graphlet enumeration, may be, the enumeration of g_{11}. Initially, GRAFT embeds the edge (a, c) of this graphlet with an edge (id_1, id_2) of the large graph (Figure 4). Then, it scans the adjacency list of vertex id_2 to find all possible mappings of vertices b, d and e of g_{11} to vertices id_4, id_5 and id_6 of the large graph. Thus, we get an embedding (not necessarily induced) of g_{11} in the large graph. Finally, GRAFT checks whether the embedding is induced or not. It only enumerates (counts) the induced embedding for g_{11}. Now, by applying the constraint that, mapping of vertices b, d and e of g_{11} to vertices id_4, id_5 and id_6 of the large graph is valid if and only if $id_4 < id_5 < id_6$, we will have to deal with only 4 repetitions of an embedding. Therefore, the normalization factor for g_{11} is 4. Other tree graphlets (g_1, g_4, g_9 and g_{10}) can also be enumerated by following a similar mechanism.

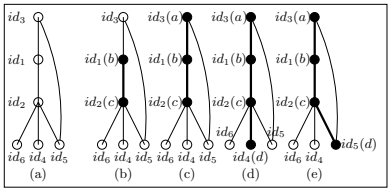

Figure 3: Embedding tree graphlets g_3

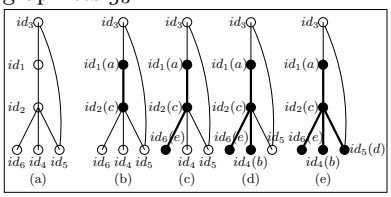

Figure 4: Embedding tree graphlets g_{11}

3.1.2 Cyclic graphlet enumeration

For enumerating an embedding of a cyclic graphlet g, we can use one of the spanning trees of g; the specific spanning tree that is used for the generation is called a **generation tree graphlet**. A tree graphlet is it's own generation tree graphlet. Multiple graphlets can have the same generation tree graphlet (e.g., g_5 and g_6 both have g_3 as their generation tree graphlet). Also, for a cyclic graphlet, there can be multiple generation tree graphlets (e.g., g_{22} can have g_9, g_{10} and g_{11} as its generation tree graphlet).

To enumerate a cyclic graphlet, GRAFT initially embeds the generation tree graphlet (which is not induced) as we explain in Section 3.1.1. Then it checks explicitly whether the desired graphlet is induced in the embedding of it's generation tree graphlet.

For example, the generation tree graphlet of g_5 is g_3. So, in order to embed the graphlet g_5 in a large graph, g_3 must be embedded at the beginning. Figure 3(e) shows an embedding (not induced) of g_3 (consisting with the vertices $\{id_1, id_2, id_3$ and $id_5\}$), which gives us the induced embedding of g_5 since the edge (a, d) is induced by the existence of the edge (id_3, id_5), and no other edge (excluding the edges of tree graphlet) exists between a pair of vertices from the set $\{id_1, id_2, id_3, id_5\}$. The similar concept of normalization as Section 3.1.1 applies here. If we apply the restriction to induce the edge (b, c) to edge (id_i, id_j) when the condition $id_i < idj$ satisfies, then we will have 4 duplications (using 4 left-rotations to align the edge (b, c) to different edges of the rectangle) of an embedding ($normalization\_factor$ is equal to 4).

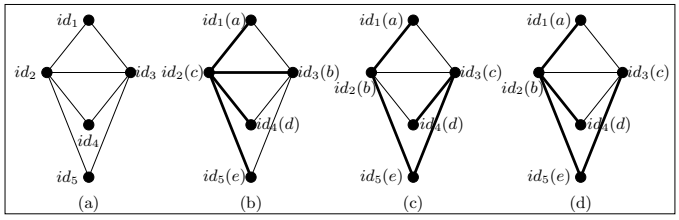

Figure 5: Three ways for finding g_{22}. (b) Using g_{11}. (c) Using g_9 and (d) Using g_{10}

4. OPTIMIZATION SCHEMES

GRAFT uses various optimization schemes which are crucial for its counting accuracy and running time. We discuss each of them in this section in the order of their significance.

4.1 Embedding criteria and the choice of FAE

From Figure 1, we can observe that some of the graphlets have multiple generation tree graphlets. For example, g_{22} has three generation tree graphlets: g_9, g_{10} and g_{11}. For the task of exact graphlet counting, we will obtain correct result, irrespective of the specific generation tree graphlet that we use for counting. But, complexity arises when GRAFT is used for approximate counting, which samples only a fraction of the edges.

If we embed g_{22} using g_{11} (Figure 5(b)), the first step is to embed g_{11} by mapping the chosen FAE (b, c) to the edge (id_3, id_2); and, the second step is to check if this embedding contributes to an induced embedding of g_{22}. Here, the mapped edge (b, c)(of g_{22}) belongs to an edge-orbit of size 1. Therefore, for approximate graphlet counting, if we map edge (b, c) with edge (id_3, id_2), there is a high probability of missing an embedding of g_{22}.

On the other hand, if we embed g_{22} using g_9 or g_{10} (Figure 5(c) and 5(d)), the edge (b, c) of g_{22} cannot be the FAE; rather, the FAE is the edge (b, e), which is mapped to the edge (id_2, id_5). The edge (b, e)(of g_{22}) belongs to an edge-orbit of size 6. Therefore, for approximate graphlet counting, if the edge (b, e) is used as FAE and is mapped to the edge (id_2, id_5) , the probability of missing an embedding of g_{22} is small. Hence, g_9 and g_{10} are more suitable generation tree graphlets for enumerating g_{22}. For instance, on ca-CondMat graph, using $p = 0.1$, the average counting error (in %) of approximate g_{22} using tree graphlet g_9 and g_{11} are 3.68 and 39.54. To summarize, the criteria for optimizing the accuracy of approximate graphlet counting is choosing the generation tree graphlet for which the FAE belongs to the largest edge-orbit.

4.2 Joint enumeration of multiple graphlets

As explained in Section 3.1.2, for counting a cyclic graphlet (say, g_x), GRAFT embeds the corresponding generation tree graphlet g_t followed by a validation to ensure that this embedding contributes to an induced embedding of g_x. If the validation step fails to find an induced embedding of g_x, then this specific embedding (not induced) of g_t does not contribute to the enumeration of the graphlet g_x. However, the embedding of g_t contributes to the induced embedding of some other graphlets, whose count should be incremented with this discovery. Therefore, if we count multiple graphlets (having the same generation tree graphlet) simultaneously, we will be able to share the workload of enumeration. Therefore, GRAFT's process of embedding graphlets having the same generation tree graphlet g_t is that after embedding the generation tree graphlet g_t, it finds the graphlet g_x whose induced embedding corresponds to this embedding of g_t. The above optimization improves the execution time significantly, as every embedding of g_t contributes to the enumeration (count) of exactly one induced embedding of the graphlets.

5. EXPERIMENTS

We perform several experiments to observe the performance of GRAFT. These experiments are performed on real-life graphs obtained from the following two web sites [3]. The name and statistics of these graphs are available from Table 1. Speedup factor and counting error(%) are two performance metrics that we use. We obtain the speedup factor by computing the ratio of execution times of GRAFT with some p less than 1 and GRAFT with $p = 1$; former is an approximate, and the latter is the exact graphlet counting. Apparently, GRAFT has a predictable value for the speedup factor; for an edge selection probability equal to p, its average speedup is close to $\frac{1}{p}$. This is so because we perform p fraction of work by randomly selecting p fraction of total edges for approximating the graphlet count. To obtain the counting error of GRAFT we first find the percentage of error in the count of each of the graphlets; then we average the error over all different graphlets.

5.1 Sampling factor vs average counting error

In this experiment, we show how the counting error (averaged over all the graphlets) of GRAFT varies with the sampling factor. Figure 6 shows our findings. As the sampling factor increases, the error of our algorithm diminishes. We show these results for three real world collaboration networks of different sizes. An important observation is that for the same sampling factor, the larger graph have smaller percentage error. So, as the graph grows, we can afford to decrease the sampling factor (thus increase the speedup factor), while keeping the counting error at the same level. For example, 5% sampling factor obtains a 12% error on ca-GrQc graph, but almost identical error percentage is achieved with a sampling factor of 1% on ca-CondMat, as the second graph is much larger.

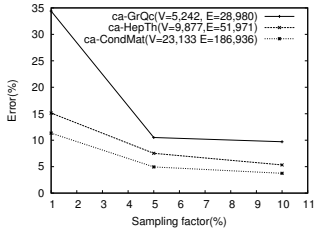

Figure 6: Sampling factor vs Average error measure for three networks.

5.2 Comparing the counting errors among different graphlets

While counting with GRAFT, the counting error of various graphlets varies based on the complexity and the size of the graphlet. To compare the relative counting error among different graphlets, we use the *ca-CondMat* graph dataset with the edge selection probability of 0.1. We repeat the counting process for 10 times and report the average counting errors in Figure 7. Each column in this graph represents

[3] `http://snap.stanford.edu/data/index.html` and `http://www-personal.umich.edu/~mejn/netdata`

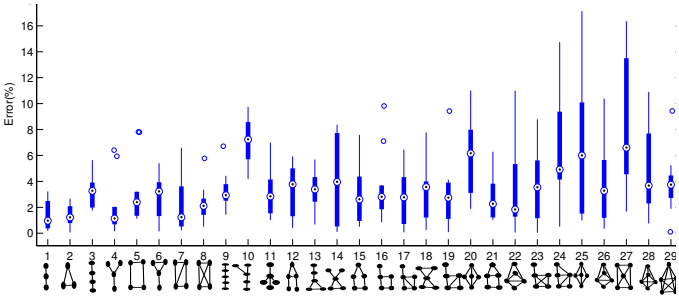

Figure 7: Box-plot for approximation errors of different graphlets in graph *ca-CondMat*

Network	# Vertex	# Edge	$R_{e/v}$	Speed-up	Error(%) GRAFT
ca-HepTh	9,875	25,973	2.6	10.0	5.11
OClinks	1,899	13,838	7.3	9.90	5.93
ca-CondMat	23,133	93,439	4.0	9.80	3.62
Polblogs	1,224	16,714	13.7	9.80	2.81
ca-AstroPh⋆	18,771	198,050	10.6	89.28	4.41

Table 1: Speedup and Error trade-off on five real-life networks.(⋆ marked graph's approximate graphlet counting was done with p=0.01.)

a distinct graphlet (which is shown as labels on the x-axis). The y-axis shows the percentage errors in counting. To show the variance of percentage error among different iterations, we show the results in a box-plot. From Figure 7 we observe that, the counting error increases with the number of vertices in the graphlets. Also, complex graphlets that have more cycles are more error-prone than tree graphlets.

5.3 Speed-up and Percentage error Comparison on different networks

In this experiment, we compare the speedup and percentage counting error of five real-world networks from collaboration, blog, and web domains that are shown in Table 1. For this we use $p = 0.1$, which gives us about $1/p = 10$ times speedup (except the case of ca-AstroPh, for which we use $p = 0.01$). The percentage error values are between 2.81% and 5.93%. Our method generally performs better as the graph becomes larger and denser. In case of network ca-AstroPh (which is much bigger than other networks), we use $p = 0.01$ which gives a much higher speed-up while maintaining the same level of accuracy.

5.4 GFD for different sampling factors

In this experiment, we justify the utility of approximate graphlet counting algorithm(GRAFT). One of the main objectives for graphlet counting is to obtain the graphlet frequency distribution (GFD) for large graph analysis. We like to show that, although counting errors prevails by adopting sampling in GRAFT, the effect of this error on GFD is negligible, because the latter compares the relative counts in a logarithm scale (log scale is used for GFD because the frequencies of different graphlets vary in exponential proportion). For this, we obtain graphlet count by running GRAFT for different sampling factors (10%, 7.5%, 5% and 1%) on various networks that we used in experiments discussed in Section 5.1. For these networks, We also obtain the exact graphlet count using GRAFT algorithm with p=1. We then find GFD from each of the results, and compare the GFD plots of a graph for different sampling factors. In Figure 8, we show this comparison for one network because the trend is similar for all the other networks. It is easy to see that the GFD histogram preserves its shape across different sampling factors, even for a sampling factor of 1% (which provides a 100-fold speedup).

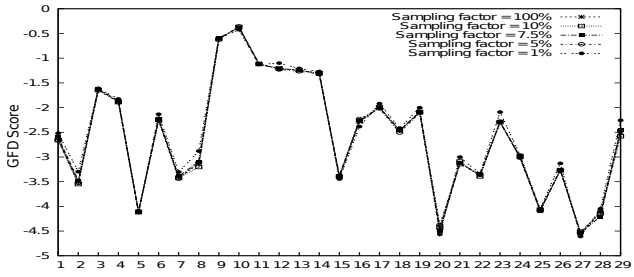

Figure 8: GFD of 29 graphlets for different sampling factor on ca-HepTh

6. CONCLUSIONS

In this paper, we present GRAFT, an effective method for approximate graphlet counting from large graphs. The algorithm offers significant speedup with a negligible counting error. For the same speedup factor, the counting accuracy of the algorithm improves with the size of the graph, so it is particularly suitable for counting graphlets in large real-life networks.

7. REFERENCES

[1] A.-L. Barabasi and R. Albert. Emergence of Scaling In Random Networks. *Science*, 286:509–512, 1999.

[2] S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca. Network analysis in the social sciences. *Science*, 323:892–895), 2009.

[3] E. C. E. Tsourakakis. Counting triangles in real-world networks using projections. *Knowl. Inf.*, 26:501–520, 2011.

[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 251–262, 1999.

[5] O. Kuchaiev, A. Stevanovic, W. Hayes, and N. Przulj. Graphcrunch 2: Software tool for network modeling, alignment and clustering. *BMC Bioinformatics*, 12(1), 2011.

[6] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proc. of the 11th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 2005.

[7] T. Milenkovic and N. Przulj. Uncovering biological network function via graphlet degree signatures. *Cancer Inform*, 6:257–273, 2008.

[8] N. Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

[9] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: Counting triangles in massive graphs with a coin. In *In Proc. of KDD*, 2009.

[10] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.