2010 IEEE 30th International Conference on Distributed Computing Systems Workshops

Efficient Counting of Network Motifs

Dror Marcus
School of Computer Science
Tel-Aviv University, Israel
Email: drormarc@post.tau.ac.il

Yuval Shavitt
School of Electrical Engineering
Tel-Aviv University, Israel
Email: shavitt@eng.tau.ac.il

***Abstract*—Counting network motifs has an important role in studying a wide range of complex networks. However, when the network size is large, as in the case of Internet Topology and WWW graphs counting the number of motifs becomes prohibitive. Devising efficient motif counting algorithms thus becomes an important goal.**

In this paper, we present efficient counting algorithms for 4-node motifs. We show how to efficiently count the total number of each type of motif, and the number of motifs adjacent to a node. We further present a new algorithm for node position-aware motif counting, namely partitioning the motif count by the node position in the motif. Since our algorithm is based on motifs, which are non-induced we also show how to calculate the count of induced motifs given the non-induced motif count. Finally, we report on initial implementation performance result using evaluation on a large-scale graph.

***Keywords*-efficient algorithms; network motifs; graphlets;**

I. Introduction

A. Motif Discovery

Network motifs, also termed graphlets, are small sub-graphs that can be found in larger graphs. In a seminal paper by Milo *et al.* [1], network motifs were defined as interaction patterns (or sub-graphs) occurring in a network more often than those in randomized networks. Significant motifs were found in various real world networks including protein interaction networks, neurobiological networks, social networks, World Wide Web (WWW) hyper-link networks, and the Internet autonomous systems (AS) network [2].

Recently, an increased interest is appearing in exploring the role of network motifs in networking. Feldman and Shavitt [3] suggested that the bi-fan motif may indicate existence of "points of presence" (PoPs) in the Internet's router level network. The distribution of the local number of triangles and the related clustering coefficient can be used to detect the presence of spamming activity in large scale Web graphs [4]. Hales and Arteconi [5] presented results from a motif analysis of networks produced by peer-to-peer protocols, showing that the motif profiles of such networks closely match protein structure networks.

Motif degree counting, namely, counting the number of motifs in which a node participates, was recently suggested as a method to classify nodes in the network into functional classes using the distribution of motifs a node is part of as an indication of its function in the network [6]. Stoica and Prieur [7] further refined the counting by taking into account the node *position* in the motif, and use it to analyze mobile phone communication graphs.

Gonen and Shavitt [8] were the first to suggest efficient algorithms for positional motif degree counting. However, some of their algorithms only approximate the count with high probability. We suggest here improved algorithms for this task, giving an exact count of all the 4-node motifs, and with better time complexity than the ones in [8]. We also present a simple algorithm for counting triangles, thus covering with efficient algorithm all motifs of size 4 and below. This leaves the problem of efficient 5-node counting open, which is usually the largest size motif used for classification.

For most network and node classification analysis we relate only to induced motifs, meaning that a subgraph is counted as a motif only if the motif is spanned by the subgraph nodes. However, the algorithms presented here and in previous works [8] count non induced motifs, namely a subgraph is counted even if additional edges exist between the subgraph nodes. Thus, we present simple transformation that given the non-induced motif-count calculates the corresponding induced-motif count.

B. Related Work

Batagelj and Mrvar [9] presented an algorithm which lists all triangles in a graph with time complexity $O(d|E|) = O(n|E|)$, where d is maximum nodal degree in the graph. This result was further improved to $O(|E|^{3/2})$ [10].

Itzhack *et al.* [11] gave an algorithm for counting all directed motifs up to size four, based on network decomposition via node removal. They claimed a time complexity of $O(|E|d^2 \log d)$.

Wernicke [12] presented the ESU algorithm, that enumerates all motifs of size k in a graph. The ESU algorithm starts with individual nodes in the graph and iteratively adds an additional node from the subgraph's neighborhood, until reaching subgraphs of size k. Stoica and Prieur [7] extended the ESU algorithm to count the number of position-aware motifs adjacent to each node in the graph.

This work was partially funded by the Israeli Science Foundation's center of knowledge grant 1685/07 and by the IBM 2008 Faculty Award.

978-0-7695-4079-5/10 $26.00 © 2010 IEEE
DOI 10.1109/ICDCSW.2010.41

Gonen and Shavitt [8] presented algorithms with time complexity $O\left(\frac{(3e)^k \cdot n \cdot |E| \cdot \log(1/\delta)}{\epsilon^2} + |E|^2 + |E| n \log n\right)$ that approximates the position aware motif degree, but only for k length paths, k-cycles and k-cycles with a chord motifs, where k is at most $O(\log n)$. They also count per all nodes all non-induced and undirected motifs up to size four, in time $O\left(\frac{n|E|\log(1/\delta)}{\epsilon^2} + |E|^2 + |E| n \log n\right)$.

C. Our Contribution

In this paper we present a set of algorithms that *exactly* count all induced and non-induced position-aware motifs of up to size four, adjacent to each node in the graph, with time complexity of $O(d \cdot |E| + |E|^2)$.

Partly based on the algorithms of Gonen and Shavitt [8], our algorithms are thus first to *exactly* count the non-induced motif adjacent to a node. From the obtained non-induced count results we later deduce the induced count, which are often of more interest in network analysis.

Specifically, we present algorithms that count, for each node, all non-induced Tailed Triangle, 4-node cycles with chord (chordal cycles), and a path of length three motifs in time complexity of $O(d \cdot |E|)$ and count non-induced cliques and cycles in time complexity of $O(d \cdot |E| + |E|^2)$. We also present a method to obtain the induced motif count in an undirected graph from the non-induced motif count, for motifs up to size four. Since most real-world complex networks are sparse (i.e. $|E| \ll n^2$), analyzing the runtime of these algorithms on such networks shows that the runtime bound is significantly lower than the runtime of the trivial exhaustive search over all possible edges/nodes.

Following analysis in Section II presents the non-induced position-aware counting algorithms for all motifs up to size four. Section III introduces the post-processing technique used to convert the non-induced count to their respective induced results. Finally, Section IV discusses further work and provides initial test results using a customly written software-package, implementing these algorithms.

II. Non-Induced Motif Count

All algorithms described in the following section assume an undirected simple input graph $G(V, E)$, which is represented by an adjacency list. Furthermore it is assumed that the graph vertices are labeled by the integers $\{1, 2, ..., n\}$*. We denote by $N(v)$ the set of neighbor nodes of v (i.e., $N(v) = \{u \in V | (v, u) \in E\}$).

A node v is adjacent to motif m_i if v is a vertex of motif m_i. For each motif m_i, $u \in S$ (where S is the set of non-isomorphic vertices in m), we say that v is adjacent at *position* u, and denote it by $m_{i.u}$, if $v \equiv u$ or if v is adjacent to m and is isomorphic to u. For simplicity, we say that v is adjacent to $m_{i.u}$, and $m_{i.u}$ are called *position aware motifs*. For example, the cycle with a chord of size four has two position aware motifs: one for the two nodes connected to the chord ($m_{2.1}$ in Fig. 1) and one for the other two nodes ($m_{2.2}$ in Fig. 1).

*Any input graph can be converted to this form in $O(n \log n + |E| \log n)$ preprocessing time using a dictionary data structure

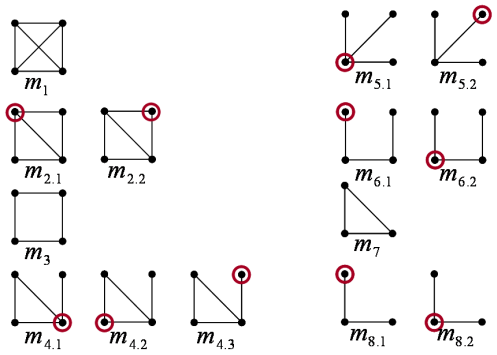

Figure 1. All three and four node undirected position aware motifs. Location of the node in the motif is marked by an additional circle (node position is not marked in symmetric motifs).

A. Counting Triangles

The following algorithm counts, for each node $u \in V$, all triangles (motif m_7, Fig. 1) adjacent to u: The algorithm (hereafter termed Algorithm 1) iterates over all edges in the graph (lines 2-6). For each edge $e(u, v) \in E$, it counts all triangles that are edge joint with $e(u, v)$ (i.e., $e(u, v)$ is an edge in the triangle), updating the respected triangle count values of both node u and node v. This edge joint triangle count is obtained using the Merge procedure (line 9), which returns all nodes sharing an edge with both u and v (note that NodeArray is not useful for counting triangles, but will be useful for subsequent algorithms calling the Merge procedure). Finally, since each node is connected to two edges in each triangle we count each triangle twice. This over-count is fixed in the final post-processing loop (line 8).

Theorem II.1. *Algorithm 1 counts, for every node $v \in V$, the exact amount of occurrences of the triangle motif (m_7) that v is part of, with time complexity of $O(d|E|)$.*

Proof: Assume, by contradiction, that there exists a node $v \in V$, for which the algorithm outputs a wrong value.

If the algorithm outputs a value greater than the amount of triangles adjacent to v, there must exist an additional node $u \in N(v)$ that shares a neighbor w with v, but the nodes v,u,w do not form a triangle in G, in contradiction to the existence of edges (v, u),(v, w),(u, w) in the graph.

Therefore, it must be that v is adjacent to more triangles the algorithm outputs. So there must exist a triangle $\triangle(v, u, w)$ that is counted in the main iteration (lines 2-6) less than two times for v. Assume, without loss of generality, that $v < u < w$. Since u and v share an edge, there must be an iteration step where v and u are selected. In this step,

Algorithm 1 Counting triangles

```
1: procedure TRIANGLECOUNT(G)
2:     for all v ∈ V(G) do
3:         for all u ∈ N(v), v < u do
4:             Merged ← MERGE(G, v, u, V_arr)
5:             m_7[v] ← m_7[v] + |Merged|
6:             m_7[u] ← m_7[u] + |Merged|
7:     for all v ∈ V(G) do
8:         m_7[v] ← m_7[v]/2

9: procedure MERGE(G, v, u, NodeArray)
10:    for all w ∈ N(u) s.t. w ≠ v do
11:        NodeArray[w] ← 1
12:    for all w ∈ N(v), w ≠ u do
13:        if NodeArray[w] = 1 then
14:            NodeArray[w] ← 3
15:            list ← APPENDTOLIST(list, w)
16:        else
17:            NodeArray[w] ← 2
18:    return list
```

since c is a neighbor of both u and v it is added to the merged list, and $\triangle(v,u,w)$ is counted for nodes a. Similarly there must be an iteration step for ($v = a$, $u = c$) where $\triangle(a,b,c)$ is counted again.

The time complexity for the Merge procedure run is $O(|N(v)| + |N(u)|) = O(d)$, the main loop (lines 2-6) iterates over all edges in graph G, and the fixup loop (line 8) iterates over all nodes, giving us the upper bound: $O(\sum_{v\in V}\sum_{u\in N(v)} d) + O(n) = O(d\cdot|E|)$ ■

B. Counting Counting Cycle with a Chord

Theorem II.2. *Algorithm 2 exactly counts, for every node $v \in V$, all non-induced occurrences of motifs $m_{2.1}$ and $m_{2.2}$ v is part of, in total time complexity of $O(d|E|)$.*

Proof: $\forall e(u,v) \in E$, $\forall w,t \in (N(v)\cap N(u))$ ($w \neq t \neq v \neq u$) we get that nodes {u,v,w,t} induce a cycle with $e(u,v)$ as the chord. Also, for every cycle with a chord $m_2(v,u,w,t)$ having $e(u,v) \in E$ as the chord edge it holds that $w,t \in (N(v)\cap N(u))$. Therefore, there are exactly $\binom{|N(v)\cap N(u)\setminus\{u,v\}|}{2}$ different cycles with chord (chordal cycles), where $e(u,v)$ is the chord, and, $\forall w \in N(v)\cap N(u)\setminus\{u,v\}$, exactly $\binom{|N(v)\cap N(u)\setminus\{u,v,w\}|}{1}$ of these chordal cycles are adjacent to node w.

Since every chordal cycles has only one chord edge, and since Algorithm 2 iterate over all edges once, we get that each chordal cycles is counted exactly once for each adjacent node and their respective position aware motifs.

Using similar runtime analysis shown in the proof of Theorem II.1, we get that the time complexity is bounded by

$$O\left(\sum_{v\in V}\sum_{u\in N(v)}\left(d + |N(v)\cap N(u)|\right)\right) = O(d|E|)$$ ■

Note that the runtime for this exact counting algorithm is better than Gonen and Shavitt [8] approximation count.

Algorithm 2 Counting non-induced 4-cycles with a chord

```
1: procedure CHORDCYCLECOUNT(G)
2:     for all v ∈ V(G) do
3:         for all u ∈ N(v), v < u do
4:             Merged ← MERGE(G, v, u, V_arr)
5:             m_2.1[v] ← m_2.1[v] +
6:                     |Merged| · (|Merged| − 1)/2
7:             m_2.1[u] ← m_2.1[u] +
8:                     |Merged| · (|Merged| − 1)/2
9:             for all w ∈ Merged do
10:                m_2.2[w] ← m_2.2[w] + (|Merged| − 1)
```

C. Counting Tailed Triangles

The tailed triangles counting algorithm is based on the algorithm presented by Gonen and Shavitt [8]. Our algorithm uses the triangle count results obtained using Algorithm 1 rather than Gonen's cycle approximation count algorithm. Furthermore, $m_{4.3}$ is computed in a more efficient manor. For each $v \in V$, in order to count occurrences of $m_{4.3}$ adjacent to v, the algorithm needs to count all triangles in v's neighborhood that v is not a part of. Rather than recounting all triangles on the graph not connected to v as done in [8], the algorithm uses the original triangles count (computed only once for all nodes of the graph) to compute $m_{4.3}$, and only later fixes any error that occurred due to falsely counting triangles adjacent to v.

Theorem II.3. *Algorithm 3 finds, for every node $v \in V$, all non-induced occurrences of motifs $m_{4.1}$, $m_{4.2}$ and $m_{4.3}$ v is part of, in total time complexity of $O(d|E|)$.*

Proof: By Theorem II.1 counting triangles can be done correctly in time $O(d\cdot|E|)$. Correctness of counting $m_{4.2}$ and $m_{4.1}$ is given by Gonen and Shavitt [8]. It remains show that motif $m_{4.3}$ is counted correctly. For every node v, define $Tr(u)$ as the number of triangles adjacent to node u, and $Tr_v(u)$ the number of triangles adjacent to u that are not adjacent to v. The number of motifs of type $m_{4.3}$ that adjacent to v is given by

$$\begin{aligned}\sum_{u\in N(v)} Tr_v(u) &= \sum_{u\in N(v)}\left(Tr(u) - (Tr(u) - Tr_v(u))\right)\\ &= \sum_{u\in N(v)} Tr(u)\\ &\quad - \sum_{u\in N(v)}\left(Tr(u) - Tr_v(u)\right)\\ &= \sum_{u\in N(v)}\left(Tr(u)\right) - 2Tr(v)\end{aligned}$$

Runtime complexity analysis is divided into three parts: counting triangles in $O(d|E|)$ time (line 2), first edge iteration (lines 3 - 9), last edge iteration (lines 10 - 14). Using similar analysis used in Theorem II.2 we get that the time complexity for the first iteration is $O(d|E|)$. Thus the

total time complexity of Algorithm 3 is:

$$O(d|E|) + O(d|E|) + O(|E|) = O(d|E|)$$

■

The above runtime result, given for the exact tailed triangle counting algorithm, is significantly better then Gonen and Shavitt [8] approximation count.

Algorithm 3 Counting non-induced tailed triangles

```
1: procedure TAILTRIANGLECOUNT(G)
2:     TRIANGLECOUNT(G)
3:     for all v ∈ V(G) do
4:         for all u ∈ N(v), v < u do
5:             Merged ← MERGE(G, v, u, V_arr)
6:             tails_v ← max{0, (|N(v)| − 2)}
7:             tails_u ← max{0, (|N(u)| − 2)}
8:             for all w ∈ Merged do
9:                 m_4.2[w] ← m_4.2[w] + tails_v + tails_u
10:    for all v ∈ V(G) do
11:        m_4.1[v] ← max{0, (m_7[v] · (|N(v)| − 2))}
12:        for all u ∈ N(v) do
13:            m_4.3[v] ← m_4.3[v] + m_7[u]
14:        m_4.3[v] ← m_4.3[v] − 2 · m_7[v]
```

D. Counting Four Nodal Cliques

Counting cliques is done in a similar fashion to [8]. We show that the merged neighbors sort done in the main edge loop is redundant, thus lowering the runtime bound to $O(d|E| + |E|^2)$.

Theorem II.4. *Algorithm 4 finds, for every node $v \in V$, all non-induced occurrences of motif m_1 v is part of, in total time complexity of $O(d|E| + |E|^2)$.*

Proof: Correctness follows from the full version of [8]. Using computation similar to those shown in the proof of Theorem II.1, we get the time complexity of Algorithm 4 to be:

$$O\left(\sum_{v\in V}\sum_{u\in N(v)}\left(d + \sum_{w\in(N(v)\cap N(u))}|N(w)|\right)\right)$$
$$= O\left(\sum_{v\in V}\sum_{u\in N(v)}(d) + \sum_{v\in V}\sum_{u\in N(v)}\sum_{w\in(N(v)\cap N(u))}|N(w)|\right)$$
$$= O(d|E|) + O(|E|^2)$$

■

Note that the runtime for this exact four nodal cliques counting algorithm is better then the one in Gonen and Shavitt [8].

E. Counting Four Nodal Cycles

Theorem II.5. *Algorithm 5 finds, for every node $v \in V$, all non-induced occurrences of motif m_3 v is part of, in total time complexity of $O(d|E| + |E|^2)$.*

Algorithm 4 Counting 4-cliques

```
1: procedure CLIQUECOUNT(G)
2:     for all v ∈ V(G) do
3:         V_arr[v] ← 0
4:     for all v ∈ V(G) do
5:         for all u ∈ N(v), v < u do
6:             Merged ← MERGE(G, v, u, V_arr)
7:             for all w ∈ Merged do
8:                 for all t ∈ N(w), w < t do
9:                     if V_arr[t] = 3 then
10:                        m_1[v] ← m_1[v] + 1
11:                        m_1[u] ← m_1[u] + 1
12:            for all w ∈ N(v) ∪ N(u) do
13:                V_arr[w] ← 0
14:    for all v ∈ V(G) do
15:        m_1[v] ← m_1[v]/3
```

Proof: Let $Cyc_{(v,u)}$ be the number of cycles size four going through the edge $e(v,u)$. Trivially, $Cyc_{(v,u)}$ is also the number of pairs $w \in N(v)$, $t \in N(u)$ such that $e(w,t) \in E$. For all nodes $v \in V$, since every cycle adjacent to v has exactly two edges connected to v, the exact amount of cycle adjacent to v, Cyc_v is:

$$Cyc_v = \frac{\sum_{u\in N(v)} Cyc_{(v,u)}}{2}$$

Using computation similar to those shown in the proof of Theorem II.1, the time complexity of Algorithm 5 is:

$$O\left(\sum_{v\in V}\sum_{u\in N(v)}\left(d + \sum_{w\in N(v)}|N(w)|\right)\right)$$
$$= O(d|E|) + O(|E|^2)$$

■

Algorithm 5 Counting 4-cycles

```
1: procedure CYCLECOUNT(G)
2:     for all v ∈ V(G) do
3:         V_arr[v] ← 0
4:     for all v ∈ V(G) do
5:         for all u ∈ N(v) s.t. v < u do
6:             MERGE(G, v, u, V_arr)
7:             for all w ∈ N(v)\{u} do
8:                 for all t ∈ N(w) do
9:                     if V_arr[t] = 2 or V_arr[t] = 3 then
10:                        m_3[v] ← m_3[v] + 1
11:                        m_3[u] ← m_3[u] + 1
12:            for all w ∈ N(v) ∪ N(u) do
13:                V_arr[w] ← 0
14:    for all v ∈ V(G) do
15:        m_3[v] ← m_3[v]/2
```

F. Counting Four Nodal Path

Theorem II.6. *Algorithm 6 finds, for every node $v \in V$, all non-induced occurrences of motif m_6 v is part of, in total time complexity of $O(d|E|)$. For each edge $e(u,v) \in E$ the*

algorithm counts all paths of length three having $e(u,v)$ in the path center.

Proof: Let $P_{u,v}$ be the exact amount of paths having edge $e(u,v)$ in the center, $P_{(u,v),(w,u)}$ the exact amount of paths starting with edge $e(w,u)$ and having edge $e(u,v)$ in the center it holds that:

$$P_{u,v} = (|N(v)\backslash\{u\}| \cdot |N(u)\backslash\{v\}|) - |N(v) \cap N(u)|$$
$$P_{(u,v),(w,u)} = |N(v)\backslash\{u\}| - |N(v) \cap \{w\}|$$

And, $\forall v \in V$:

$$m_{6.1}[v] = \sum_{u \in N(v)} \sum_{w \in N(u)} (P_{(u,w),(v,u)})$$
$$m_{6.2}[v] = \sum_{u \in N(v)} (P_{u,v})$$

The time complexity analysis is similar to that of Theorem II.5. ■

Algorithm 6 Counting 4-node paths

```
1: procedure PathCount(G)
2:     for all v ∈ V(G) do
3:         V_arr[v] ← 0
4:     for all v ∈ V(G) do
5:         for all u ∈ N(v) s.t. v < u do
6:             Merged ← Merge(G, v, u, V_arr)
7:             m_6.2[v]
8:                 ← m_6.2[v] + (|N(v)\{u}| · |N(u)\{v}|) − |Merged|
9:             m_6.2[u]
10:                ← m_6.2[u] + (|N(v)\{u}| · |N(u)\{v}|) − |Merged|
11:            for all w ∈ N(v)\{u} do
12:                if V_arr[w] = 3 then
13:                    m_6.1[w] ← m_6.1[w] + |N(u)\{v}| − 1
14:                else
15:                    m_6.1[w] ← m_6.1[w] + |N(u)\{v}|
16:            for all w ∈ N(u)\{v} do
17:                if V_arr[w] = 3 then
18:                    m_6.1[w] ← m_6.1[w] + |N(v)\{u}| − 1
19:                else
20:                    m_6.1[w] ← m_6.1[w] + |N(v)\{u}|
21:            for all w ∈ N(v) ∪ N(u) do
22:                V_arr[w] ← 0
```

G. Counting Claws

Counting Claws is done in the same way described in [8]. For each $v \in V$ the claw motifs count is obtained using the following equations:

$$m_{5.1}[v] = \binom{|N(v)|}{3}$$
$$m_{5.2}[v] = \sum_{u \in N(v)} \binom{|N(u)|-1}{2}$$

III. Obtaining Induced Motifs

The per node non-induced motif count, collected using the algorithms presented in the previous section, can be converted to an induced count using a post-processing technique described in the following section.

Denote by $NI(m)$ the total number of non-induced appearances of motif m in G, $I(m)$ the total number of induced appearances of motif m in graph G. In addition $\forall v \in V(G)$ denote $NI_v(m)$ by the number of non-induced position aware motifs v participates in G, and $I_v(m)$ the number of induced position aware motifs.

A. 4-Clique

Denote by m_1 the 4-Clique motif.
Trivially:

$$I(m_1) = NI(m_1)$$

Since all nodes in the clique are isomorphic, there is one position aware clique, so the above result also holds for each node adjacent clique count (i.e. $I_v(m_1) = NI_v(m_1)$).

B. 4-Cycle with a chord

Denote by m_2 the 4-Cycle with a chord motif. A non-induced motif can be found only from an induced motif with a node degree vector which is not smaller for every component. Thus, m_2 can appear as a subgraph isomorphism only in m_1 and only by removing a single edge. So the 4-cycle with a chord induced count is obtained by:

$$I(m_2) = NI(m_2) - 6I(m_1) = NI(m_2) - 6NI(m_1)$$

Using similar calculation we get, for each node $v \in V$:

$$I_v(m_{2.1}) = NI_v(m_{2.1}) - 3I_v(m_1)$$
$$I_v(m_{2.2}) = NI_v(m_{2.2}) - 3I_v(m_1)$$

C. 4-Cycle

Denote by m_3 the 4-Cycle motif. According to the node degree vector of motif m_3, m_3 can appear as a subgraph isomorphism only in m_1 and m_2. Removing any two matching edges from m_1 induces a cycle. Three such matching exists. Removing the chord edge from m_2 induces a cycle. The 4-cycle induced count is obtained by:

$$\begin{aligned} I(m_3) &= NI(m_3) - I(m_2) - 3I(m_1) \\ &= NI(m_3) - NI(m_2) + 6NI(m_1) - 3NI(m_1) \end{aligned}$$

Using similar calculation we get, for each node $v \in V$:

$$I_v(m_3) = NI_v(m_3) - I_v(m_{2.1}) - I_v(m_{2.2}) - 3I_v(m_1)$$

D. Tailed Triangles

Denote by m_4 the Tailed Triangles motif. According to the node degree vector of motif m_4, m_4 can appear as a subgraph isomorphism only in the m_1 and m_2. Removing any two edges from any one node in m_1 induces a Tailed Triangle. Removing any edge from the cycle in m_2 (All 4 edges other than the edge between the two nodes with degree $= 2$) induces a Tailed Triangle. Thus, I(m_4) can be obtained by:

$$\begin{aligned} I(m_4) &= NI(m_4) - 4I(m_2) - (4 \cdot \binom{3}{2})I(m_1) \\ &= NI(m_4) - 4NI(m_2) + 12NI(m_1) \end{aligned}$$

Using similar calculation we get, for each node $v \in V$:

$$\begin{aligned} I_v(m_{4.1}) &= NI_v(m_{4.1}) - 2I_v(m_{2.2}) - 3I_v(m_1) \\ I_v(m_{4.2}) &= NI_v(m_{4.2}) - 2I_v(m_{2.1}) - 3I_v(m_1) \\ I_v(m_{4.3}) &= NI_v(m_{4.3}) - 2I_v(m_{2.1}) - 2I_v(m_{2.2}) \\ &\quad -6I_v(m_1) \end{aligned}$$

E. Claws

Denote by m_5 the Claw motif. According to the node degree vector of motif m_5, m_5 can appear as a subgraph isomorphism only in m_1, m_2, and m_4.

For each node in m_1, removing all the edges that are not connected to it induces a claw, four such nodes exist. For each node v in m_2 with degree ≥ 3, removing all the edges that are not connected to v induces a claw, two such nodes exist. Finally, there is only one node in m_4 with degree ≥ 3. Removing all the edges that are not connected to this node induces a claw. So $I(m_5)$ can be deduced by:

$$\begin{aligned} I(m_5) &= NI(m_5) - I(m_4) - 2I(m_2) - 4I(m_1) \\ &= NI(m_5) - NI(m_4) + 2NI(m_2) - NI(m_1) \end{aligned}$$

Using similar calculation we get, for each node $v \in V$:

$$\begin{aligned} I_v(m_{5.1}) &= NI_v(m_{5.1}) - I_v(m_{4.2}) - I_v(m_{2.1}) - I_v(m_1) \\ I_v(m_{5.2}) &= NI_v(m_{5.2}) - I_v(m_{4.1}) - I_v(m_{4.3}) \\ &\quad -I_v(m_{2.1}) - 2I_v(m_{2.2}) - 3I_v(m_1) \end{aligned}$$

F. Simple Path of Length Three

Denote by m_6 the simple path motif of length three(a path with three edges). According to the node degree vector of motif m_6, m_6 can appear as a subgraph isomorphism in all motifs other than m_5.

Removing an edge, connecting a node with degree 3 and a node with degree 2, from m_4 induces a path, 2 such edges exist. Removing any single edge from m_3 induces a path, 4 such edges exist. Removing the chord and any other edge, or removing 2 matching edges that are not the chord from m_2 induces a path, giving a total of 6 possible paths. In m_1 every node pair are connected. Therefore, any permutation of the 4 nodes (4!) creates a legal path. The edges are undirected so we count each path twice in the permutation (once for each direction), giving us a total of $4!/2 = 12$ distinct paths (removing all edges that are not in the selected path induces m_6). Finally, $I(m_6)$ can be deduced by:

$$\begin{aligned} I(m_6) &= NI(m_6) - 2I(m_4) - 4I(m_3) - 6I(m_2) \\ &\quad -12I(m_1) \\ &= NI(m_6) - 2NI(m_4) - 4NI(m_3) + 6NI(m_2) \\ &\quad -12NI(m_1) \end{aligned}$$

Using similar calculation we get, for each node $v \in V$:

$$\begin{aligned} I_v(m_{6.1}) &= NI_v(m_{6.1}) - 2I_v(m_{4.1}) - I_v(m_{4.3}) \\ &\quad -2I_v(m_3 - 2I_v(m_{2.1}) - 4I_v(m_{2.2}) \\ &\quad -6I_v(m_1) \\ I_v(m_{6.2}) &= NI_v(m_{6.2}) - 2I_v(m_{4.2}) - I_v(m_{4.3}) \\ &\quad -2I_v(m_3 - 2I_v(m_{2.1}) - 4I_v(m_{2.2}) \\ &\quad -6I_v(m_1) \end{aligned}$$

IV. Implementation and Discussion

We compared our algorithm with FANMOD [13], which is the previous fastest algorithm for this task, using an Intel Core 2 Quad Q8400 CPU machine.

The FANMOD algorithm can detect motifs up to a size of eight vertices by enumerating all subgraphs of a given size within the input network or by uniformly sampling them using the algorithm described by Wernicke [12]. FANMOD may also determine the frequency of motifs in a user-specified number of random graphs, thereby detecting motifs which are over (under) represented in the original network. For the purpose of our performance analysis we limited both FANMOD run modes, i.e., sampling and full-enumeration, to run only on the input graph, while the number of random networks, was set to zero.

Using an Internet autonomous systems (AS) graph collected over a month by the DIMES project [14], which contains 26,561 nodes and 92,584 edges, the run time for counting all the network motifs was 40 min while FANMOD's sampling algorithm and FANMOD's full enumeration algorithm performed the same task at 2 hours and at 48 hours respectively.

We also used smaller scale free graphs, which were generated using the iNet Internet Topology Generator [15], in order to see how the three algorithms' run time depends on the graph size. Execution runtimes for our algorithm for graphs generated with 5000 nodes, 10000 nodes, and 20000 nodes were 11 sec, 64 sec, and 12 min, respectively. Using FANMOD with sampling the corresponding runtimes were 57 sec, 3.5 min, and 17 min. Finally, the FANMOD full enumeration algorithm runtimes were 7 min, 34 min, 7 hours. Clearly, the algorithm presented in this report greatly outperforms FANMOD's full enumeration algorithm and moreover outperformed FANMOD's less accurate sampling

algorithm. Further performance analysis is currently being conducted for comparing our algorithm with the approach suggested by Itzhack *et al.* [11].

Future work will focus on using the obtained results to classify nodes of the Internet. Similar to [6], we intend to use the motif enumeration per node in order to classify the nodes of the Internet AS graph, based on their functionality as it is expressed in their motif count vector.

REFERENCES

[1] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, pp. 824–827, 2002.

[2] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon, "Superfamilies of evolved and designed networks," *Science*, vol. 303, pp. 1538–1542, 2004.

[3] D. Feldman and Y. Shavitt, "Automatic large scale generation of internet PoP level maps," in *GLOBECOM*, 2008, pp. 2426–2431.

[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis., "Effcient semi-streaming algorithms for local triangle counting in massive graphs," in *ACM SIGCOMM international conference on Knowledge discovery and data mining (KDD)*, 2008, pp. 16–24.

[5] D. Hales and S. Arteconi, "Motifs in evolving cooperative networks look like protein structure networks," *The Journal of Networks and Heterogeneous Media*, vol. 3, no. 2, pp. 239–249, 2008, special Issue of ECCS'07.

[6] T. Milenkovic and N. Przulj, "Uncovering biological network function via graphlet degree signatures," *Social Networks*, vol. 6, pp. 257–273, 2008.

[7] A. Stoica and C. Prieur, "Structure of neighborhoods in a large social network," in *International Conference on Computational Science and Engineering (SocialCom)*, Aug. 2009.

[8] M. Gonen and Y. Shavitt, "Approximating the number of network motifs," in *WAW '09: Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph*, Feb. 2009, pp. 13–24.

[9] V. Batagelj and A. Mrvar, "A subquadratic triad census algorithm for large sparse networks with small maximum degree," *Social Networks*, vol. 23, pp. 237–243, 2001.

[10] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theor. Comput. Sci.*, vol. 407, no. 1-3, pp. 458–473, 2008.

[11] R. Itzhack, Y. Mogilevski, and Y. Louzoun, "An optimal algorithm for counting network motifs," *Physica A*, vol. 381, pp. 482–490, 2007.

[12] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 3, pp. 347–359, Oct. 2006.

[13] S. Wernicke and F. Rasche, "FANMOD: a tool for fast network motif detection," *Bioinformatics*, vol. 22, pp. 1152–1153, 2006.

[14] Y. Shavitt and E. Shir, "DIMES: Let the internet measure itself," *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 71–74, Oct. 2005.

[15] C. Jin, C. J. Qian, and S. Jamin, "Inet: Internet topology generator," 2000, http://topology.eecs.umich.edu/inet.