ARTICLE IN PRESS

Computer Networks xxx (2011) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

RAGE – A rapid graphlet enumerator for large networks ☆

D. Marcus [a,*], Y. Shavitt [b]

[a] School of Computer Science, Tel Aviv University, Israel
[b] School of Electrical Engineering, Tel Aviv University, Israel

ARTICLE INFO

Article history:
Received 18 February 2011
Received in revised form 4 July 2011
Accepted 25 August 2011
Available online xxxx

Keywords:
Efficient algorithms
Network motifs
Graphlets
Subgraph enumeration

ABSTRACT

Counting network graphlets (and motifs) was shown to have an important role in studying a wide range of complex networks. However, when the network size is large, as in the case of the Internet topology and WWW graphs, counting the number of graphlets becomes prohibitive for graphlets of size 4 and above. Devising efficient graphlet counting algorithms thus becomes an important goal.

In this paper, we present efficient counting algorithms for 4-node graphlets. We show how to efficiently count the total number of each type of graphlet, and the number of graphlets adjacent to a node. We further present a new algorithm for node position-aware graphlet counting, namely partitioning the graphlet count by the node position in the graphlet. Since our algorithms are based on non-induced graphlet count, we also show how to calculate the count of induced graphlets given the non-induced count.

We implemented our algorithms on a set of both synthetic and real-world graphs. Our evaluation shows that the algorithms are scalable and perform up to 30 times faster than the state-of-the-art. We then apply the algorithms on the Internet Autonomous Systems (AS) graph, and show how fast graphlet counting can be leveraged for efficient and scalable classification of the ASes that comprise the Internet. Finally, we present RAGE, a tool for rapid graphlet enumeration available online.

1. Introduction

1.1. Graphlet discovery

Network graphlets are small connected sub-graphs of a larger network [1]. In a seminal paper by Milo et al. [2], network motifs were defined as interaction patterns (or graphlets) 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 [3].

Recently, there is an increased interest in exploring the role of network motifs and graphlets in networking. Feldman and Shavitt [4] suggested that the bi-fan graphlet (a directed 2–2 full bi-partite graph) 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 [5]. Hales and Arteconi [6] 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.

Graphlet degree counting, counting the number of graphlets in which a node participates, was recently suggested as a method to classify nodes in the network into functional classes [7]. The suggestion assumes that the graphlet degree vector of a node portrays its function in the network.

Gonen and Shavitt [8,20] were the first to suggest efficient algorithms for positional graphlet degree counting,

☆ This work was partially funded by the Israeli Science Foundation's center of knowledge grant 1685/07.

* Corresponding author.
E-mail addresses: drormarc@post.tau.ac.il (D. Marcus), shavitt@eng.tau.ac.il (Y. Shavitt).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.08.019

namely partitioning the graphlet count by the node position in the graphlet. 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 graphlets, and with better time complexity than the ones in their publication [8]. We also provide a simple algorithm for counting triangles, based on new-vertex-listing [9] and slightly modified to assist in per-node count. We thus cover with efficient algorithms all graphlets of size 4 and below. This leaves the problem of efficient 5-node counting open, which is usually the largest size graphlet used for classification.

For most network and node classification analysis we relate only to *induced* graphlets, meaning that a subgraph $H(V_H,E_H)$ is counted as graphlet $g(V_g,E_g)$ only if there exists an isomorphism $f: V_H \to V_g$ such that $(u,v) \in E_H$ *iff* $(f(u), f(v)) \in E_g$. However, the algorithms presented here and in some previous works [8,20] count *non-induced* graphlets, namely a subgraph is counted even if additional edges exist between the subgraph nodes ($(f(u),f(v)) \in E_g$ *if* $(u,v) \in E_H$). Thus, we present simple transformation that given the non-induced graphlet-count calculates the corresponding induced-graphlet count.

1.2. Related Work

Batagelj and Mrvar [10] 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 $O(|E|^{3/2})$ [9].

Itzhack et al. [11] gave an algorithm for counting all directed graphlets up to size 4, based on network decomposition via node removal. They claimed a time complexity of $O(|E|d^2 \log d)$. Shervashidze et al. [12] provide methodology for counting all undirected graphlets up to size five in bounded degree graphs, mostly by using the intersection between neighborhoods of each node pair of the graph edges.

Wernicke [13] 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 [14] extended the ESU algorithm to count the number of position-aware graphlets adjacent to each node in the graph.

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 graphlet degree, but only for k length paths, k-cycles and k-cycles with a chord graphlet, where k is at most $O(\log n)$. They also count per all nodes all non-induced and undirected graphlets up to size four, in time $O\left(\frac{n|E| \log(1/\delta)}{\epsilon^2} + |E|^2 + |E| n \log n\right)$. Gonen et al. [15] gave a sub-linear algorithm for approximating the non-induced star count, for any given star size.

1.3. Our contribution

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

Specifically, we present algorithms that count: for each node, all non-induced tailed triangles, 4-node cycles with chord (chordal cycles), and a path of length three graphlets in time complexity of $O(d \cdot |E|)$; Count cliques and non-induced cycles in time complexity of $O(d \cdot |E| + |E|^2)$. We also present a method to obtain the induced graphlet count from the non-induced graphlet count, for graphlet up to size four. Parts of the techniques presented are similar in spirit to those noted in [12], for global graphlet counting.

Since most real-world complex networks are sparse (i.e. $|E| = O(n)$), analyzing the runtime of these algorithms on such networks shows that the runtime bound is much lower than the runtime of the trivial exhaustive search, over all possible edges/nodes.

As an application of our algorithms, we present methods enabling fast graphlet counting to be used for role classification of Internet Autonomous Systems [16], a classification which helps understand the complex ecosystem of the commercial Internet. Our methods improve on current work, by mitigating the need of inferring the relationship between ASes, a task which is known to be hard and inaccurate [17–19].

2. Non-induced graphlet 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 $\{1,2,\ldots,n\}$.[1] 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 graphlet m_i if v is a vertex of graphlet m_i. Given S, the set of non-isomorphic vertices in m_i, we say that v is adjacent to graphlet m_i at *position* $u \in S$ (or adjacent to $m_{i \cdot u}$), if $v \equiv u$ or if v is adjacent to m_i and is automorphically equivalent to u. For simplicity, we define $m_{i \cdot u}$ as *position aware graphlet*. For example, the cycle with a chord of size four has two position-aware graphlets: 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).

Although this section discusses per-node graphlet count, the global graphlet count, i.e. the global number of different graphlets appearing in the graph, can be deduced using the counting algorithms presented here: For each graphlet pattern, we select a single position aware graphlet, sum all the per-node counts in the graph and divide this sum by the number of nodes in the graphlet that are automorphically equivalent to the selected position.

2.1. Counting triangles

Algorithm 1 counts, for each node $u \in V$, all triangles (graphlet m_7, Fig. 1) adjacent to u: The algorithm 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

[1] Any input graph can be converted to this form in $O(n \log n + |E| \log n)$ pre-processing time using a dictionary data structure.

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

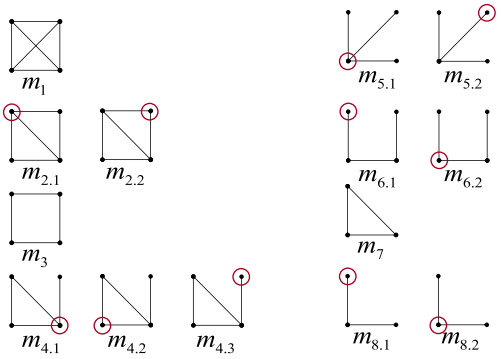

Fig. 1. All three and four node undirected position aware graphlets. Location of the node in the graphlet is marked by an additional circle (node position is not marked in symmetric graphlets).

$e(u,v)$ (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. While NodeArray (which is passed "by ref" via V_{arr}) is not useful for counting triangles, it 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).

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 ← MergeG, v, u, V_arr
 5:       m7[v] ← m7[v] + |Merged|
 6:       m7[u] ← m7[u] + |Merged|
 7:   for all v ∈ V(G) do
 8:     m7[v] ← m7[v]/2
 9: procedure Merge(G, v, u, NodeArray)
10:   for all w ∈ N(v) do
11:     NodeArray[w] ← 0
12:   for all w ∈ N(u) s.t. w ≠ v do
13:       NodeArray[w] ← 1
14:   for all w ∈ N(v), w ≠ u do
15:     if NodeArray[w] = 1 then
16:       NodeArray[w] ← 3
17:       list ← AppendToList list, w
18:    else
19:       NodeArray[w] ← 2
20:   return list
```

Theorem 1. *Algorithm 1 counts, for every node $v \in V$, the exact amount of occurrences of the triangle graphlet (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 than 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, since w is a neighbor of both u and v it is added to the merged list, and $\triangle(v,u,w)$ is counted for node v. Similarly there must be an iteration step for the edge (v,w) where $\triangle(v,u,w)$ is counted again, in contradiction to the assumption.

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: $O\left(\sum_{v\in V}\sum_{u\in N(v)} d\right) + O(n) = O(d \cdot |E|)$. □

2.2. Counting cycle with a chord

Algorithm 2 uses a cycle with a chord which can be described as two triangles with a joint edge. As in triangle count (Algorithm 1), this algorithm iterates over the edges of the graph, using the list of nodes obtained by Merge procedure (merged list) to find all the triangles. Each two nodes in the merged list create two triangles with the iterated edge as their joint edge.

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

Proof. $\forall e(u,v) \in E, \quad \forall w, \quad t \in (N(v) \cap N(u))(w \neq t \neq v \neq u)$ 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, $\forall w \in N(v) \cap N(u) \backslash \{u,v\}$, exactly $\binom{|N(v)\cap N(u)\setminus\{u,v,w\}|}{1}$ of these chordal cycles are adjacent to node w.

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

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

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

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

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

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 ← MergeG, 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)
```

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

Proof. By Theorem 1, counting triangles can be done correctly in time $O(d \cdot |E|)$. For each pair of nodes v, u we define $Tr(u)$ as the number of triangles adjacent to node u and $Tr_v(u)$ as the number of triangles adjacent to u that are not adjacent to v. The number of graphlets of type $m_{4.1}$ that are adjacent to u is given by $Tr(u) \cdot (|N(u)| - 2)$, for each node this value is computed exactly once (line 11). For each node w and triangle $\triangle uvw$ the number of graphlets of type $m_{4.2}$ which both are adjacent to w and contain the triangle $\triangle uvw$ can be defined by $|\{t | t \in N(u) \cup N(v),\ t \neq u,\ v, w\}| = |N(u)| - 2 + |N(v)| - 2$. Thus the total number of graphlets of type $m_{4.2}$ which are adjacent to w can be obtained by $\sum_{\{(u,v)|(u,v)\in E(G), w\in N(v)\cap N(u)\}} |N(u)| - 2 + |N(v)| - 2$, which is computed in line 9. Finally, the number of graphlets of type $m_{4.3}$ that are adjacent to v is given by

$$\sum_{u\in N(v)} Tr_v(u) = \sum_{u\in N(v)} (Tr(u) - (Tr(u) - Tr_v(u))) \quad (1)$$

$$= \sum_{u\in N(v)} Tr(u) - \sum_{u\in N(v)} (Tr(u) - Tr_v(u)) \quad (2)$$

$$= \left(\sum_{u\in N(v)} Tr(u)\right) - 2Tr(v). \quad (3)$$

Note that in the last transition of the equation we use the fact that the value $Tr(u) - Tr_v(u)$ defines the number of triangles adjacent to both nodes u and v. Therefore $\sum_{u\in N(v)}(Tr(u) - Tr_v(u))$ counts every triangle adjacent to node v exactly twice-once for each extra node in the triangle (each of the two neighbors of v in the triangle).

The runtime complexity analysis is divided into three parts: Counting triangles in $O(d|E|)$ time (line 2), first edge iteration (lines 3–9), and last edge iteration (lines 10–14). Using similar analysis as in Theorem 2, we get that the time complexity for the first iteration is $O(d|E|)$. Thus the total time complexity of Algorithm 3 is:

$$O(d|E|) + O(d|E|) + O(|E|) = O(d|E|). \qquad \square$$

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

Algorithm 3. Counting non-induced tailed triangles

```
1: procedure TailTriangleCount(G)
2:   TriangleCountG
3:   for all v ∈ V(G) do
4:     for all u ∈ N(v), v < u do
5:       Merged ← MergeG, 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]
```

2.4. Counting four nodal cliques

Counting cliques is done in a similar fashion to [8]. We show that the sorting of merged edge nodes neighborhoods done in [8] is redundant, thus lowering the runtime bound to $O(d|E| + |E|^2)$. The algorithm shares the same operational idea as that of the cycle with a chord count (Algorithm 2): Instead of only selecting two nodes from merged node list, the current algorithm also checks that the two selected nodes from the merge list are connected with an edge (lines 8–9 in Algorithm 4, see Fig. 2).

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

Proof. Correctness follows from [20]. Using computation similar to those shown in the proof of Theorem 1, we get the time complexity of Algorithm 4 to be:

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

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

$$= O(d|E|) + O(|E|^2). \qquad \square$$

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

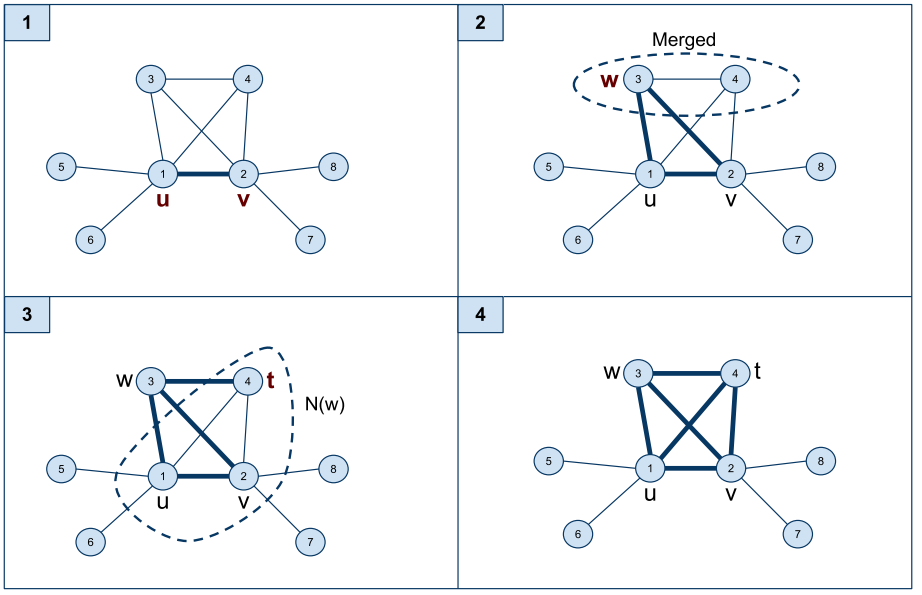

Fig. 2. Counting cliques. Edge $e(v,u)$ is first iterated after selecting node v and its neighboring node u (1). Node w is then selected from nodes u and v merged neighbors (2), after which node t is selected from w neighbors (3). Finally, since node t is also a neighbor of both u and v a clique is found (4).

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

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 ← MergeG, 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:            m1[v] ← m1[v] + 1
11:            m1[u] ← m1[u] + 1
12:      for all w ∈ N(v) ∪ N(u) do
13:        V_arr[w] ← 0
14:  for all v ∈ V(G) do
15:    m1[v] ← m1[v]/3
   (*) V_arr is set in the MERGE call (line 6). V_arr[t] = 3
   iff t is a neighbor of both v and u
```

2.5. Counting four nodal cycles

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

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 1, the time complexity of Algorithm 5 is:

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

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] = 1 or V_arr[t] = 3 then
10:            m3[v] ← m3[v] + 1
11:            m3[u] ← m3[u] + 1
12:      for all w ∈ N(v) ∪ N(u) do
13:        V_arr[w] ← 0
14:  for all v ∈ V(G) do
15:    m3[v] ← m3[v]/2
```

2.6. Counting four nodal path

Theorem 6. *Algorithm 6 finds, for every node $v \in V$, all non-induced occurrences of graphlet 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:

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

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 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 ← MergeG, v, u, V_arr
7:       m6.2[v]
8:
  ←m6.2[v] + (|N(v)\{u}| · |N(u)\{v}|) − |Merged|
9:       m6.2[u]
10:
  ←m6.2[u] + (|N(v)\{u}| · |N(u)\{v}|) − |Merged|
11:      for all w ∈ N(v)\{u} do
12:        if V_arr[w] = 3 then
13:          m6.1[w] ← m6.1[w] + |N(u)\{v}| − 1
14:        else
15:          m6.1[w] ← m6.1[w] + |N(u)\{v}|
16:      for all w ∈ N(u)\{v} do
17:        if V_arr[w] = 3 then
18:          m6.1[w] ← m6.1[w] + |N(v)\{u}| − 1
19:        else
20:          m6.1[w] ← m6.1[w] + |N(v)\{u}|
21:      for all w ∈ N(v) ∪ N(u) do
22:        V_arr[w] ← 0
```

2.7. Counting claws

Counting claws (a four node star) is done in the same way described in [8]. For each $v \in V$ the claw graphlet's count is obtained using the following equations:

$$m_{5.1}[v] = \binom{|N(v)|}{3},$$

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

3. Obtaining induced graphlets

The per-node and global non-induced graphlet 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 graphlet m in G, $I(m)$ the total number of induced appearances of graphlet m in graph G. In addition, for each node v in graph G, we denote $NI_v(m)$ by the number of non-induced position aware graphlets for which v participates in G, and $I_v(m)$ the number of induced position aware graphlets.

3.1. 4-Clique

Denote by m_1 the 4-clique graphlet.
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. $I_v(m_1) = NI_v(m_1)$).

3.2. 4-Cycle with a chord

Denote by m_2 the 4-cycle with a chord graphlet. A non-induced graphlet can be found only from an induced graphlet with a node degree vector which is not smaller for every component. Thus, m_2 can only be obtained by removing a single edge from m_1. 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 calculations we get, for each node $v \in V$:

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

3.3. 4-Cycle

Denote by m_3 the 4-cycle graphlet. According to the node degree vector of graphlet m_3, m_3 can be obtained from either m_1 or m_2. Removing any two matching edges from m_1 induces a cycle. Three such matchings exist. Removing the chord edge from m_2 induces a cycle. The 4-cycle induced count is obtained by:

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

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

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

3.4. Tailed triangles

Denote by m_4 the Tailed triangles graphlet. According to the node degree vector of graphlet m_4, m_4 can be obtained from either m_1 or 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:

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

$$I(m_4) = NI(m_4) - 4I(m_2) - \left(4 \cdot \binom{3}{2}\right) I(m_1)$$
$$= NI(m_4) - 4NI(m_2) + 12NI(m_1).$$

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

$$I_v(m_{4.1}) = NI_v(m_{4.1}) - 2I_v(m_{2.1}) - 3I_v(m_1),$$
$$I_v(m_{4.2}) = NI_v(m_{4.2}) - 2I_v(m_{2.1}) - 2I_v(m_{2.2}) - 6I_v(m_1),$$
$$I_v(m_{4.3}) = NI_v(m_{4.3}) - 2I_v(m_{2.2}) - 3I_v(m_1).$$

3.5. Claws

Denote by m_5 the claw graphlet. According to the node degree vector of graphlet m_5, m_5 is a subgraph isomorphism only of 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 $\geqslant 3$, 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 $\geqslant 3$. Removing all the edges that are not connected to this node induces a claw. So $I(m_5)$ can be deduced by:

$$I(m_5) = NI(m_5) - I(m_4) - 2I(m_2) - 4I(m_1)$$
$$= NI(m_5) - NI(m_4) + 2NI(m_2) - NI(m_1).$$

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

$$I_v(m_{5.1}) = NI_v(m_{5.1}) - I_v(m_{4.1}) - I_v(m_{2.1}) - I_v(m_1),$$
$$I_v(m_{5.2}) = NI_v(m_{5.2}) - I_v(m_{4.2}) - I_v(m_{4.3}) - I_v(m_{2.1}) - 2I_v(m_{2.2}) - 3I_v(m_1).$$

3.6. Simple path of length three

Denote by m_6 the simple path graphlet of length three (a path with three edges). According to the node degree vector of graphlet m_6, m_6 is a subgraph isomorphism in all graphlets other than m_5.

Removing an edge connecting a node of degree 3 and a node of degree 2 from m_4 induces a path. Two such edges exist. Removing any single edge from m_3 induces a path. Four such edges exist. Removing the chord and any other edge, or removing two matching edges that are not the chord from m_2 induces a path, giving a total of six possible paths. In m_1 every node pair is connected. Therefore, any permutation of the four 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:

$$I(m_6) = NI(m_6) - 2I(m_4) - 4I(m_3) - 6I(m_2) - 12I(m_1)$$
$$= NI(m_6) - 2NI(m_4) - 4NI(m_3) + 6NI(m_2) - 12NI(m_1).$$

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

$$I_v(m_{6.1}) = NI_v(m_{6.1}) - I_v(m_{4.2})2I_v(m_{4.3}) - 2I_v(m_3) - 2I_v(m_{2.1}) - 4I_v(m_{2.2}) - 6I_v(m_1),$$

$$I_v(m_{6.2}) = NI_v(m_{6.2}) - 2I_v(m_{4.1}) - I_v(m_{4.2}) - 2I_v(m_3) - 2I_v(m_{2.1}) - 4I_v(m_{2.2}) - 6I_v(m_1).$$

4. Evaluation

We implemented our algorithms, creating a tool named *RAGE* [2] and used it for evaluating their performance, scalability and applicability on the Internet Autonomous Systems (AS) graph.

The Internet is comprised of tens of thousands of ASes. The AS-graph is the coarsest view of the Internet, comprised of ASes as nodes. A link between two ASes exists when the two ASes have a peering agreement that allows them to directly exchange information (in the shape of IP packets) without the mediation of a third AS.

The AS connection graphs we used were obtained from the DIMES project [21], collected during August 2009, and from Dhamdhere et al. [16]. All runtime tests were run on an Intel Core 2 Quad Q8400 CPU machine having Windows XP as its operating system.

We compared RAGE with FANMOD [22], which is the previous fastest tool available for this task. We also compared our algorithm with our implementation of the approach suggested by Itzhack et al. [11], converted to count per node graphlets. We hereby refer to the latter as the IML approach.

The FANMOD tool can detect graphlets 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 [13]. FANMOD may also determine the frequency of graphlets in a user-specified number of random graphs, thereby detecting graphlets 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 AS graph collected over the month of August 2009 by the DIMES project [21], which contains 26,561 nodes and 92,584 edges, the run time for counting all the network graphlets was 40 min while FANMOD's sampling algorithm and FANMOD's full enumeration algorithm performed the same task at 2 h and at 48 h, respectively. We also used smaller scale-free graphs (i.e. sparse graphs that exhibit power law degree distribution), generated with the iNet Internet Topology Generator [23], in order to see how the three algorithms' run time depends on the graph size. Runtime results on the AS graph and iNet graphs, containing 5k nodes and 8k edges, 10k nodes and 20k edges, and 20k nodes and 52K edges are summarized in Table 1.

RAGE greatly outperforms FANMOD's accurate count (using full enumeration), running 2–5% of the runtime compared to FANMOD. Moreover RAGE also outperforms FANMOD's less accurate sampling algorithm. Finally, the table shows that RAGE achieves a speedup of 10 times over our implementation of the IML algorithms.

To further evaluate the runtime properties of RAGE, we ran our algorithms on three sets of random graphs, having 1k, 10k, and 100k nodes. Each graph set was built using the Erdös–Rényi model[24] with increasing number of edges as input, ranging from 5k to 2M edges. As the runtime

[2] RAGE is available online at www.eng.tau.ac.il/~shavitt/RAGE/Rage.htm

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

Table 1
Comparison of runtimes (in seconds) of RAGE and FANMOD using the synthetic and the AS graph.

Method	5k nodes	10k nodes	20k nodes	AS-Graph (26k nodes)
RAGE	11	64	720	2400
FANMOD (sampling)	57	210	1020	7200
FANMOD (full-enum)	420	2040	25200	172800
IML	140	1203	9850	25000

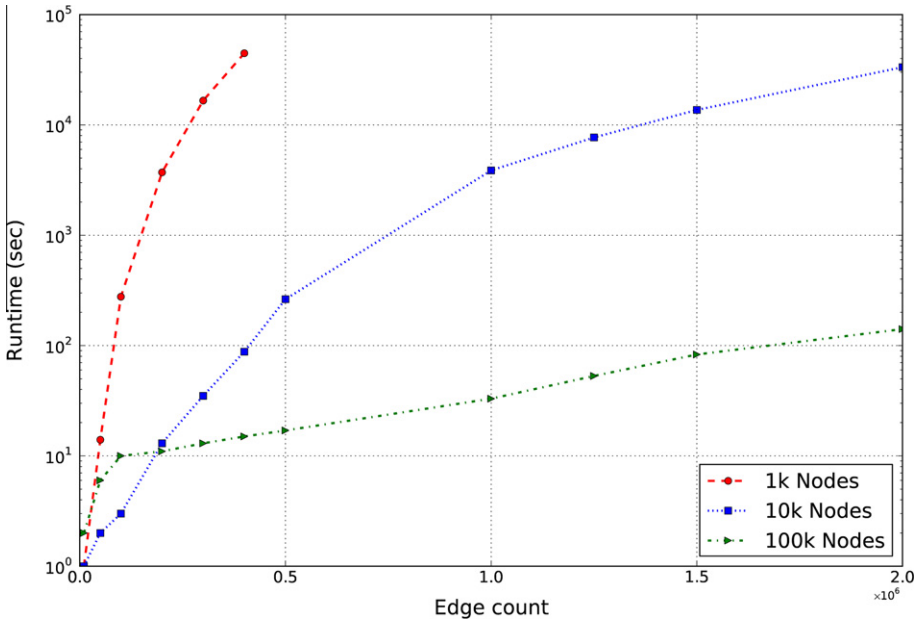

Fig. 3. RAGE runtime in seconds (log scale) on random Erdös–Rényi generated graphs with 1k, 10k, and 100k nodes and varying edge count. Results show RAGE preforms best on sparse graphs, even when run on large graphs.

results show (Fig. 3), the algorithm preforms best on sparse graphs: While the runtime for a 1k node graph having 300k edges is 16611 s (approximately 4.5 h) the runtime for the same edge count on a 10k node graph is 35 s and for a 100k node graph only 13 s.

In the above example, while the node count increases the edge count remains constant. This lowers the ratio between edges and nodes which in turn causes the graph to become more sparse and less inter-connected, e.g., the 1k node, 300k edges graph has an average degree of 600 and clustering coefficient of 0.6, whereas the 10k node graph has an average degree of 60 and a clustering coefficient of 0.67×10^{-2}.

5. Application example: classification of ASes

There are various types of ASes, often classified by their commercial role. Dhamdhere and Dovrolis [16] described five different types of ASes: Large Transit Providers (LTP) – international ISPs that have a large customer base and cover a wide geographical areas; Small Transit Providers (STP) – regional ISPs that provide both Internet access and transit services; Access or Hosting Providers (AHP) – ISPs which provide Internet access or hosting services both for residential users and for enterprise customers which do not have AS numbers; Content Providers (CP) – provide information for user consumption, e.g., Amazon or Google. Enterprise Customers (EC) – organizations, universities and companies that are more users of the Internet rather than providers of content, Internet access or transit services. EC ASes composes the main part of the AS set, being the long low degree tail with approximately 25,000 out of a total 27,000 ASes [16].

Classifying ASes is an important step towards better understanding of the Internet structure, as it helps position an AS in the commercial Internet ecosystem. However, the classification of the AS to their types is not a trivial task, more so when trying to distinguish between AHP and CP, mostly since there is an overlap in their behavioral patterns. Therefore, following [16], we refer to ASes of type AHP and CP as one type – AHCP. Note that it is impossible to distinguish between the different classes (except perhaps LTP) using only the undirected AS degree [16].

We applied RAGE on the AS graph, and used the resulting graphlet counts for the AS classification. Our premise is that these count results can be leveraged for AS classification, giving similar accuracy as previous classification methods [16,25], without the need of additional informa-

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

Table 2
AS classification accuracy per AS type, RAGE (with and without position-aware indication) compared to Dhamdhere et al. [16].

AS type	Accuracy (%)		
	RAGE (position aware)	RAGE (position blind)	[16]
EC	86	81	78
STP	84	86	86
AHCP	70	71	86
Total	79	79	83

tion such as the type of relationship between each connected AS pair.

Since only a few LTP exist (top 30° ASes) [16], they are all well known, and can be easily identified by their large degree. Thus, we focus on identifying STP, AHCP and EC ASes.

We computed the AS adjacent graphlets count of the AS graph from August 2009 obtained from Dhamdhere and Dovrolis [16]. The per-AS graphlet counts are normalized (according to the $L1$ norm) since we are interested in the relative count of each graphlet in comparison to the other graphlets.

We use the random forest classifier [26],[3] using the above counts, along with the AS degree, as features. To train the classifier we used a subset 3000 ASes, taken from Dhamdhere and Dovrolis AS classification [16]. Note that these classifications contain inherent noise. Testing and training of the classifier was done using a 10-fold cross validation method. We then further validated the resulting classification model using a separate set of 140 known ASes (40 STP, 40 AHCP and 60 EC). Similar to previous classification studies [16,25], we define our accuracy according to the percentage of correctly classified ASes out of the known AS set. We found our classification accuracy to be 79% (Table 2), a result close to those reported before [16,25]. We obtained similar results when testing other classifiers, such as SimplisticLogistic [28] logistic regression and J48 decision tree [29] aving a total accuracy of 79% and 76% respectively. Finally, using the same classification approach as above, we used position-blind graphlet count, counting all node adjacent graphlet without separating the graphlets into their position-aware classes. The classification results for this approach have not been significantly different from the one using the position-aware count (see Table 2).

6. Conclusion

Counting graphlets and motifs was shown in the last decade to be an important tool in analyzing complex networks in diverse fields. As our data collection tools become better, the networks we need to analyze are growing, and thus the task of graphlet counting becomes more challenging. In this paper we present efficient algorithms for counting graphlets of size 3 and 4, and prove their correctness and efficiency. Finally, we demonstrate the usage of graphlet enumeration for the problem of Internet AS classification.

[3] The random forest classifier was run using the WEKA [27] data mining software, configured to run with the classifier's default settings.

References

[1] N. Pržulj, D.G. Corneil, I. Jurisica, Modeling interactome: scale-free or geometric?, Bioinformatics 20 (2004) 3508–3515
[2] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (2002) 824–827.
[3] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, U. Alon, Superfamilies of evolved and designed networks, Science 303 (2004) 1538–1542.
[4] D. Feldman, Y. Shavitt, Automatic large scale generation of internet PoP level maps, in: GLOBECOM, 2008, pp. 2426–2431.
[5] L. Becchetti, P. Boldi, C. Castillo, 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.
[6] D. Hales, S. Arteconi, Motifs in evolving cooperative networks look like protein structure networks, The Journal of Networks and Heterogeneous Media 3 (2) (2008) 239–249 (special issue of ECCS'07).
[7] T. Milenkovic, N. Przulj, Uncovering biological network function via graphlet degree signatures, Social Networks 6 (2008) 257–273.
[8] M. Gonen, 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, 2009, pp. 13–24.
[9] M. Latapy, Main-memory triangle computations for very large (sparse (power-law)) graphs, Theoretical Computer Science 407 (1–3) (2008) 458–473.
[10] V. Batagelj, A. Mrvar, A subquadratic triad census algorithm for large sparse networks with small maximum degree, Social Networks 23 (2001) 237–243.
[11] R. Itzhack, Y. Mogilevski, Y. Louzoun, An optimal algorithm for counting network motifs, Physica A 381 (2007) 482–490.
[12] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, K. Borgwardt, Efficient graphlet kernels for large graph comparison, in: 12th International Conference on Artificial Intelligence and Statistics (AISTATS), Society for Artificial Intelligence and Statistics, Clearwater Beach, Fl, USA, April 16–18, 2009, 2009.
[13] S. Wernicke, Efficient detection of network motifs, IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 3 (2006) 347–359.
[14] A. Stoica, C. Prieur, Structure of neighborhoods in a large social network, in: International Conference on Computational Science and Engineering (SocialCom), 2009.
[15] M. Gonen, D. Ron, Y. Shavitt, Counting stars and other small subgraphs in sublinear time, SIAM Journal on Discrete Mathematics 25 (3) (2011) 1365–1411.
[16] A. Dhamdhere, C. Dovrolis, Ten years in the evolution of the internet ecosystem, in: Proceedings of the 8th ACM SIGCOMM Conference on Internet measurement, IMC'08, ACM, 2008, pp. 183–196.
[17] L. Gao, On inferring autonomous system relationships in the Internet, IEEE/ACM Transactions on Networking 9 (6) (2001) 733–745.
[18] Y. Shavitt, E. Shir, U. Weinsberg, Near-deterministic inference of AS relationships, in: ConTEL, 2009.
[19] R. Cohen, D. Raz, The Internet dark matter – on the missing links in the AS connectivity map, in: IEEE Infocom 2006, Barcelona, Spain, 2006.
[20] M. Gonen, Y. Shavitt, Approximating the number of network motifs, Internet Mathematics 6 (3) (2009) 349–372.
[21] Y. Shavitt, E. Shir, DIMES: Let the internet measure itself, ACM SIGCOMM Computer Communication Review 35 (2005) 71–74.
[22] S. Wernicke, F. Rasche, FANMOD: a tool for fast network motif detection, Bioinformatics 22 (2006) 1152–1153.
[23] C. Jin, C.J. Qian, S. Jamin, Inet: Internet topology generator, <http://www.topology.eecs.umich.edu/inet>, 2000.
[24] P. Erdos, A. Renyi, On random graphs I, Publicationes Mathematicae (Debrecen) 6 (1959) 290–297.
[25] X. Dimitropoulos, D. Krioukov, G. Riley, K. Claffy, Revealing the autonomous system taxonomy: the machine learning approach, in: Passive and Active Measurement (PAM) Workshop, 2006.
[26] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32.
[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, SIGKDD Explorer Newsletter 11 (2009) 10–18.
[28] M. Sumner, E. Frank, M. Hall, Speeding up logistic model tree induction, in: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Springer, 2005, pp. 675–683.
[29] R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman, Publishers, San Mateo, CA, 1993.

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019

Dror Marcus received his B.Sc degree from The Open University of Israel in 2009 (cum laude). He is currently a M.Sc candidate at Tel Aviv University, focusing on the Internet mapping, complex networks and large scale graphs.

Yuval Shavitt has a D.Sc. in electrical engineering from the Technion, Haifa, Israel, and is currently a faculty member in the School of Electrical Engineering at Tel-Aviv University, Israel. His research interests include Internet measurements, mapping, and characterization; and data mining peer-to-peer networks.

Please cite this article in press as: D. Marcus, Y. Shavitt, RAGE – A rapid graphlet enumerator for large networks, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.08.019