Complex Network Analysis using Parallel Approximate Motif Counting

George M. Slota Kamesh Madduri
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA, USA
Email: gms5016@psu.edu, madduri@cse.psu.edu

***Abstract*—Subgraph counting forms the basis of many complex network analysis metrics, including motif and anti-motif finding, relative graphlet frequency distance, and graphlet degree distribution agreements. Determining exact subgraph counts is computationally very expensive. In recent work, we present FASCIA, a shared-memory parallel algorithm and implementation for approximate subgraph counting. FASCIA uses a dynamic programming-based approach and is significantly faster than exhaustive enumeration, while generating high-quality approximations of subgraph counts. However, the memory usage of the dynamic programming step prohibits us from applying FASCIA to very large graphs. In this paper, we introduce a distributed-memory parallelization of FASCIA by partitioning the graph and the dynamic programming table. We discuss a new collective communication scheme to make the dynamic programming step memory-efficient. These optimizations enable scaling to much larger networks than before. We also present a simple parallelization strategy for distributed subgraph counting on smaller networks. The new additions let us use subgraph counts as graph signatures for a large network collection, and we analyze this collection using various subgraph count-based graph analytics.**

I. INTRODUCTION

Subgraph counting is a computationally intensive problem. A naïve algorithm, which exhaustively enumerates all vertices reachable in k hops from a vertex, runs in $O(n^k)$ time, where n is the number of vertices in the network and k is the number of vertices in the subgraph. For large networks, this imposes considerable constraints on the sizes of possible subgraphs that can be counted. Thus, there has been a lot of recent work on approximation algorithms. Approaches are either based on sampling or on exploiting network topology. Sampling-based methods analyze a subset of the network and extrapolate counts based on the observed occurrences and network properties [1]–[3], while the other class of methods impose some constraint on the network or transform the network, so that the possible search space [4] is restricted. The *color-coding*-based counting approach belongs to the second category and forms the basis of our current work.

The color-coding technique [5] can be used to determine counts of non-induced tree-structured templates in $O(m \cdot 2^k \cdot e^k \cdot \frac{\log 1/\delta}{\epsilon^2})$ time, where m is the number of edges in the graph, and δ and ϵ are confidence and error parameters, respectively (i.e., the estimated count is within $c(1 \pm \epsilon)$ with probability $(1 - 2\delta)$, where c is the exact count). The algorithm can be informally stated as follows: every vertex in the network is randomly colored with one of k possible colors. The number of *colorful embeddings* is then counted, where colorful in this context means that each vertex in the template embedding has a distinct color. The total embedding count is then scaled by the probability that the template is colorful, to give an approximation for the total number of possible embeddings. The dynamic programming algorithm used to count colorful embeddings avoids the prohibitive $O(n^k)$ bound seen in exhaustive search.

In previous work, we introduced FASCIA [6], a fast shared-memory parallel implementation of the color-coding algorithm for approximate treelet counting. The current work builds on our prior work with distributed-memory parallelization. Multi-node strong scaling lets us process larger graphs and template sizes, and we achieve very fast and accurate counts for large-scale graphs. Distributed FASCIA necessitates a new distributed representation of the dynamic programming table data structure that is central to the count approximations. The primary contribution of this work is presenting how this data structure can be managed in a memory-efficient manner in a distributed setting.

Subgraph counts in isolation give little or no insight into the topological structure of a complex network. However, a number of analytics introduced over the past decade use subgraph counts as a means for identifying latent structural patterns within networks, for ascertaining possible variations between neighborhoods of individual vertices within a single network, and for comparative analysis of networks. We give an overview of some of these subgraph-based analytics in the next section. Our second contribution is an extensive comparative analysis of real-world networks using subgraph count-based graph analytics. The efficient distributed-memory parallelization of FASCIA enables these computations by improving time to solution for distributed counting on small networks, and decreasing the memory requirements for large networks.

II. BACKGROUND

A. Motif Finding

Network motifs are defined as subgraphs that occur more frequently in a network than would be expected by random

chance. There has been considerable study of network motifs in bioinformatics [1], [2], [4], [7]–[10], usually to discern structurally-significant characteristics in protein-protein interaction and related biological networks. Alon et al. [9] developed a color-coding based approach for the purpose of determining biological network motifs. Recent work has also focused on applying subgraph counting and network motif finding methods for social, informational, and other networks [11]–[13].

Motif finding is typically carried out by finding complete subgraph counts for all possible templates of up to a certain size. The relative frequency of each subgraph is determined by scaling the counts by either the total count or the average count. We will demonstrate the resilience of treelet counts for motif finding with noisy or incomplete networks later in this paper.

B. Graphlets

Graphlets are formally defined as small undirected subgraphs between two and five vertices in size. Prior work by Pržulj et al. [14]–[17] extensively studies graphlets in the context of biological networks. Pržulj et al. also identified all possible discrete *orbits* within each graphlet. Orbits in this context refer to distinct automorphic vertices in the subgraph; i.e., it can be useful to explicitly differentiate between a vertex in the center of a star versus a vertex on one of the leaves.

Graphlets are also considered to only be *induced* subgraph occurrences. Induced occurrences imply that for every subgraph embedding of a template in a network, edges can exist between vertices in the subgraph if and only if they exist in the template. Most prior work focuses on induced occurrences, as they are computationally less expensive to count. However, Alon et al. [9] argue that non-induced subgraph counts can provide a more accurate analysis of noisy real-world networks.

The graphlet frequency distance (GFD) was proposed by Pržulj et. al [15] as a global comparative measure based on the local structural characteristics of different networks. To calculate the GFD, counts of all $i = 1, 2, \ldots, 29$ graphlets in network G are determined as $C_i(G)$. Each of the 29 counts are then log-scaled by the total count of all graphlets, with the distance value between networks G and H being the sum of the absolute differences of all scaled counts.

$$S_i(G) = -\log\left(\frac{C_i(G)}{\sum_{i=1}^{29} C_i(G)}\right), D(G,H) = \sum_{i=1}^{29} |S_i(G) - S_i(H)|$$

To avoid possible confusion created by using the term *graphlet* outside of its standard context (subgraphs of 2 to 5 vertices), as well as the generic term subgraph, this paper will use the term *treelet* in their place where appropriate (for instance, in this paper, we compute treelet frequency distances).

C. Clustering

Bordino et al. [18] demonstrate that one can use the relative frequency of subgraphs within networks to distinguish and cluster different networks. Using the relative frequencies of undirected subgraphs up to four vertices and other topological properties such as in-degree, out-degree, and PageRank as representative features for a network, they show up to 75% clustering accuracy for networks chosen from seven distinct categories. Using directed edges and 284 features in total, they achieved just over 90% clustering accuracy. Recent work by Rahman et al. [3] implements an approximate graphlet counting algorithm and uses graphlet counts as a vector to cluster various network types. We perform a similar clustering for a larger group of networks using treelet counts.

III. Fascia Distributed-memory Parallelization

We begin this section with an overview of the approximate treelet counting method. We then describe how we parallelize this computation in a fast and memory-efficient manner. An in-depth description of color-coding, the dynamic programming formulation it uses, and its shared-memory parallelization is available in other papers [6], [12].

A. Algorithm Overview

The color coding-based method is constituted by several steps (see Algorithm 1). Initially, the input template is partitioned into subtemplates by following an iterative method of *single edge cuts*. Next, the number of iterations to be performed is determined based on desired approximation quality input parameters. For each iteration, we randomly color the graph with at least k colors, where k is the number of vertices in the template. We then begin with the last subtemplate created during our partitioning procedure and perform a bottom-up dynamic programming scheme, tracing through the subtemplates in the reverse order they were created during partitioning.

Algorithm 1 Subgraph counting using color coding.

1: Partition input template T (k vertices) into subtemplates S_i using *single edge cuts*.
2: Determine $Niter \approx \frac{e^k \log 1/\delta}{\epsilon^2}$, the number of iterations to execute. δ and ϵ are input parameters that control approximation quality.
3: **for** $it = 1$ to $Niter$ **do**
4: Randomly assign to each vertex v in graph G a color between 0 and $k-1$.
5: Use a dynamic programming scheme to count *colorful* non-induced occurrences of T.
6: Take average of all $Niter$ counts to be final count.

We are going to ignore several nuances about the dynamic phase of the algorithm for now. The general idea is given

in Algorithm 2. We can trace backwards up a tree that represents the partitioning of the original template. The count for each subtemplate is dependent only on the counts for its children subtemplates. Specifically, the count for a given subtemplate S_i with a given color mapping I at vertex v, is the sum over all products of the counts of its children created during partitioning with mappings created by distributing the colors of I, having one child subtemplate rooted at vertex v and the other child subtemplate rooted at all vertices u in the neighborhood of v. Through this approach, we can trace all the way back to our original template T. Our count estimate for this iteration is the sum of all counts of T, over all vertices v, over all possible color sets I. The final count estimate is then the average of all iteration counts, scaled by the probability that any given embedding of T in the graph is colorful.

Algorithm 2 FASCIA dynamic programming routine with distributed counting.

```
for it = 1 to Niter do in parallel
    Color G(V, E) with k colors
    Initialize 3D count table
    for all S_i in reverse order of partitioning do
        for all v ∈ V do in parallel        ▷ thread-level
            Update count table for template S_i
            using child subtemplate counts
```

B. Distributed Counting

There are several avenues for parallelization in this method. For shared memory parallelization of FASCIA and for very small graphs, we partition the outer loop across threads (i.e, each thread performs N_{iter}/p iterations with p-way threading). We refer to this as the outer-loop parallel approach. This idea can be extended to a distributed setting, and the iteration chunks of the outer loop can be assigned to different tasks. With this setting, we can also perform the inner loop (counts for all vertices) in parallel using multithreading. We refer to this hybrid parallelization strategy as *distributed counting*. As the dynamic programming table storing counts scales as $n \cdot \binom{k}{k/2}$, where n is the number of vertices in the graph and k is the number of vertices in the template, memory use is an issue with this approach for large graphs and templates.

C. Partitioned Counting and Memory Optimizations

For modest-sized graphs (more than 2 million vertices) and large templates ($k > 10$), memory utilization quickly becomes problematic with distributed or shared-memory counting. This is an issue even when using various previously-introduced memory-saving techniques such as a fast hashing scheme or vertex-based table initializations [6]. We have therefore also implemented a distributed

Algorithm 3 FASCIA Partitioned Counting Approach.

```
for it = 1 to Niter do
    Color G(V, E) with k colors
    for all S_i in reverse order of partitioning do
        Init Table_{i,d} for V_d (vertex partition on task d)
        for all v ∈ V_d do              ▷ Thread-level parallelism
            for all c ∈ C_i do
                Compute all Count_{S_i,c,v}
        N_d, I_d, B_d ← Compress(Table_{i,d})
        for all d = 1 to NumTasks do
            N_i, I_i, B_i ← Bcast(N_d, I_d, B_d)
    Count_d += Σ_v^{V_d} Σ_c^{C_T} Count_{T,c,v}
Count ← Reduce(Count_d)
Scale Count based on Niter and colorful embed prob.
```

graph partitioning-based approach, where each task performs counts for a subset of all $v \in V$.

In this approach, each task is required to have a full table for each child subtemplate, while only needing to create the table for the current subtemplate for the task's subset of vertices. At the end of each iteration, unnecessary child tables are deleted. The table subsets for the current subtemplate are then distributed to all nodes in the cluster. Due to the large memory footprint of these arrays, this approach has substantial data transfer overhead. We have therefore reduced total data transfer costs by utilizing a compressed sparse row (CSR) format for storing the counts. This allows for fast distributed counting on graphs much larger than previously possible.

The CSR format is a commonly-used sparse matrix storage format consisting of three arrays. One array stores all values matrix non-zero values in row-major ordering. This array would be structured as $[(\text{row}_1)(\text{row}_2)\cdots(\text{row}_n)]$, where (row_i) is a list of all nonzero values in that row. A second array of the same length as this first array is used to hold the column indexes at each of the nonzero values stored in the first array. The final array is of length n, or the number of rows, and it holds indexes to the start of the sequence of values for each row.

By considering a table for each discrete subtemplate S_i as a matrix of size $n \times C_i$, where n is the number of vertices in G and C_i is the number of possible color sets for S_i, we can use the CSR format to our table in order to compress it and reduce data transfer volume. The first array stores all non-zero counts for all vertices and color mappings. The second array is the color mapping indexes for each count value, as computed using a combinatorial number system approach (see [6]). The final array denotes the indexes for the start of count values for each vertex.

Because the lookup for any specific (x, y) index can be slow using this format and the color-coding approach requires a significant number of such lookups, we ideally want to decompress values for direct access. However,

in order to further minimize memory footprint, we only decompress when the values are needed to compute the count of the new parent subtemplate.

Algorithm 3 details the pseudocode. For every S_i, we only initialize our table for the task's specific subset of vertices V_d. We compute all the counts for the subset of vertices ($v \in V_d$) for all possible colorsets ($c \in C_i$) for the current subtemplate. We then compress the table into CSR format, with N_d denoting the array of count values, I_d the array of color mapping indexes, and B_d containing the begin indexes for each vertex. We distribute the counts among all d tasks, so that each task now has the child counts required to compute counts for the new parent template.

At the completion of each iteration, each task computes the final count for the template for its subset of vertices. We simply keep a running sum of the counts for each task for each iteration. After all iterations are completed, we reduce the sums from all tasks, scale them by the number of iterations and probability that the template is colorful, and then will have produced our final count estimate. It should be noted that no additional approximations are introduced during parallelization. There would still be minor variation in the serial and parallel approaches due to the random coloring and rounding error incurred when performing floating-point arithmetic.

IV. Experimental Setup

We performed experiments on various parallel platforms and interactive systems, including Gordon at the San Diego Supercomputer Center, Stampede at the Texas Advanced Computing Center, and the CyberSTAR and Hammer systems at Penn State University. For experiments where execution times are reported, we used the Compton system at Sandia National Laboratories. Each Compton node has 2 Intel Xeon E5-2670 (Sandy Bridge) processors with 64 GB memory. Code was compiled with the Intel C compiler `icc` using `-openmp` and `-O3` flags.

A. Networks Analyzed

We analyzed networks from eleven different categories, obtained from many different sources [19]–[21] (see Table I). These include collaboration networks from Arvix and the DBLP Computer Science Bibliography [22]–[24], communication networks of emails and Facebook wall posts [22], [25], [26], four Erdős-Rényi $G(n,p)$ random graphs, snapshots of the peer-to-peer Gnutella network at various times [22], [27], four biological protein-protein interaction (PPI) networks [28], five road networks [21], [23], four random scale-free Barabăsi-Albert networks [29], six social networks of online relationships of various types [20], [30], [31], four random small-world graphs [32], and four web crawls of various universities and Google [23]. The social network of Orkut [24] was additionally analyzed for parallel performance, as well as a synthetic social contact network of Portland [33]. In total, over 50 different networks were considered.

Network Type	Count	n ($\times 10^3$) min	n ($\times 10^3$) max	m ($\times 10^3$) min	m ($\times 10^3$) max
Collaboration	6	26	425	14	1050
Communication	4	30	63	87	855
$G(n,p)$	4	10	100	100	1000
Peer-to-peer	9	6	63	9.7	77
Bio PPI	4	0.7	22	1.3	22
Road	5	440	1970	530	2800
Scale-free	4	10	100	100	1000
Social	6	60	150	214	5400
Small-world	4	10	100	100	1000
Web Crawl	4	280	875	761	3900
Orkut	-	3100	-	117000	-
Portland	-	1620	-	31000	-

Table I
NETWORKS ANALYZED IN THIS STUDY: CATEGORIES, COUNTS, SIZES.

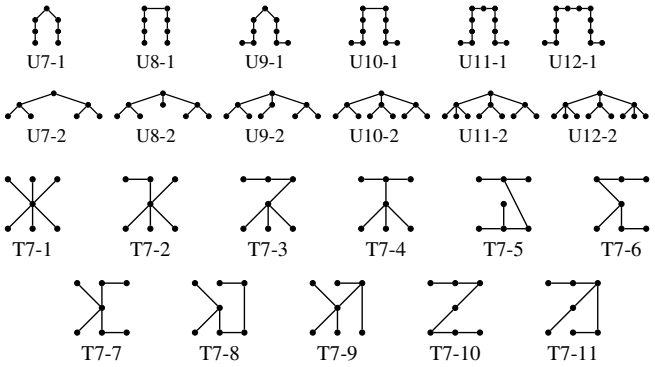

Figure 1. Templates structures used in MPI scaling, CSR memory usage, and for network noise analyses.

All graphs considered are undirected with multiple edges and self loops removed. This mainly only affected the structure of the communication and social networks. The color-coding method can be applied to directed graphs as well. However, the current implementation is unable to do so and this extension is left for future work.

B. Templates Analyzed

All tree-structured templates between three and nine nodes were considered in our analysis. There is 1 template with 3 nodes, 2 templates with 4 nodes, 3 templates with 5 nodes, 6 templates with 6 nodes, 11 templates with 7 nodes, 23 templates with 8 nodes, and 47 templates with 9 nodes. The templates were created by parsing data output by an online graph generator [34]. In total, 92 subgraphs were considered when calculating motifs and the treelet frequency distances. For analyzing our distributed algorithm's performance with regards to scaling and memory use, we use two templates of 7, 8, 9, 10, 11, and 12 vertices. Figure 1 shows the 11 different 7-vertex templates used in Section V-E.

V. Performance Results and Network Analysis

In this section, we present performance results achieved with our distributed and partitioned implementations of FASCIA. We will demonstrate strong scaling along with reduction in data transferred due to use of the CSR communication layout. Additionally, we also use color-coding treelet counting in several network analysis methods: motif finding, network type clustering, as well as relative treelet frequencies. The latter uses modifications to existing techniques that commonly use the smaller graphlets, with the changes to calculation methodologies noted where appropriate. We will also study robustness of using treelet counts for analyzing noisy or incomplete networks by perturbing a subset of vertices/edges.

A. Parallel Performance

Figure 2 (top) gives strong scaling results of partitioned treelet counting by FASCIA for the U12-1 and U12-2 templates on the Orkut network. A single iteration of counting is performed using the partitioning methodology previously explained. Running times are reported for up to 15 nodes, or 240 total cores. As can be seen from Figure 2, an approximate speedup of 3.5× is observed on the U12-1 template, in comparison to the shared-memory multithreaded single-node execution. With U12-2, a more complex template, we achieve a 7× speedup on 15 nodes.

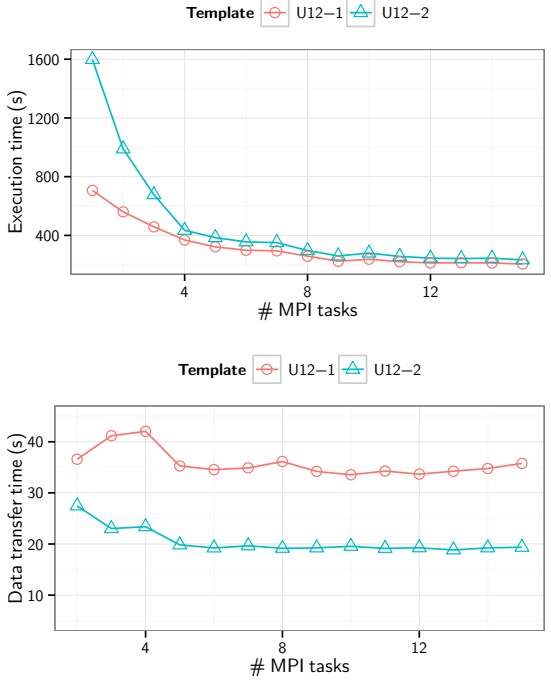

Figure 2. Parallel scaling of the partitioned counting approach. U12-1 and U12-2 templates, Orkut network; Total execution time (top) and total data transfer time (bottom).

Figure 2 (bottom) gives the cumulative time spent in data transfer for both the U12-1 and U12-2 templates on 2 to 15 nodes. This includes both the time needed to compress the data and the time spent in MPI collective communication routines. We observe that compression takes about a quarter of the total time. Although the overall running time for U12-2 is higher than U12-1, the transfer time for U12-2 is lower because of the smaller table size.

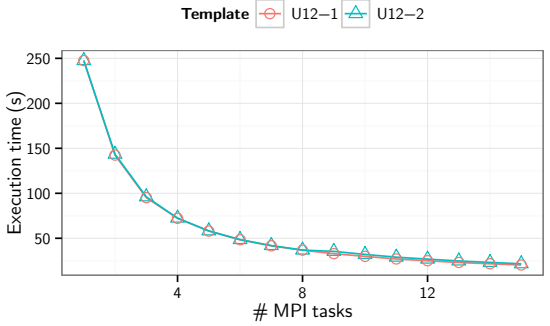

Figure 3. Parallel scaling of 16,000 iterations for counting U12-1 and U12-2 templates on the *C. elegans* PPI network.

We do not observe good data transfer time scaling due to two primary reasons. Firstly, there is a tradeoff between time for compression and communication. The larger the number of nodes, the greater the amount of data to be broadcast. Secondly, the performance of the MPI Broadcast on our test system does not scale linearly with transfer size, for large arrays and for a relatively small number of nodes (i.e., a broadcast of d size to n nodes takes longer to complete than a broadcast of $\frac{d}{2}$ size to $2 * n$ nodes). This mostly accounts for the decrease in total transfer times for both templates from about 2 to 6 nodes.

As the total communication time only accounts for a small fraction of total running time, we do not yet consider it to be the primary bottleneck for scaling in our approach. Analysis of computation time differences between nodes indicates that there is computational load imbalance using a naïve vertex partitioning method. Reordering of vertices or a distributed dynamic load balancing scheme would likely alleviate this bottleneck. We will investigate this in future work.

Figures 3 gives the scaling for $16,000$ total iterations of counting templates U12-1 and U12-2 on the *C. elegans* PPI network doing full outer-loop parallelization and distribution. For this instance, we are concurrently performing color-coding iterations on 240 cores (15 MPI tasks, 16 threads per node). Due to the minimal communication overhead required in performing this count, parallel speedup is about 12.5× for both templates. We can thus utilize a combination of both the partitioning and outer-loop distributed parallelization for strong scaling on large networks.

B. CSR Bandwidth Reduction

The large size of the dynamic programming table can make partitioned counts accumulation difficult, due to the

considerable amount of data transferred at each step of the algorithm. The overall data transfer volumes can be upwards of dozens of gigabytes for a single color-coding iteration, even for modestly-sized graphs with millions of vertices and a relatively small value of k. To this effect, it is important to try and minimize these costs in order to improve execution times.

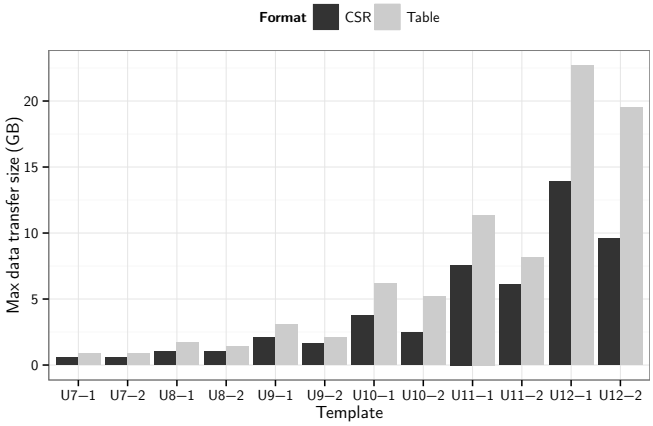

Figure 4. Maximum data transfer required on Orkut using the U12-1 and U12-2 templates with baseline table and CSR storage.

Figure 4 gives the maximum data transfer during a single iteration for all of the UX-1 and UX-2 templates from 7 to 12 vertices on the Orkut network. On average, we observe a consistent 35% reduction in data transferred, with a reduction of over 50% occurring for select templates. Also, note that the Orkut network has a relatively high average vertex degree, which results in a considerably higher rate of template embeddings for any given vertex relative to a network that is not nearly as dense. In general, the total time required for CSR compression and decompression is much lower than the time required to broadcast the compressed table.

C. Relative Treelet Frequency Distances

The relative treelet frequency distances were calculated for all networks and groups using all treelets between four and nine vertices in size. We ran each count for 1000 iterations to minimize error in the estimates. The methodology used to calculate these distances is slightly different from the approach described in the background Section. Instead of taking the logarithm of the total count over all subgraphs, we scale values by the total counts for the specific treelet size. This is done because, even using a log scale, the differences in count magnitudes between four and nine-node treelets is far too large to scale them by the same denominator.

Figure 5 demonstrates the results of these calculations on a heatmap. For visualization purposes, the final distance values are log-scaled. Red represents low disagreement while orange, yellow, and finally white show increasingly higher disagreements. Each row-column coordinate represents the distance calculated between the networks (names on the right hand side and bottom). Networks are ordered by group.

A number of observations are possible by observing Figure 5. Several groups show very minimal intra-group disagreement, including the road networks, the small world and $G(n, p)$ random graphs, as well as the collaboration and peer-to-peer networks in a lesser extent. The protein-protein interaction networks show agreement among the unicellular organisms (*E. coli*, *H. pylori*, *S. cerevisiae*), but low agreement with the multicellular organism (*C. elegans*).

The peer-to-peer results are interesting, in that there appears to be two distinct subgroups with high intra-group agreement. Each distinct network is simply a snapshot of part of the same larger network at varying points in time. This might highlight the highly dynamic and fluid nature of peer-to-peer networks, as connections are being constantly made and broken as new content is released and shared. Or it might just be an artifact due to noise and the relatively small sample of the network that was taken at each date.

There is also some agreement between the random small-world networks with the peer-to-peer and collaboration networks, demonstrating a correspondence to the small-world phenomenon that is known to exist in these networks. However, there is no such correlation with the social network crawls, which is surprising and likely suggests that there are other network measures necessary to take into account when determining network similarity beyond subgraph frequency, or that the graph generator parameters or algorithm used to create the small-world graphs need further tuning.

We also created a heatmap with treelet degree distribution agreements between all networks, using a similar calculation methodology based on graphlet degree distribution agreements work by Pržulj [14]. However, due to space limitations, we omit these results. They are available in an extended version of this paper [35].

D. Clustering Using Treelet Frequency Counts

We also examine whether we can use the treelet occurrence frequencies to cluster networks into categories. The relative frequencies of all 4-9 vertex tree-structured subgraphs for the same ten network groups were considered to be a feature vector, and we used the the k-means and E-M clustering algorithms with the number of clusters set to 10. Approximately 70% and 75% average clustering accuracies were produced using the k-means and E-M algorithms, respectively. This is about the same accuracy as reported by Bordino et al. on undirected graphs (75%). However, they only considered seven network groups at a time, and used other topological features beyond subgraph frequencies. This result indicates that large treelet frequencies are useful features to consider when attempting to classify networks of various types.

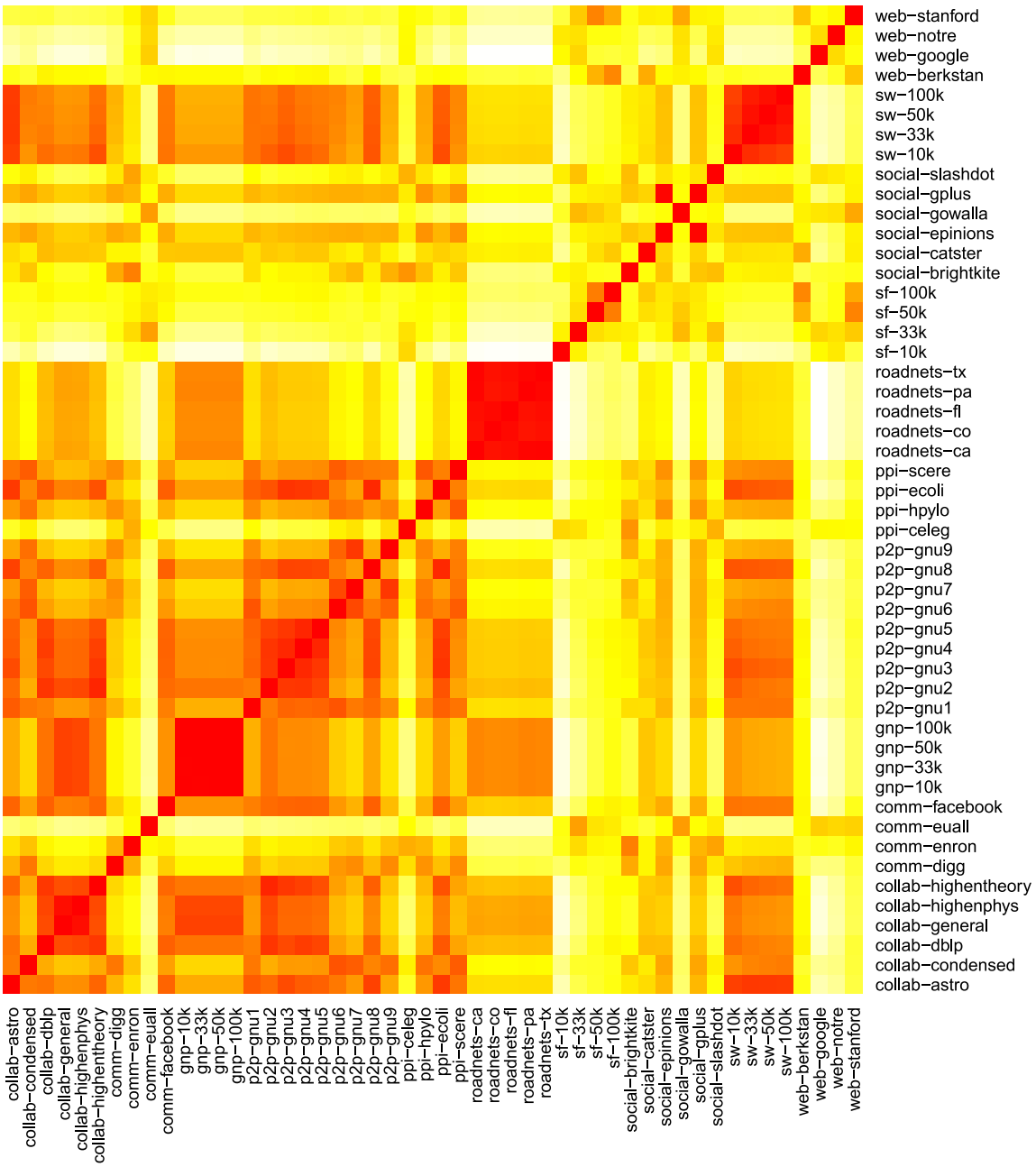

Figure 5. Treelet frequency distances between all tested networks. Darker implies a lower distance or higher similarity.

E. Node and Edge Deletion and Edge Rewiring

As most real-world networks are incomplete, noisy, and dynamically evolving, the utility of treelet count for network analysis should be examined under these considerations. We select four networks and introduce varying amounts and types of network alterations. A 100K vertex $G(n, p)$ graph, the Notre Dame web crawl, the Slashdot social network, and one of the Gnutella peer-to-peer snapshots were all modified by deleting vertices, deleting edges, and randomly rewiring edges. 5%, 10%, 20%, 50%, and 75% modifications of total vertices or edges were performed. The differences in treelet frequency distance between the modified and original network were then noted. As before, 1000 iterations were performed to retrieve the counts for all networks.

Figure 6 gives the results of vertex deletion on motif plots for all eleven different seven vertex templates (T7-1 to T7-11) on the four aforementioned networks. Vertex deletion has the most pronounced effect on the p2p network and the least effect on the gnp100k network. It is surprising that for some templates, the scaled counts are accurate even with

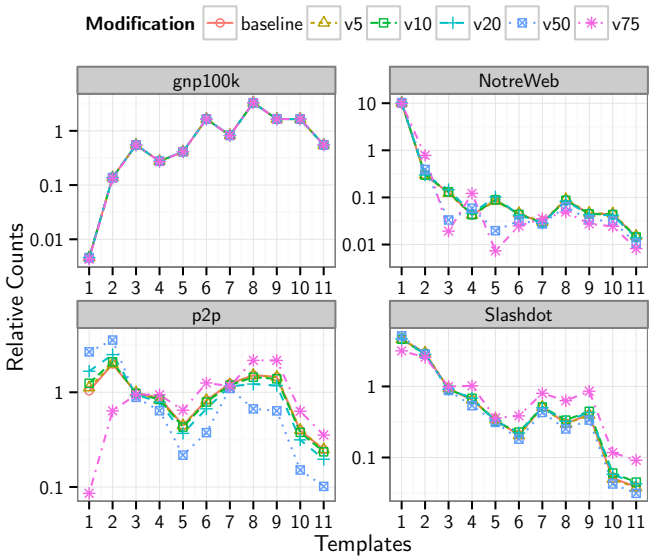

Figure 6. Treelet counts after 5%, 10%, 20%, 50%, and 75% **vertices** are deleted.

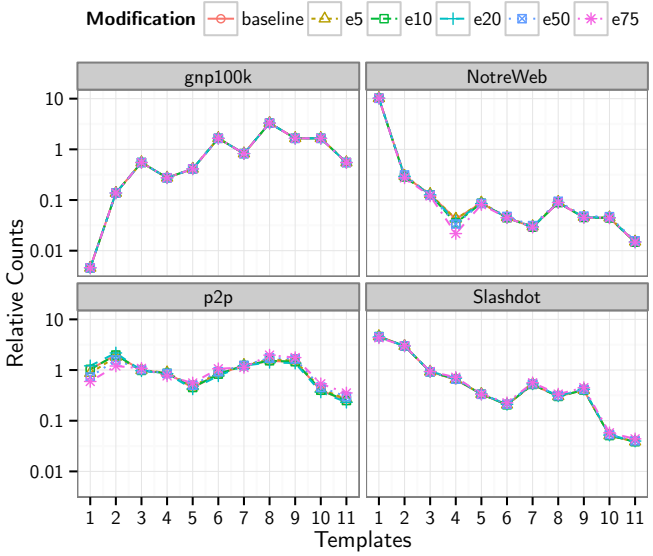

Figure 7. Treelet counts after 5%, 10%, 20%, 50%, and 75% **edges** are deleted.

75% of the vertices deleted. This suggests that combining sampling-based schemes with color-coding might be quite effective to obtain counts for some templates.

Using the methodology to calculate treelet frequency distances (as originally presented with log scaling), we calculate disagreement values for all networks before and after perturbation. The maximal distance between the original and modified counts for any network was with the Notre Dame web crawl, having a disagreement value of 4.1. With relative count scaling, this can be contrasted to the average disagreement value between all baseline networks, which is 9.2. The minimal disagreement for the modified networks was with 100K vertex $G(n, p)$ graph, having an absolute disagreement of only 0.6. These results support the assertion that using treelet counts for network analysis can be useful even with incomplete networks.

Figure 7 gives results obtained with random edge removal from the networks. Edge removal has a lower impact on treelet counts as expected. The calculated maximal disagreement was with the Gnutella peer-to-peer graph, having a disagreement value of 1.2 with 75% edges removed. All other values were well below 1. This lends further credibility to the use of treelet analysis on networks with a high proportion of known vertices, but a lower confidence in known edges, such as protein interaction networks in computational biology.

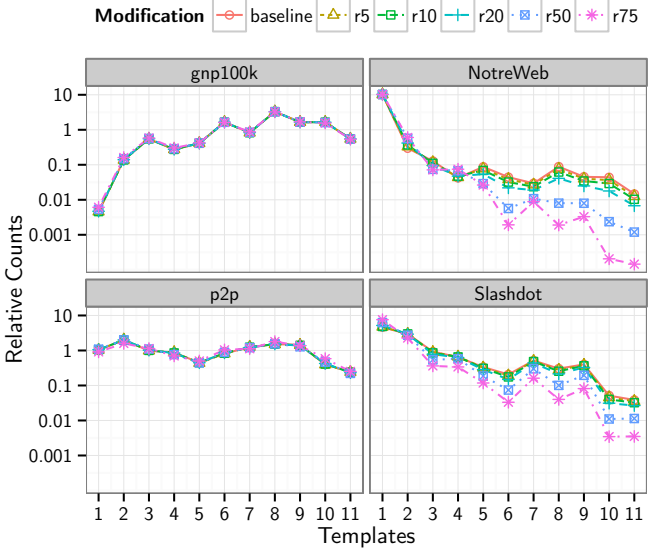

Figure 8. Subgraph counts after 5%, 10%, 20%, 50%, and 75% **edges are rewired**.

We performed another study to observe the effect of randomly rewiring a proportion of edges within the network. These results are shown in Figure 8. Uninterestingly, the treelet counts on the random network once again show minimal change, along with the peer-to-peer network. However, the treelet counts on the social network and web crawl are quite different. A high degree of random rewiring seems to have a greater affect on the local topology in the social network and the web crawl.

The disagreement values calculated for Slashdot and the Notre Dame web crawl are relatively high for 75% rewiring (6.6 and 10.4, respectively), and the values calculated at 20% are much lower (1.0 and 2.4). Even with a rather high proportion of 20% spurious edges, the counts obtained on these modified networks are demonstrably similar to that of the original networks, providing further evidence to support the use of non-induced treelet counts in analysis of mildly noisy or incomplete networks.

F. Execution times for Analyses

The initial motivation for our partitioned implementation was to overcome the memory usage imposed by dynamic programming table for large networks. The multi-node parallel implementations also greatly reduce the time spent in performing the comparative network analyses reported in the previous subsections. Table II lists the approximate running times required to generate some of the counts used in Figures 5–8. The times are given by days (d), hours (h), minutes (m), and seconds (s) for processing time in serial, on a 16-core server, and on a 15-node cluster. The total times are approximate, as we utilized several different environments to collect the data points required to generate the figures. Some of the serial execution times were extrapolated based on strong scaling observed with the shared- and distributed-memory implementations.

Network	Size		Time		
	n	m	Serial	1 node	15 nodes
Motif counts					
Portland	1.6 M	31 M	4 d	11 h	1 h
Road	1 M	1.5 M	11 h	3 h	10 m
Slashdot	82 K	440 K	2 h	30 m	2 m
Enron	33 K	180 K	40 m	10 m	50 s
gnp33k	33 K	180 K	1 h	10 m	40 s
Network perturbation analysis (for Figs. 6–8)					
NotreWeb	330 K	760 K	2 d	15 h	1 h
Slashdot	82 K	440 K	1 d	6 h	30 m
p2p	63 K	77 K	10 h	3 h	10 m
gnp100k	100 K	1 M	3 d	10 h	40 m

Table II
APPROXIMATE ANALYSIS TIMES WITH FASCIA IN SERIAL, ON A SINGLE NODE (OPENMP), AND ON 15 NODES (OPENMP + MPI). SIZE: K= $\times 10^3$, M= $\times 10^6$, TIME: (D)AYS, (H)OURS, (M)INUTES, (S)ECONDS.

As seen in Table II, exploiting multi-node parallelism for large network analysis can lead to significant performance improvements. Motif finding, by calculating the counts on the given networks for 11 templates of size 7, could be completed in less than two hours on 15 nodes, with most of the time spent on counts for the large Portland network. Further, our network noise analysis, which involved calculating all 11 possible 7-vertex templates for 64 different network configurations, could be fully processed in about two-and-a-half hours. The number of color-coding iterations performed could be reduced for some of these networks if lower accuracy guarantees are desired.

G. Comparisons to Recent Work

SAHAD [13] and PARSE [12] by Zhao et al. both utilize the color-coding approach for determining approximate subgraph counts in distributed environments. PARSE is an MPI-based approach and SAHAD is a newer and more scalable version that uses Hadoop. The parallelization strategies and software environment used in their performance studies are very different from the settings used in our study. Hence it is difficult to perform a head-to-head comparison. To get a sense of relative speedup with our approach, consider the following selection of performance results: The PARSE paper reports an execution time of about an hour on 400 cores of a cluster for a 2 million vertex, 50 million edge network and a 6-vertex chain template. For the same network, a single color-coding iteration with SAHAD for a 10-vertex tree template is reported to take 25 minutes on 42 nodes (1344 cores). With distributed FASCIA on 15 nodes (240 cores), we can count occurrences of a 12-vertex chain and 12-vertex tree on a 3 million vertex, 117 million edge network in about 3 and 4 minutes, respectively.

Rahman et al. have recently designed GRAFT [3], a tool for quickly counting graphlets in large networks using a sampling-based technique. On the com-DBLP network from SNAP [19] ($n = 330K$, $m = 930K$), they report a single node execution time of about 47 seconds to count all 29 graphlets, with approximately 5% error. On the same network and with the same approximate error bound, FASCIA counts all 92 tree-structured templates of size 5 to 9 vertices, in about 78 seconds. Using networks grouped as peer-to-peer, collaboration, road, and citation, Rahman et al. demonstrated that graphlets could be used for clustering networks, and reported 77% and 91% clustering accuracy rates when using 29 and 18 graphlets, respectively. As mentioned before, FASCIA achieves up to 75% accuracy for clustering networks into 10 groups. Further quality and performance comparisons of sampling vs. color-coding and the use of graphlets vs. treelets would make for interesting future work.

VI. CONCLUSIONS

This work introduces distributed-memory parallelizations of FASCIA, a fast implementation of approximate subgraph counting using color-coding. By partitioning the dynamic programming table and distributing iterations across nodes of a cluster, approximate counts of non-induced tree-structured subgraphs, or treelets, can be retrieved quickly for networks with up to millions of vertices and hundreds of millions of edges.

Using this new implementation, we conducted an extensive study to highlight the efficacy of using large tree-structured templates to describe network topology, in the same way that smaller subgraphs of varying structures were previously used. Additionally, we demonstrate the robustness of using non-induced subgraph counts as graph signatures by quantifying the impact of random edge deletion and rewiring alterations on subgraph counts.

ACKNOWLEDGMENT

This work is supported by NSF grant ACI-1253881 and used instrumentation funded by the NSF grant OCI-0821527. This work also used the Extreme Science and Engineering Discovery Environment (XSEDE), which is

supported by NSF grant OCI-1053575. We thank Siva Rajamanickam (Sandia Labs) for facilitating timely access to computing resources at Sandia. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs," *Bioinformatics*, vol. 20, no. 11, pp. 1746–1758, 2004.

[2] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 347–359, 2004.

[3] M. Rahman, M. Bhuiyan, and M. Hasan, "GRAFT: An efficient graphlet counting method for large graph analysis," *IEEE Trans. on Knowledge and Data Engineering*, 2014, to appear.

[4] J. Chen, W. Hsu, M. L. Lee, and S.-K. Ng, "NeMoFinder: dissecting genome-wide protein-protein interactions with meso-scale network motifs," in *Proc. 12th ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data mining (KDD)*, 2006, pp. 106–115.

[5] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *J. ACM*, vol. 42, no. 4, pp. 844–856, 1995.

[6] G. M. Slota and K. Madduri, "Fast approximate subgraph counting and enumeration," in *Proc. 42nd Int'l. Conf. on Parallel Processing (ICPP)*, 2013, pp. 210–219.

[7] 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, no. 5594, pp. 824–827, 2002.

[8] S. Omidi, F. Schreiber, and A. Masoudi-Nejad, "MODA: an efficient algorithm for network motif discovery in biological networks," *Genes Genet Syst*, vol. 84, no. 5, pp. 385–395, 2009.

[9] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. Sahinalp, "Biomolecular network motif counting and discovery by color coding," *Bioinformatics*, vol. 24, no. 13, pp. i241–i249, 2008.

[10] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Proc. 3rd IEEE Int'l. Conf. on Data Mining (ICDM)*, 2003, p. 549.

[11] J. Leskovec, A. Singh, and J. Kleinberg, "Patterns of influence in a recommendation network," *Proc. 10th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pp. 380–389, 2006.

[12] Z. Zhao, M. Khan, V. S. A. Kumar, and M. V. Marathe, "Subgraph enumeration in large social contact networks using parallel color coding and streaming," in *Proc. 39th Int'l. Conf. on Parallel Processing (ICPP)*, 2010, pp. 594–603.

[13] Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. S. A. Kumar, and M. V. Marathe, "SAHAD: Subgraph analysis in massive networks using Hadoop," in *Proc. 26th Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, 2012, pp. 390–401.

[14] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, pp. e177–83, 2007.

[15] ——, "Modeling interactome, scale-free or geometric?" *Bioinformatics*, vol. 20, no. 18, pp. 3508–3515, 2004.

[16] T. Milenković and N. Pržulj, "Uncovering biological network function via graphlet degree signatures," *Cancer Informatics*, vol. 6, pp. 257–273, 2008.

[17] N. Pržulj, D. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein-protein interaction networks," *Bioinformatics*, vol. 22, no. 8, pp. 974–980, 2006.

[18] I. Bordino, D. Donata, A. Gionis, and S. Leonardi, "Mining large networks with subgraph counting," in *Proc. 8th IEEE Int'l. Conf. on Data Mining (ICDM)*, 2008, pp. 737–742.

[19] J. Leskovec, "SNAP: Stanford Network Analysis Project," http://snap.stanford.edu/index.html, last accessed Feb 2014.

[20] J. Kunegis, "KONECT - the Koblenz network collection," in *Proc. Int. Web Observatory Workshop*, 2013, pp. 1343–1350.

[21] "9th DIMACS Implementation Challenge – Shortest Paths," http://www.dis.uniroma1.it/challenge9/download.shtml, last accessed Feb 2014.

[22] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. on Knowledge Discovery from Data*, vol. 1, no. 1, 2007.

[23] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[24] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. 12th IEEE Int'l. Conf. on Data Mining (ICDM)*, 2012, pp. 745–754.

[25] V. Bimal, A. Mislove, M. Cha, and K. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. 2nd ACM Workshop on Online Social Networks (WOSN)*, 2009, pp. 37–42.

[26] B. Klimmt and Y. Yang, "Introducing the Enron corpus," in *Proc. 1st Conf. on Email and Anti-Spam (CEAS)*, 2004.

[27] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50–57, 2002.

[28] I. Xenarios, L. Salwinski, X. J. Duan, P. Higney, S. M. Kim, and D. Eisenberg, "DIP, the database of interacting proteins: a research tool for studying cellular networks of protein interactions," *Nucleic Acids Research*, vol. 30, no. 1, pp. 303–305, 2002.

[29] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006.

[30] J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *Proc. 26th Annual Conf. on Neural Inf. Proc. Systems (NIPS)*, 2012, pp. 548–556.

[31] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *Proc. 2nd Int'l. Semantic Web Conf. (ISWC)*, 2003, pp. 351–368.

[32] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *4th SIAM Int'l. Conf. on Data Mining (SDM)*, 2004, pp. 442–446.

[33] Network Dynamics and Simulation and Science Laboratory, "Synthetic data products for societal infrastructures and proto-populations: Data set 1.0," Virginia Polytechnic Institute and State University, Tech. Rep. NDSSL-TR-06-006, 2006.

[34] F. Ruskey, "The (Combinatorial) Object Server," http://theory.cs.uvic.ca/root.html, last accessed Feb 2014.

[35] G. M. Slota and K. Madduri, "Complex network analysis using parallel approximate motif counting," The Pennsylvania State University, Tech. Rep. CSE #14-001, 2014.