Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs

Ethan R. Elenberg, Karthikeyan Shanmugam,
Michael Borokhovich, Alexandros G. Dimakis
The University of Texas
Austin, Texas 78712, USA
{elenberg, karthiksh, michaelbor}@utexas.edu, dimakis@austin.utexas.edu

ABSTRACT

We study the problem of approximating the 3-profile of a large graph. 3-profiles are generalizations of triangle counts that specify the number of times a small graph appears as an induced subgraph of a large graph. Our algorithm uses the novel concept of 3-profile sparsifiers: sparse graphs that can be used to approximate the full 3-profile counts for a given large graph. Further, we study the problem of estimating local and ego 3-profiles, two graph quantities that characterize the local neighborhood of each vertex of a graph.

Our algorithm is distributed and operates as a vertex program over the GraphLab PowerGraph framework. We introduce the concept of edge pivoting which allows us to collect 2-hop information without maintaining an explicit 2-hop neighborhood list at each vertex. This enables the computation of all the local 3-profiles in parallel with minimal communication.

We test our implementation in several experiments scaling up to 640 cores on Amazon EC2. We find that our algorithm can estimate the 3-profile of a graph in approximately the same time as triangle counting. For the harder problem of ego 3-profiles, we introduce an algorithm that can estimate profiles of hundreds of thousands of vertices in parallel, in the timescale of minutes.

1. INTRODUCTION

Given a small integer k (*e.g.* $k = 3$ or 4), the k-profile of a graph $G(V, E)$ is a vector with one coordinate for each distinct k-node graph H_i (see Figure 1 for $k = 3$). Each coordinate counts the number of times that H_i appears as an induced subgraph of G. For example, the graph $G = K_4$ (the complete graph on 4 vertices) has the 3-profile $[0, 0, 0, 4]$ since it contains 4 triangles and no other (induced) subgraphs. The graph C_5 (the cycle on 5 vertices, *i.e.* a pentagon) has the 3-profile $[0, 5, 5, 0]$. Note that the sum of the k-profile is always $\binom{|V|}{k}$, the total number of subgraphs.

One can see k-profiles as a generalization of triangle (as well as other motif) counting problems. They are increasingly popular for graph analytics both for practical and theoretical reasons. They form a concise graph description that has found several applications for the web [4, 24], social networks [35], and biological networks [26] and seem to be empirically useful. Theoretically, they connect to the emerging theory of graph homomorphisms, graph limits and graphons [6, 35, 21].

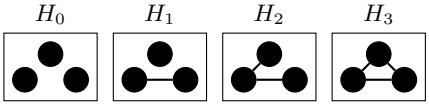


Figure 1: Subgraphs in the 3-profile of a graph. We call them (empty, edge, wedge, triangle). The 3-profile of a graph counts how many times each of H_i appears in G.

In this paper we introduce a novel distributed algorithm for estimating the $k = 3$-profiles of massive graphs. In addition to estimating the (global) 3-profile, we address two more general problems. One is calculating the *local* 3-profile for each vertex v_j. This assigns a vector to each vertex that counts how many times v_j participates in each subgraph H_i. These local vectors contain a higher resolution description of the graph and are used to obtain the global 3-profile (simply by rescaled addition as we will discuss).

The second related problem is that of calculating the ego 3-profile for each vertex v_j. This is the 3-profile of the graph $N(v_j)$ *i.e.* the neighbors of v_j, also called the ego graph of v_j. The 3-profile of the ego graph of v_j can be seen as a projection of the vertex into a coordinate system [35]. This is a very interesting idea of viewing a big graph as a collection of small dense graphs, in this case the ego graphs of the vertices. Note that calculating the ego 3-profiles for a set of vertices of a graph is different (in fact, significantly harder) than calculating local 3-profiles.

Contributions: Our first contribution is a provable edge sub-sampling scheme: we establish sharp concentration results for estimating the entire 3-profile of a graph. This allows us to randomly discard most edges of the graph and still have 3-profile estimates that are provably within a bounded error with high probability. Our analysis is based on modeling the transformation from original to sampled graph as a one step Markov chain with transitions expressed as a function of the sampling probability. Our result is that a random sampling of edges forms a 3-profile sparsifier, *i.e.* a subgraph that preserves the elements of the 3-profile with sufficient probability concentration. Our result is a general-

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

ACM 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783413.

ization of the triangle sparsifiers by Tsourakakis *et al.* [34]. Our proof relies on a result by Kim and Vu [15] on concentration of multivariate polynomials, similarly to [34]. Unfortunately, the Kim and Vu concentration holds only for a class of polynomials called totally positive and some terms in the 3-profile do not satisfy this condition. For that reason, the proof of [34] does not directly extend beyond triangles. Our technical innovation involves showing that it is still possible to decompose our polynomials as combinations of totally positive polynomials using a sequence of variable changes.

Our second innovation deals with efficiently designing a *distributed* algorithm for estimating 3-profiles on the subsampled graph. We rely on the Gather-Apply-Scatter model used in Graphlab PowerGraph [10] but, more generally, our algorithm fits the architecture of most graph engines. We introduce the concept of *edge pivoting* which allows us to collect 2-hop information without maintaining an explicit 2-hop neighborhood list at each vertex. This enables the computation of all the local 3-profiles in parallel. Each edge requires only information from its endpoints and each vertex only computes quantities using data from incident edges. For the problem of ego 3-profiles, we show how to calculate them by combining edge pivot equations and local clique counts.

We implemented our algorithm in GraphLab and performed several experiments scaling up to 640 cores on Amazon EC2. We find that our algorithm can estimate the 3-profile of a graph in approximately the same time as triangle counting. Specifically, we compare against the PowerGraph triangle counting routine and find that it takes us only 1%-10% more time to compute the full 3-profile. For the significantly harder problem of ego 3-profiles, we were able to compute (in parallel) the 3-profiles of up to $100,000$ ego graphs in the timescale of several minutes. We compare our parallel ego 3-profile algorithm to a simple sequential algorithm that operates on each ego graph sequentially and shows tremendous scalability benefits, as expected. Our datasets involve social network and web graphs with edges ranging in number from tens of millions to over one billion. We present results on both overall runtimes and network communication on multicore and distributed systems.

2. RELATED WORK

In this section, we describe several related topics and discuss differences in relation to our work.

Graph Sub-Sampling: Random edge sub-sampling is a natural way to quickly obtain estimates for graph parameters. For the case of triangle counting such graphs are called a triangle sparsifiers [34]. Related ideas were explored in the Doulion algorithm [32, 33, 34] with increasingly strong concentration bounds. The recent work by Ahmed *et al.* [1] develops subgraph estimators for clustering coefficient, triangle count, and wedge count in a streaming sub-sampled graph. Other recent work [29, 5, 14] uses random sampling to estimate parts of the 3 and 4-profile. These methods do not account for a distributed computation model and require more complex sampling rules. As discussed, our theoretical results build on [34] to define the first 3-profile sparsifiers, sparse graphs that are *a fortiori* triangle sparsifiers.

Triangle Counting in Graph Engines: Graph engines (*e.g.* Pregel, GraphLab, Galois, GraphX, see [27] for a comparison) are frameworks for expressing distributed computation on graphs in the language of vertex programs. Triangle counting algorithms [28, 4] form one of the standard graph analytics tasks for such frameworks [10, 27]. In [7], the authors list triangles efficiently, by partitioning the graph into components and processing each component in parallel. Typically, it is much harder to perform graph analytics over the MapReduce framework but some recent work [25, 31] has used clever partitioning and provided theoretical guarantees for triangle counting.

Matrix formulations: Fast matrix multiplication has been used for certain types of subgraph counting. Alon *et al.* proposed a cycle counting algorithm which uses the trace of a matrix power on high degree vertices [2]. Some of our edge pivot equations have appeared in [16, 17, 36], all in a centralized setting. Related approximation schemes [32] and randomized algorithms [36] depend on centralized architectures and computing matrix powers of very large matrices.

Frequent Subgraph Discovery: The general problem of finding frequent subgraphs, also known as motifs or subgraph isomorphisms, is to find the number of occurrences of a small query graph within a larger graph. Typically frequent subgraph discovery algorithms offer pruning rules to eliminate false positives early in the search [37, 19, 11]. This is most applicable when subgraphs have labelled vertices or directed edges. For these problems, the number of unique isomorphisms grows much larger than in our application.

In [35], subgraphs were queried on the ego graphs of users. While enumerating all 3-sets and sampling 4-sets neighbors can be done in parallel, forming the ego subgraphs requires checking for edges between neighbors. This suggests that a graph engine implementation would be highly preferable over an Apache Hive system. Our algorithms simultaneously compute the ego subgraphs and their profiles, reducing the amount of communication between nodes. Our algorithm is suitable for both NUMA multicore and distributed architectures, but our implementation focus in this paper is on GraphLab.

Graphlets: First described in [26], graphlets generalize the concept of vertex degree to include the connected subgraphs a particular vertex participates in with its neighbors. Unique graphlets are defined at a vertex based on its degree in the subgraph. Graphlet frequency distributions (GFDs) have proven extremely useful in the field of bioinformatics. Specifically, GFD analysis of protein interaction networks helps to design improved generative models [12], accurate similarity measures [26], and better features for classification [30]. Systems that use our edge pivot equations (in a different form) appear in prior literature for calculating GFDs [13, 22] but not for enabling distributed computation.

3. UNBIASED 3-PROFILE ESTIMATION

In this section, we are interested in estimating the number of 3 node subgraphs of type H_0, H_1, H_2 and H_3, as depicted in Figure 1, in a given graph G. Let the estimated counts be denoted $X_i,\ i \in \{0,1,2,3\}$. Let the actual counts be $n_i,\ i \in \{0,1,2,3\}$. This set of counts is called a 3-profile of the graph, denoted with the following vector:

$$\mathbf{n}_3(G) = [n_0,\, n_1,\, n_2,\, n_3]\,. \tag{1}$$

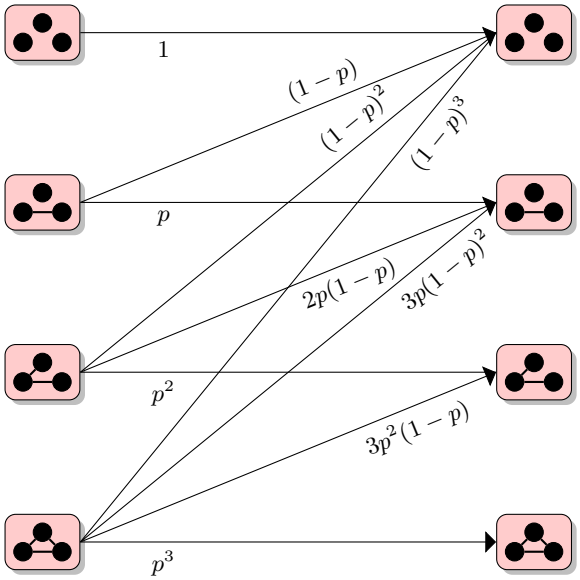


Figure 2: Edge sampling process.

Because the vector is a scaled probability distribution, there are only 3 degrees of freedom. Therefore, we calculate

$$\mathbf{n}_3(G) = \left[\binom{|V|}{3} - n_1 - n_2 - n_3,\, n_1,\, n_2,\, n_3 \right].$$

In the case of a large graph, computational difficulty in estimating the 3-profile depends on the total number of edges in the large graph. We would like to estimate each of the 3-profile counts within a multiplicative factor. So we first sub-sample the set of edges in the graph with probability p. We compute all 3-profile counts of the sub-sampled graph exactly. Let $\{Y_i\}_{i=0}^3$ denote the exact 3-profile counts of the random sub-sampled graph. We relate the sub-sampled 3-profile counts to the original ones through a one step Markov chain involving transition probabilities. The sub-sampling process is the random step in the chain. Any specific subgraph is preserved with some probability and otherwise transitions to one of the other subgraphs. For example, a 3-clique is preserved with probability p^3. Figure 2 illustrates the other transition probabilities.

In expectation, this yields the following linear system:

$$\begin{bmatrix} \mathbb{E}[Y_0] \\ \mathbb{E}[Y_1] \\ \mathbb{E}[Y_2] \\ \mathbb{E}[Y_3] \end{bmatrix} = \begin{bmatrix} 1 & 1-p & (1-p)^2 & (1-p)^3 \\ 0 & p & 2p(1-p) & 3p(1-p)^2 \\ 0 & 0 & p^2 & 3p^2(1-p) \\ 0 & 0 & 0 & p^3 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix}, \quad (2)$$

from which we obtain unbiased estimators for each entry in $\mathbf{X}(G) = [X_0,\, X_1,\, X_2,\, X_3]$:

$$X_0 = Y_0 - \frac{1-p}{p} Y_1 + \frac{(1-p)^2}{p^2} Y_2 - \frac{(1-p)^3}{p^3} Y_3 \quad (3)$$

$$X_1 = \frac{1}{p} Y_1 - \frac{2(1-p)}{p^2} Y_2 + \frac{3(1-p)^2}{p^3} Y_3 \quad (4)$$

$$X_2 = \frac{1}{p^2} Y_2 - \frac{3(1-p)}{p^3} Y_3 \quad (5)$$

$$X_3 = \frac{1}{p^3} Y_3. \quad (6)$$

LEMMA 1. *$\mathbf{X}(G)$ is an unbiased estimator of $\mathbf{n}(G)$.*

PROOF. By substituting (3)–(6) into (2), clearly $\mathbb{E}[X_i] = n_i$ for $i = 0, 1, 2, 3$. □

We now turn to prove concentration bounds for the above estimators. We introduce some notation for this purpose. Let X be a real polynomial function of m real random variables $\{t_i\}_{i=1}^m$. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in \mathbb{Z}_+^m$ and define $\mathbb{E}_{\geq 1}[X] = \max_{\boldsymbol{\alpha}:\|\boldsymbol{\alpha}\|_1 \geq 1} \mathbb{E}(\partial^{\boldsymbol{\alpha}} X)$, where

$$\mathbb{E}(\partial^{\boldsymbol{\alpha}} X) = \mathbb{E}\left[\left(\frac{\partial}{\partial t_1}\right)^{\alpha_1} \ldots \left(\frac{\partial}{\partial t_m}\right)^{\alpha_m} [X(t_1, \ldots, t_m)] \right]. \quad (7)$$

Further, we call a polynomial totally positive if the coefficients of all the monomials involved are non-negative. We state the main technical tool we use to obtain our concentration results.

THEOREM 1 (KIM-VU CONCENTRATION [15]). *Let X be a random totally positive Boolean polynomial in m Boolean random variables with degree at most k. If $\mathbb{E}[X] \geq \mathbb{E}_{\geq 1}[X]$, then*

$$\mathbb{P}\left(|X - \mathbb{E}[X]| > a_k \sqrt{\mathbb{E}[X]\mathbb{E}_{\geq 1}[X]}\lambda^k \right) = \mathcal{O}\left(\exp\left(-\lambda + (k-1)\log m\right)\right) \quad (8)$$

for any $\lambda > 1$, where $a_k = 8^k k!^{1/2}$.

The above theorem was used to analyze 3-profiles of Erdős-Rényi random ensembles ($G_{n,p}$) in [15]. Later, this was used to derive concentration bounds for triangle sparsifiers in [34]. Here, we extend §4.3 of [15] to the 3-profile estimation process, on an arbitrary edge-sampled graph.

THEOREM 2. *(Generalization of triangle sparsifiers to 3-profile sparsifiers) Let $\mathbf{n}(G) = [n_0, n_1, n_2, n_3]$ be the 3-profile of a graph $G(V, E)$. Let $|V| = n$ and $|E| = m$. Let $\hat{\mathbf{n}}(G) = [Y_0, Y_1, Y_2, Y_3]$ be the 3-profile of the subgraph obtained by sampling each edge in G with probability p. Let α, β and Δ be the largest collection of H_1's, wedges and triangles that share a common edge. Define $\mathbf{X}(G)$ according to (3)–(6), $\epsilon > 0$, and $\gamma > 0$. If p, ϵ satisfy:*

$$\begin{aligned} \frac{n_0}{3\max\{\alpha, \beta, \Delta\}} &\geq \frac{a_3^2 \log^6\left(m^{2+\gamma}\right)}{\epsilon^2} \\ \frac{p}{\max\{\frac{1}{\sqrt[3]{n_3}}, \Delta/n_3\}} &\geq \frac{a_3^2 \log^6\left(m^{2+\gamma}\right)}{\epsilon^2} \\ \frac{p}{\alpha/n_1} &\geq \frac{a_1^2 \log^2\left(m^{\gamma}\right)}{\epsilon^2} \\ \frac{p}{\max\{\frac{\beta}{n_2}, \frac{1}{\sqrt{n_2}}\}} &\geq \frac{a_2^2 \log^4\left(m^{1+\gamma}\right)}{\epsilon^2}, \end{aligned} \quad (9)$$

then $\|\mathbf{X}(G) - \mathbf{n}(G)\|_\infty \leq 12\epsilon\binom{|V|}{3}$ with probability at least $1 - \frac{1}{m^\gamma}$.

PROOF. The following sketch outlines the new polynomial decomposition technique required to apply the Kim-Vu concentration. Full proof details can be found in [9].

Let m be the total number of edges in the original graph G. If e is an edge in the original graph G, let t_e be the random indicator after sampling: $t_e = 1$ if e is sampled and 0 otherwise. Let $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ denote the set of distinct subgraphs of the kind H_0, H_1, H_2 and H_3 (anti-clique, edge,

wedge and triangle) respectively. Let $A, \_(e), \Lambda(e,f)$ and $\Delta(e,f,g)$ denote an anti-clique with no edges, a H_1 with edge e , a H_2 with two edges e, f and a triangle with edges e, f, g respectively in the original graph G. Our estimators (3)-(6) are a function of Y_i's and each Y_i can be written as a polynomial of at most degree 3 in all the variables t_e.

$$
\begin{aligned}
Y_0 &= n_0 + \sum_{\_(e)\in\mathcal{H}_1}(1-t_e) + \sum_{\Lambda(e,f)\in\mathcal{H}_2}(1-t_e)(1-t_f) + \\
&\quad \sum_{\Delta(e,f,g)\in\mathcal{H}_3}(1-t_e)(1-t_f)(1-t_g) \\
Y_3 &= \sum_{\Delta(e,f,g)\in\mathcal{H}_3} t_e t_f t_g, \quad D_1 = \sum_{\Lambda(e,f)\in\mathcal{H}_2}(t_e+t_f) \\
D_2 &= \sum_{\Lambda(e,f)\in\mathcal{H}_2} t_e t_f, \quad T_1 = \sum_{\Delta(e,f,g)\in\mathcal{H}_3}(t_e+t_f+t_g) \quad (10) \\
T_2 &= \sum_{\Delta(e,f,g)\in\mathcal{H}_3}(t_e t_f + t_f t_g + t_g t_e), \; S_1 = \sum_{\_(e)\in\mathcal{H}_1} t_e \\
Y_1 &= S_1 + D_1 - 2D_2 + T_1 - 2T_2 + 3Y_3 \\
Y_2 &= D_2 + T_2 - 3Y_3
\end{aligned}
$$

We observe that in the above even by change of variables $y_e = (1-t_e)$, Y_1 and Y_2 are not totally positive polynomials. This means that Theorem 1 cannot be applied directly to the $Y_i's$ or X_i's. The strategy we adopt is to split the Y_1 and Y_2 into many polynomials, each of which is totally positive, and then apply Theorem 1 on each of them. $P = \{Y_0, Y_3, S_1, D_1, D_2, T_1, T_2\}$ form the set of totally positive polynomials (proved below). Substituting the above equations into (3)-(6), we have the following system of equations that connect X_i's and the set of totally positive polynomials P. We only show the equation for X_0 for brevity:

$$
\begin{aligned}
X_0 = Y_0 &- \frac{1-p}{p}(S_1 + D_1 + T_1) \\
&+ \frac{1-p^2}{p^2}(D_2 + T_2) - \frac{1-p^3}{p^3}Y_3. \qquad (11)
\end{aligned}
$$

Let α_e, β_e, and Δ_e be the maximum number of H_1's, H_2's, and H_3's containing an edge e in the original graph G. Let α, β and Δ be the maximum of α_e, β_e, and Δ_e over all edges e. We now show concentration results for the totally positive polynomials alone. The detailed proof of the following lemma is in the extended version [9].

LEMMA 2. *Define variables $y_e = 1-t_e$. Then Y_0 is totally positive in y_e. With respect to the variables y_e, $\mathbb{E}_{\geq 1}[Y_0] \leq 3\max\{\alpha,\beta,\Delta\}$. Further, Y_3, S_1, D_1, T_1, T_2 and D_2 are totally positive in the variables t_e. With respect to the variables t_e, we have:*

1. $\mathbb{E}_{\geq 1}[Y_3] \leq \max\{1, p^2\Delta\}$, $\mathbb{E}_{\geq 1}[S_1] \leq \alpha$.
2. $\mathbb{E}_{\geq 1}[D_1] \leq \beta$, $\mathbb{E}_{\geq 1}[T_1] \leq \Delta$.
3. $\mathbb{E}_{\geq 1}[D_2] \leq \max\{p\beta, 1\}$, $\mathbb{E}_{\geq 1}[T_2] \leq \max\{2p\Delta, 1\}$.

Let $A \in P$. Then $\mathbb{E}_{\geq 1}[A] \leq \mathbb{E}[A]$ and Lemma 2 imply the following conditions:

$$
n_0 \geq 3\max\{\alpha,\beta,\Delta\}, \; p \geq \max\{\frac{1}{\sqrt[3]{n_3}}, \frac{1}{\sqrt{n_2}}, \frac{\Delta}{n_3}, \frac{\beta}{n_2}, \frac{\alpha}{n_1}\} \qquad (12)
$$

We apply Theorem 1 to all the totally positive polynomials, along with condition (12). Let the λ variables in Theorem 1 be denoted by λ_1(for Y_0), λ_2(for Y_3), λ_3(for S_1), λ_4(for D_1), λ_5(for T_1), λ_6(for D_2), and λ_7(for T_2). Choose an $\epsilon > 0$. We force the following conditions:

$$
\begin{aligned}
&a_3\sqrt{\mathbb{E}[Y_0]\mathbb{E}_{\geq 1}[Y_0]}\lambda_1^3 = \epsilon\mathbb{E}[Y_0], \; a_3\sqrt{\mathbb{E}[Y_3]\mathbb{E}_{\geq 1}[Y_3]}\lambda_2^3 = \epsilon\mathbb{E}[Y_3] \\
&a_1\sqrt{\mathbb{E}[S_1]\mathbb{E}_{\geq 1}[S_1]}\lambda_3 = \epsilon\mathbb{E}[S_1], \; a_1\sqrt{\mathbb{E}[D_1]\mathbb{E}_{\geq 1}[D_1]}\lambda_4 = \epsilon\mathbb{E}[D_1] \\
&a_1\sqrt{\mathbb{E}[T_1]\mathbb{E}_{\geq 1}[T_1]}\lambda_5 = \epsilon\mathbb{E}[T_1], \; a_2\sqrt{\mathbb{E}[D_2]\mathbb{E}_{\geq 1}[D_2]}\lambda_6^2 = \epsilon\mathbb{E}[D_2] \\
&a_2\sqrt{\mathbb{E}[T_2]\mathbb{E}_{\geq 1}[T_2]}\lambda_7^2 = \epsilon\mathbb{E}[T_2]
\end{aligned}
$$

Let $\gamma > 0$. For the right hand side of the inequalities in Theorem 1 to be $\mathcal{O}(\exp(-\gamma\log m))$, assuming all the bounds in Lemma 2, it is sufficient to have

$$
\begin{aligned}
\frac{n_0}{3\max\{\alpha,\beta,\Delta\}} &\geq \frac{a_3^2\log^6\left(m^{2+\gamma}\right)}{\epsilon^2} \\
\frac{p}{\max\{\frac{1}{\sqrt[3]{n_3}}, \Delta/n_3\}} &\geq \frac{a_3^2\log^6\left(m^{2+\gamma}\right)}{\epsilon^2} \\
\frac{p}{\alpha/n_1} &\geq \frac{a_1^2\log^2\left(m^{\gamma}\right)}{\epsilon^2} \\
\frac{p}{\max\{\frac{\beta}{n_2}, \frac{1}{\sqrt{n_2}}\}} &\geq \frac{a_2^2\log^4\left(m^{1+\gamma}\right)}{\epsilon^2}.
\end{aligned} \qquad (13)
$$

These conditions are obtained via intermediate conditions on the λ_i variables and (12). Again, the missing details are found in the extended version of this paper [9]. Therefore, subject to (13), all totally positive polynomials concentrate within a multiplicative factor of $(1\pm\epsilon)$ with probability at least $1 - \mathcal{O}\left(\frac{1}{m^\gamma}\right)$.

Under the above concentration result, let the deviations of X_i's be denoted by δX_i. Now we calculate the deviation of X_0 using (11).

$$
\begin{aligned}
\delta X_0 &\leq \epsilon\mathbb{E}[Y_0] + \epsilon\frac{1-p}{p}\left(|\mathbb{E}[S_1]| + |\mathbb{E}[D_1]| + |\mathbb{E}[T_1]|\right) \\
&\quad + \epsilon\frac{1-p^2}{p^2}\left(|\mathbb{E}[D_2]| + |\mathbb{E}[T_2]|\right) - \epsilon\frac{1-p^3}{p^3}|\mathbb{E}[Y_3]| \\
&\leq \epsilon(n_0 + n_1 + 3n_2 + 7n_3) \leq 7\epsilon(n_0 + n_1 + n_2 + n_3).
\end{aligned}
$$

Similarly for other X_i's, we get

$$
\delta X_1 \leq 12\epsilon\left(n_1 + n_2 + n_3\right), \; \delta X_2 \leq 6\epsilon\left(n_2 + n_3\right), \; \delta X_3 \leq \epsilon n_3.
$$

Therefore, sampling every edge independently with probability p satisfying all conditions in (13), all $X_i's$ concentrate within an additive gap of $(1\pm 12\epsilon)\binom{|V|}{3}$ with probability at least $1 - \frac{1}{m^\gamma}$. The constants in this proof can be tightened by a more accurate analysis. □

The sampling probablilty p in Theorem 2 depends polylogarithmically on the number of edges and linearly on the fraction of each subgraph which occurs on a common edge. For example, if all of the wedges in G depend on a single edge, *i.e.* $\beta = n_2$, then the last equation suggests the presence of that particular edge in the sampled graph will dominate the overall sparsifier quality.

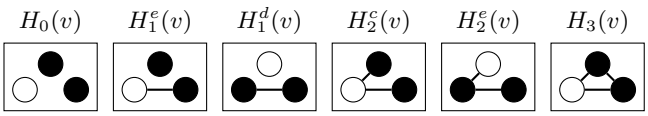


Figure 3: Unique 3-subgraphs from vertex perspective (white vertex corresponds to v).

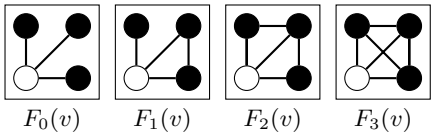


Figure 4: 4-subgraphs for Ego 3-profiles (white vertex corresponds to v).

4. LOCAL 3-PROFILE CALCULATION

In this section, we describe how to obtain two types of 3-profiles for a given graph G in a deterministic manner. These algorithms are distributed and can be applied independently of the edge sampling described in Section 3.

The key to our approach is to identify subgraphs at a vertex based on degree with which it participates in the subgraph. From the perspective of a given vertex v, there are actually six distinct 3 node subgraphs up to isomorphism as given in Figure 3. Let $n_{0,v}, n^e_{1,v}, n^d_{1,v}, n^e_{2,v}, n^d_{2,v}$, and $n_{3,v}$ denote the corresponding local subgraph counts at v. We will first outline an approach that calculates these counts and then add across different vertex perspectives to calculate the final 4 scalars ($n_{2,v} = n^e_{2,v} + n^c_{2,v}$ and $n_{1,v} = n^e_{1,v} + n^d_{1,v}$). It is easy to see that the global counts can be obtained from these local counts by summing across vertices:

$$n_i = \frac{1}{3}\left(\sum_{v\in V} n_{i,v}\right), \ \forall i. \tag{14}$$

4.1 Distributed Local 3-profile

We will now give our approach for calculating the local 3-profile counts of $G(V,E)$ using only local information combined with $|V|$ and $|E|$.

Scatter: We assume that every edge (v,a) has access to the neighborhood sets of both v and a, i.e. $\Gamma(v)$ and $\Gamma(a)$. Therefore, intersection sizes are first calculated at every edge, *i.e.* $|\Gamma(v)\cap\Gamma(a)|$. Each edge computes the following scalars and stores them:

$$\begin{aligned}
&n_{3,va} = |\Gamma(v)\cap\Gamma(a)|, \ n^e_{2,va} = |\Gamma(v)| - |\Gamma(v)\cap\Gamma(a)| - 1.\\
&n^c_{2,va} = |\Gamma(a)| - |\Gamma(v)\cap\Gamma(a)| - 1.\\
&n_{1,va} = |V| - (|\Gamma(v)| + |\Gamma(a)| - |\Gamma(v)\cap\Gamma(a)|).
\end{aligned} \tag{15}$$

The computational effort at every edge is at most $\mathcal{O}(d_{\max})$, where $d_{\max}$ is the maximum degree of the graph, for the neighborhood intersection size.

Gather: In the next round, vertex v "gathers" the above scalars in the following way:

$$\begin{aligned}
&n^e_{2,v} = \sum_{a\in\Gamma(v)} n^e_{2,va}, \ \ n^e_{1,v} = \sum_{a\in\Gamma(v)} n_{1,va}.\\
&n^c_{2,v} \overset{a}{=} \frac{1}{2}\sum_{a\in\Gamma(v)} n^c_{2,va}, \ \ n_{3,v} \overset{b}{=} \frac{1}{2}\sum_{a\in\Gamma(v)} n_{3,va}.\\
&n^d_{1,v} \overset{c}{=} |E| - |\Gamma(v)| - n_{3,v} - n^e_{2,v}.\\
&n_{0,v} = \binom{|V|-1}{2} - n_{1,v} - n_{2,v} - n_{3,v}.
\end{aligned} \tag{16}$$

Here, relations (a) and (b) are because triangles and wedges from center are double counted. (c) comes from noticing that each triangle and wedge from endpoint excludes an extra edge from forming $H^d_1(v)$. In this Gather stage, the communication complexity is $\mathcal{O}(M)$ where it is assumed that $\Gamma(v)$ is stored over M different machines. The corresponding distributed algorithm is described in Algorithm 1.

4.2 Distributed Ego 3-profile

In this section, we give an approach to compute ego 3-profiles for a set of vertices $\mathcal{V} \subseteq V$ in G. For each vertex v, the algorithm returns a 3-profile corresponding to that vertex's ego $N(v)$, a subgraph induced by the neighborhood set $\Gamma(v)$, including edges between neighbors and excluding v itself. Formally, our goal is to compute $\{\mathbf{n}(N(v))\}_{v\in\mathcal{V}}$. Clearly, this can be accomplished in two steps repeated serially on all $v \in \mathcal{V}$: first obtain the ego subgraph $N(v)$ and then pass as input to Algorithm 1, summing over the ego vertices $\Gamma(v)$ to get a global count. The serial implementation is provided in Algorithm 2. We note that this was essentially done in [35], where ego subgraphs were extracted from a common graph separately from 3-profile computations.

Instead, Algorithm 3 provides a parallel implementation which solves the problem by finding cliques in parallel for all $v \in \mathcal{V}$. The main idea behind this approach is to realize that calculating the 3-profile on the induced subgraph $N(v)$ is exactly equivalent to computing specific 4-node subgraph frequencies among v and 3 of its neighbors, enumerated as $F_i(v)$, $0 \le i \le 3$ in Figure 4. Now, the aim is to calculate $F_i(v)$'s, effectively part of a local 4-profile.

Scatter: We assume that every edge (v,a) has already computed the scalars from (15). Additionally, every edge (v,a) also computes the list $\mathcal{N}_{va} = \Gamma(v)\cap\Gamma(a)$ instead of only its size. The computational complexity is still $\mathcal{O}(d_{\max})$.

Gather: First, the vertex "gathers" the following scalars, forming three *edge pivot equations* in unknown variables $F_i(v)$:

$$\begin{aligned}
\sum_{a\in\Gamma(v)} \binom{n^c_{2,va}}{2} &= 3F_0(v) + F_1(v)\\
\sum_{a\in\Gamma(v)} \binom{n_{3,va}}{2} &= F_2(v) + 3F_3(v)\\
\sum_{a\in\Gamma(v)} n^c_{2,va} n_{3,va} &= 2F_1(v) + 2F_2(v)
\end{aligned} \tag{17}$$

By choosing two subgraphs that the edge (v,a) participates in, and then summing over neighbors a, these equations gather implicit connectivity information 2 hops away from v. However, note that there are only three equations in four variables and we must count one of them directly, namely the number of 4-cliques $F_3(v)$. Therefore, at the same gather step, the vertex also creates the list $\mathcal{CN}_v = \bigcup_{a\in\Gamma(v), p\in\mathcal{N}_{va}}(a,p)$. Essentially, this is the list of edges in the subgraph induced by $\Gamma(v)$. This requires worst case communication proportional to the number of edges in $N(v)$, independent of the number of machines M.

Scatter: Now, at the next scatter stage, each edge (v, a) accesses the pair of lists $\mathcal{CN}_v, \mathcal{CN}_a$. Now, each edge (v, a) computes the number of 4-cliques it is a part of, defined as follows:

$$n_{4,va} = \sum_{i,j \in \Gamma(v) \cap \Gamma(a)} \mathbf{1}\left((i,j) \in \mathcal{CN}_v\right). \tag{18}$$

This incurs computation time of $|\mathcal{CN}_v|$.
Gather: In the final gather stage, every vertex v accumulates these scalars to get $F_3(v) = \frac{1}{3} \sum_{a \in \Gamma(v)} n_{4,va}$ requiring $\mathcal{O}(M)$ communication time. As in the previous section, the scaling accounts for extra counting. Finally, the vertex solves the equations (17) using $F_3(v)$.

5. IMPLEMENTATION AND RESULTS

In this section, we describe the implementation and the experimental results of the 3-PROF, EGO-PAR and EGO-SER algorithms. We implement our algorithms on GraphLab v2.2 (PowerGraph) [10]. The performance (running time and network usage) of our 3-PROF algorithm is compared with the Undirected Triangles Count Per Vertex (hereinafter referred to as TRIAN) algorithm shipped with GraphLab. We show that in time and network usage comparable to the built-in TRIAN algorithm, our 3-PROF can calculate all the local and global 3-profiles. Then, we compare our parallel implementation of the ego 3-profile algorithm, EGO-PAR, with the naive serial implementation, EGO-SER. It appears that our parallel approach is much more efficient and scales much better than the serial algorithm. The sampling approach, introduced for the 3-PROF algorithm, yields promising results – reduced running time and network usage while still providing excellent accuracy. We support our findings with several experiments over various data sets and systems.
Vertex Programs: Our algorithms are implemented using a standard GAS (gather, apply, scatter) model [10]. We implement the three functions `gather()`, `apply()`, and `scatter()` to be executed by each vertex. Then we signal subsets of vertices to run in a specific order.

Algorithm 1 3-PROF

Input: Graph $G(V, E)$ with $|V|$ vertices, $|E|$ edges
Gather: For each vertex v union over edges of the 'other' vertex in the edge, $\cup_{a \in \Gamma(v)} a = \Gamma(v)$.
Apply: Store the gather as vertex data `v.nb`, size automatically stored.
Scatter: For each edge e_{va}, compute and store scalars in (15).
Gather: For each vertex v, sum edge scalar data of neighbors
$\mathtt{g} \leftarrow \sum_{(v,a) \in \Gamma(v)} \mathtt{e.data}$.
Apply: For each vertex v, calculate and store the quantities described in (16).
return [`v: v.n0 v.n1 v.n2 v.n3`]

The Systems: We perform the experiments on three systems. The first system is a single power server, further referred to as Asterix. The server is equipped with 256 GB of RAM and two Intel Xeon E5-2699 v3 CPUs, 18 cores each. Since each core has two hardware threads, up to 72 logical cores are available to the GraphLab engine.

Algorithm 2 EGO-SER

Input: Graph $G(V, E)$ with $|V|$ vertices, $|E|$ edges, set of ego vertices $\mathcal{V}$
for $v \in \mathcal{V}$ **do**
Signal v and its neighbors.
Include an edge if both its endpoints are signaled.
Run Algorithm 1 on the graph induced by the neighbors and edges between them.
end for
return [`v: vego.n0 vego.n1 vego.n2 vego.n3`]

Algorithm 3 EGO-PAR

Input: Graph $G(V, E)$ with $|V|$ vertices, $|E|$ edges, set of ego vertices $\mathcal{V}$
Gather: For each vertex v union over edges of the 'other' vertex in the edge, $\cup_{e_{va}} a = \Gamma(v)$.
Apply: Store the gather as vertex data `v.nb`, size automatically stored.
Scatter: For each edge e_{va}, compute and store as edge data:
Scalars in (15).
The list $\mathcal{N}_{va}$.
Gather: For each vertex v, sum edge data of neighbors:
Acumulate LHS of (17).
$\mathtt{g}.\mathcal{CN} \leftarrow \mathtt{g}.\mathcal{CN} \cup \mathcal{N}_{va}$.
Apply: Obtain $\mathcal{CN}_v$ and equations in (17) using the scalars and $\mathtt{g}.\mathcal{CN}$.
Scatter: Scatter $\mathcal{CN}_v, \mathcal{CN}_a$ to all edges (v, a).
Compute $n_{4,va}$ as in (18).
Gather: Sum edge data $n_{4,va}$ of neighbors at v.
Apply: Compute $F_3(v)$.
return [`v: vego.n0 vego.n1 vego.n2 vego.n3`]

The next two systems are EC2 clusters on AWS (Amazon Web Services) [3]. One is comprised of 12 m3.2xlarge machines, each having 30 GB RAM and 8 virtual CPUs. Another system is a cluster of 20 c3.8xlarge machines, each having 60 GB RAM and 32 virtual CPUs.
The Data: In our experiments we used five real graphs. These graphs represent different datasets: social networks (LiveJournal and Twitter), citations (DBLP), knowledge content (Wikipedia), and WWW structure (PLD – pay level domains). Graph sizes are summarized in Table 1.

Table 1: Datasets

Name	Vertices	Edges (undirected)
Twitter [18]	41,652,230	1,202,513,046
PLD [23]	39,497,204	582,567,291
LiveJournal [20]	4,846,609	42,851,237
Wikipedia [8]	3,515,067	42,375,912
DBLP [20]	317,080	1,049,866

5.1 Results

Experimental results are averaged over $3 - 10$ runs.
Local 3-profile vs. triangle count: The first result is that our 3-PROF is able to compute all the local 3-profiles in almost the same time as the GraphLab's built-in TRIAN computes the local triangles (i.e., number of triangles including each vertex). Let us start with the first AWS cluster

with less powerful machines (m3.x2large). In Figure 5 (a) we can see that for the LiveJornal graph, for each sampling probability p and for each number of nodes (i.e., machines in the cluster), 3-PROF achieves running times comparable to TRIAN. Notice also the benefit in running time achieved by sampling. We can reduce running time almost by half, without significantly sacrificing accuracy (which will be discussed shortly). While the running time is decreased as the number of nodes grows (more computing resources become available), the network usage becomes higher (see Figure 5 (c)) due to the extensive inter-machine communication in GraphLab. We can also see that sampling can significantly reduce network usage. In Figures 5 (b) and (d), we can see similar behavior for the Wikipedia graph: running time and network usage of 3-PROF is comparable to TRIAN.

Next, we conduct the experiments on the second AWS cluster with more powerful (c3.8xlarge) machines. For LiveJournal, we note modest improvements in running time for nearly the same network bandwidth observed in Figure 5. On this system we were able to run 3-PROF and TRIAN on the much larger PLD graph. In Figures 6 (b) and (d) we compare the running time and network usage of both algorithms. For the large PLD graph, the benefit of sampling can be seen clearly; by setting $p = 0.1$, the running time of 3-PROF is reduced by a factor of 4 and the network usage is reduced by a factor of 2. Figure 7 shows the performance of 3-PROF and TRIAN on the LiveJournal and Wikipedia graphs. We can see that the behavior of running times and the network usage of the 3-PROF algorithm is consistently comparable to TRIAN across the various graphs, sampling, and system parameters.

Let us now show results of the experiments performed on a single powerful machine (Asterix). Figure 11 (a) shows the running times for 3-PROF and TRIAN for Twitter and PLD graphs. We can see that on the largest graph in our dataset (Twitter), the running time of 3-PROF is less than 5% larger than that of TRIAN, and for the PLD graph the difference is less than 3% (for $p = 1$). Twitter takes roughly twice as long to compute as PLD, implying that these algorithms have running time proportional to the graph's number of edges.

Finally, we show that while the sampling approach can significantly reduce the running time and network usage, it has negligible effect on the accuracy of the solution. Notice that the sampling accuracy refers to the global 3-profile count (*i.e.*, the sum of all the local 3-profiles over all vertices in a graph). In Figure 12 we show accuracy of each scalar in the 3-profile. For the accuracy metrics, we use ratio between the exact count (obtained running 3-PROF with $p = 1$) divided by the estimated count (i.e., the output of our 3-PROF when $p < 1$). It can be seen that for the three graphs, all the 3-profiles are very close to 1. E.g., for the PLD graph, even when $p = 0.01$, the accuracy is within 0.004 from the ideal value of 1. Error bars mark one standard deviation from the mean, and across all graphs the largest standard deviation is 0.031. As p decreases, the triangle estimator suffers the greatest loss in both accuracy and consistency.

Ego 3-profiles: The next set of experiments evaluates the performance of our EGO-PAR algorithm for counting ego 3-profiles. We show the performance of EGO-PAR for various graphs and systems and also compare it to a naive serial algorithm EGO-SER. Let us start with the AWS system with (c3.8xlarge machines). In Figure 8 we see the

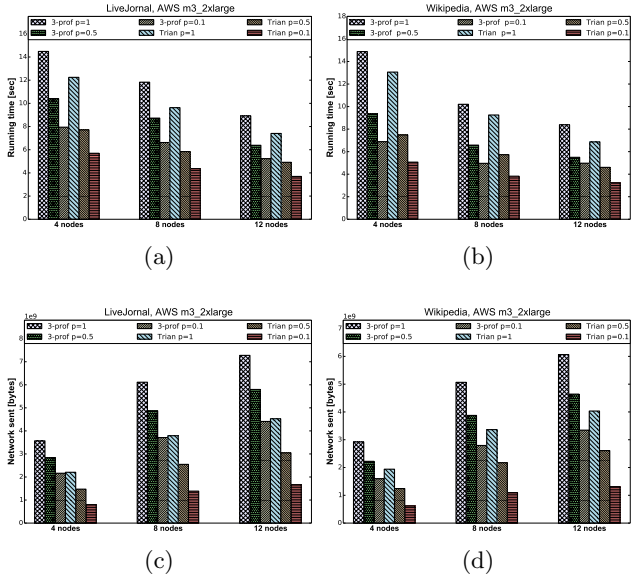


Figure 5: AWS `m3_2xlarge` cluster. 3-prof vs. trian algorithms for LiveJournal and Wikipedia datasets (average of 3 runs). 3-prof achieves comparable performance to triangle counting. (a,b) – Running time for various numbers of nodes (machines) and various sampling probabilities p. (c,d) – Network bytes sent by the algorithms for various numbers of nodes and various sampling probabilities p.

running time of EGO-SER and EGO-PAR on the LiveJournal graph. The task was to find ego 3-profiles of 100, 1K, and 10K randomly selected nodes. Since the running time depends on the size and structure of each induced subgraph, EGO-SER and EGO-PAR operated on the same list of ego vertices. While for 100 random vertices EGO-SER performed well (and even achieved the same running time as EGO-PAR for the PLD graph), its performance drastically degraded for a larger number of vertices. This is due to its iterative nature – it finds ego 3-profiles of the vertices one at a time and is not scalable. Note that the open bars mean that this experiment was not finished. The numbers above them are extrapolations, which are reasonable due to the serial design of the EGO-SER.

On the contrary, the EGO-PAR algorithm scales extremely well and computes ego 3-profiles for 100, 1K, and 10K vertices almost in the same time. In Figure 9 (a), we can see that as the number of nodes (i.e., machines) increases, running time of EGO-PAR decreases since its parallel design allows it to use additional computational resources. However, EGO-SER cannot benefit from more resources and its running time even increases when more machines are used. The increase in running time of EGO-SER is due to the increase in network usage when using more machines (see Figure 9 (b)). The network usage of EGO-PAR also increases, but this algorithm compensates by leveraging additional computational power. In Figure 10, we can see that EGO-PAR performs well even when finding ego 3-profiles for all the LiveJournal vertices (4.8M vertices).

Finally in Figure 11 (b) and (c), we can see the comparison of EGO-PAR and EGO-SER on the PLD and the DBLP graphs

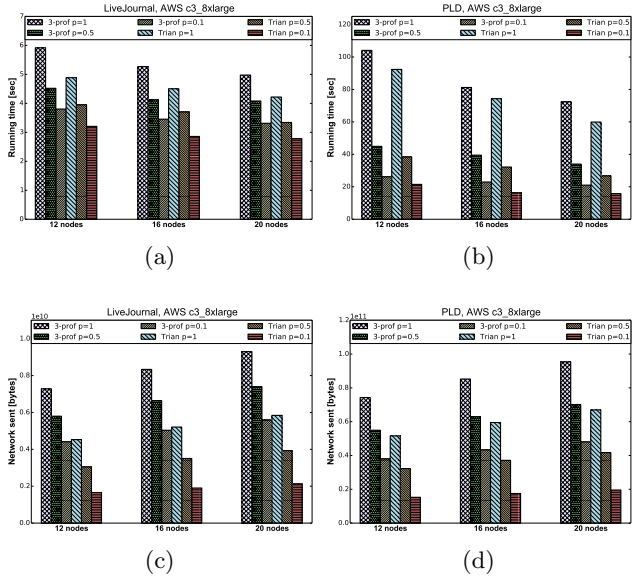


Figure 6: AWS `c3_8xlarge` cluster. 3-prof vs. trian algorithms for LiveJournal and PLD datasets (average of 3 runs). 3-prof achieves comparable performance to triangle counting. (a,b) – Running time for various numbers of nodes (machines) and various sampling probabilities p. (c,d) – Network bytes sent by the algorithms for various numbers of nodes and various sampling probabilities p.

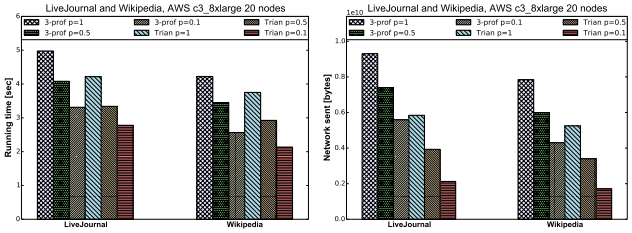


Figure 7: AWS `c3_8xlarge` cluster with 20 nodes. 3-prof vs. trian results for LiveJournal and Wikipedia datasets (average of 3 runs). (a) – Running time for both graphs for various sampling probabilities p. (b) – Network bytes sent by the algorithms for both graphs for various sampling probabilities p.

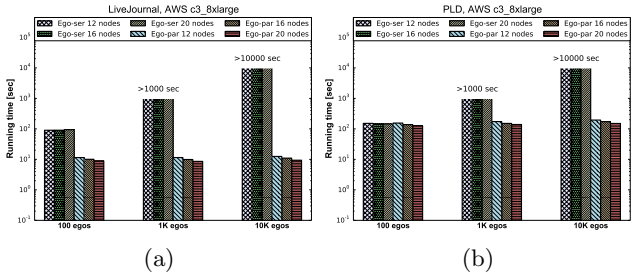


Figure 8: AWS `c3_8xlarge` cluster. Ego-par vs. Ego-ser results for LiveJournal and PLD datasets (average of 5 runs). Running time of Ego-par scales well with the number of ego centers, while Ego-ser scales linearly.

on the Asterix machine. For both graphs, we see a very good scaling of EGO-PAR, while the running time of EGO-SER scales linearly with the size of the ego vertices list.

6. CONCLUSIONS

In summary, we have reduced several 3-profile problems to triangle and 4-clique finding in a graph engine framework. Our concentration theorem and experimental results confirm that local 3-profile estimation via sub-sampling is comparable in runtime and accuracy to local triangle counting.

This paper offers several directions for future work. First, both the local 3-profile and the ego 3-profile can be used as features to classify vertices in social or bioinformatic networks. Additionally, we hope to extend our theory and algorithmic framework to larger subgraphs, as well as special classes of input graphs. Our edge sampling Markov chain and unbiased estimators should easily extend to $k > 3$. Equations in (17) are useful to count local or global 4-profiles in a centralized setting, as shown recently in [17, 36]. Tractable distributed algorithms for $k > 3$ using similar edge pivot equations remain as future work. Our observed dependence on 4-clique count suggests that an improved graph engine-based clique counting subroutine will improve the parallel algorithm's performance.

Acknowledgements

We would like to thank the anonymous reviewers for their useful comments. The authors also acknowledge support from NSF CCF-1344364, NSF CCF-1344179, ARO YIP W911NF-14-1-0258, DARPA XDATA, and research gifts by Google and Docomo.

7. REFERENCES

[1] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella. Graph Sample and Hold : A Framework for Big-Graph Analytics Categories and Subject Descriptors. In *KDD*, 2014.

[2] N. Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, Mar. 1997.

[3] Amazon web services. http://aws.amazon.com, 2015.

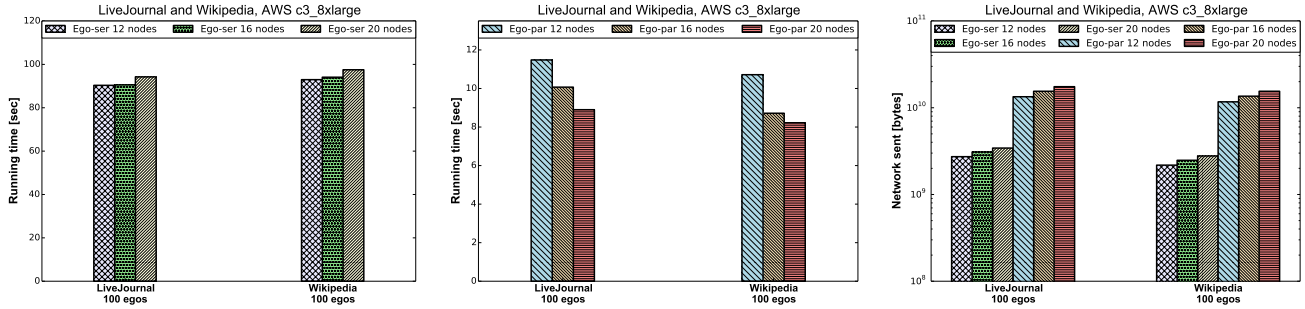
[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, 2008.

[5] M. A. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan. GUISE: Uniform Sampling of Graphlets for Large Graph Analysis. *IEEE 12th International Conference on Data Mining*, pages 91–100, Dec. 2012.

[6] C. Borgs, J. Chayes, and K. Vesztergombi. Counting Graph Homomorphisms. *Topics in Discrete Mathematics*, pages 315–371, 2006.

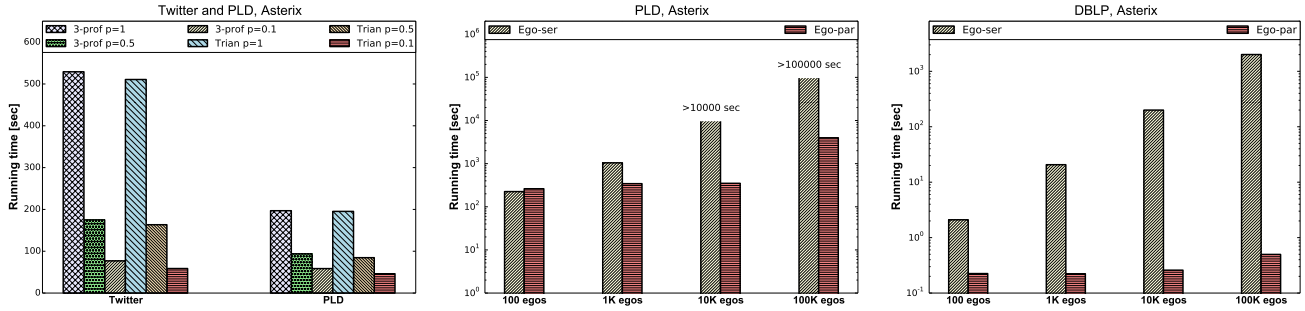
[7] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *Proc. 17th ACM SIGKDD*, page 672, New York, New York, USA, 2011. ACM Press.

[8] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011.



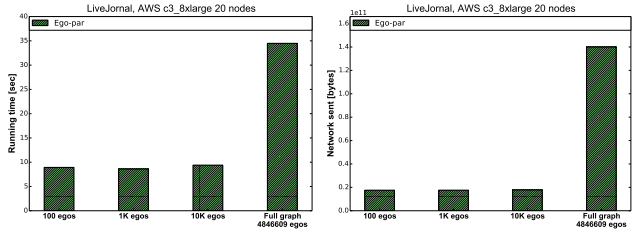
(a) (b) (c)

Figure 9: AWS `c3_8xlarge` cluster. Ego-par vs. Ego-ser results for LiveJournal and Wikipedia datasets (average of 5 runs). Running time of Ego-par decreases with the number of machines due to its parallel design. Running time of Ego-ser does not decrease with the number of machines due to its iterative nature. Network usage increases for both algorithms with the number of machines.



(a) (b) (c)

Figure 11: Asterix machine. Results for Twitter, PLD, and DBLP datasets. (a) – Running time of 3-prof vs. trian for various sampling probabilities p. (b,c) – Running time of Ego-par vs. Ego-ser for various number of ego centers. Results are averaged over 3, and 3, and 10 runs, respectively.



(a) (b)

Figure 10: AWS `c3_8xlarge` cluster with 20 nodes. Ego-par results for LiveJournal dataset (average of 5 runs). The algorithm scales well for various number of ego centers and even full ego centers list. (a) – Running time. (b) – Network bytes sent by the algorithm.

[9] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs, 2015.

[10] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 17–30, 2012.

[11] W. Han and J. Lee. Turbo$_{iso}$: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. In *SIGMOD*, pages 337–348, 2013.

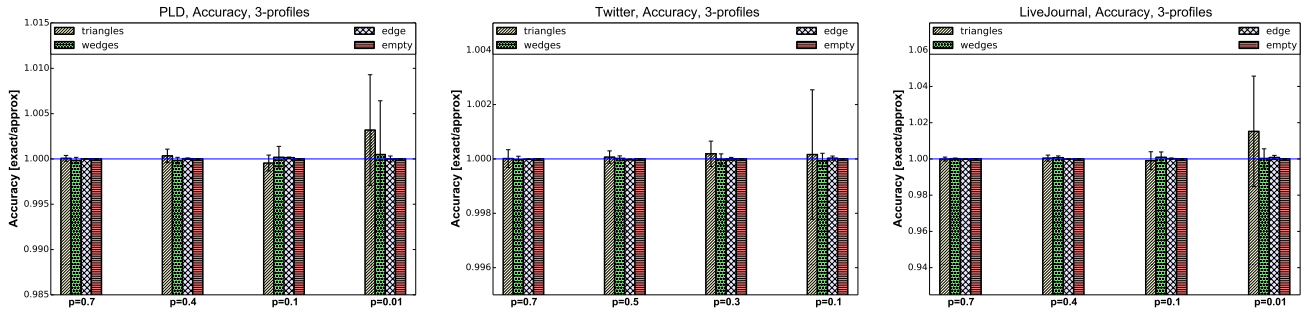
[12] F. Hormozdiari, P. Berenbrink, N. Przulj, and S. C. Sahinalp. Not All Scale-free Networks are Born Equal: The Role of the Seed Graph in PPI Network Evolution. *PLoS Computational Biology*, 3(7):e118, July 2007.

[13] T. Hočevar and J. Demšar. A Combinatorial Approach to Graphlet Counting. *Bioinformatics*, 30(4):559–65, Feb. 2014.

[14] M. Jha, C. Seshadhri, and A. Pinar. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. 2014.

[15] J. H. Kim and V. H. Vu. Concentration of Multivariate Polynomials and Its Applications. *Combinatorica*, 20(3):417–434, 2000.

[16] T. Kloks, D. Kratsch, and H. Müller. Finding and Counting Small Induced Subgraphs Efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000.



(a) (b) (c)

Figure 12: Global 3-profiles accuracy achieved by 3-prof algorithm for various graphs and for each profile count. Results are averaged over 5, and 5, and 10 iterations, respectively. Error bars indicate 1 standard deviation. The metric is a ratio between the exact profile count (when $p = 1$) and the given output for $p < 1$. All results are very close to the optimum value of 1.

[17] M. Kowaluk, A. Lingas, and E.-M. Lundell. Counting and Detecting Small Subgraphs via Equations. *SIAM Journal of Discrete Mathematics*, 27(2):892–909, 2013.

[18] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Proc. 19th International World Wide Web Conference*, pages 591–600, 2010.

[19] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. *Proc. VLDB Endowment*, 6(2):133–144, Dec. 2012.

[20] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[21] L. Lovász. *Large Networks and Graph Limits*, volume 60. American Mathematical Soc., 2012.

[22] D. Marcus and Y. Shavitt. RAGE - A Rapid Graphlet Enumerator for Large Networks. *Computer Networks*, 56(2):810–819, Feb. 2012.

[23] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer. Graph structure in the web – revisited. In *Proc. 23rd International World Wide Web Conference, Web Science Track*. ACM, 2014.

[24] D. O'Callaghan, M. Harrigan, J. Carthy, and P. Cunningham. Identifying Discriminating Network Motifs in YouTube Spam. Feb. 2012.

[25] R. Pagh and C. E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

[26] N. Przulj. Biological Network Comparison Using Graphlet Degree Distribution. *Bioinformatics*, 23(2):177–183, 2007.

[27] N. Satish, N. Sundaram, M. A. Patwary, J. Seo, J. Park, M. A. Hassaan, S. Sengupta, Z. Yin, and P. Dubey. Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets. In *SIGMOD*, pages 979–990, 2014.

[28] T. Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. PhD thesis, 2007.

[29] C. Seshadhri, A. Pinar, and T. G. Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proc. SDM*, pages 10–18, 2013.

[30] N. Shervashidze, K. Mehlhorn, and T. H. Petri. Efficient Graphlet Kernels for Large Graph Comparison. In *AISTATS*, pages 488–495, 2009.

[31] S. Suri and S. Vassilvitskii. Counting Triangles and the Curse of the Last Reducer. In *Proc. 20th International World Wide Web Conference*, page 607. ACM Press, 2011.

[32] C. E. Tsourakakis. Fast Counting of Triangles in Large Real Networks: Algorithms and Laws. In *IEEE International Conference on Data Mining*, 2008.

[33] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: Counting Triangles in Massive Graphs with a Coin. In *SIGKDD*, 2009.

[34] C. E. Tsourakakis, M. Kolountzakis, and G. L. Miller. Triangle Sparsifiers. *Journal of Graph Theory and Applications*, 15(6):703–726, 2011.

[35] J. Ugander, L. Backstrom, M. Park, and J. Kleinberg. Subgraph Frequencies: Mapping the Empirical and Extremal Geography of Large Graph Collections. In *Proc. World Wide Web Conference*, 2013.

[36] V. V. Williams, J. Wang, R. Williams, and H. Yu. Finding Four-Node Subgraphs in Triangle Time. *SODA*, pages 1671–1680, 2014.

[37] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *International Conference on Data Mining*, 2002.