2024 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 22–25, 2024, LONDON, UK

COUNTING TRIANGLES OF GRAPHS VIA MATRIX PARTITIONING

Georgios Kollias[†], *Vassilis Kalantzis*[†], *Lior Horesh*[†], *Shashanka Ubaru*[†], *and Panagiotis A. Traganitis*[*]

[†]IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY, USA
[*]Dept. of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, USA

2024 IEEE 34th International Workshop on Machine Learning for Signal Processing (MLSP) | 979-8-3503-7225-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/MLSP58920.2024.10734796

ABSTRACT

Counting the number of triangles is an important task in the computation of several network-related metrics such as transitivity ratio, link recommendation, near-clique subgraph detection, and clustering coefficient. This contribution presents a new algorithm based on non-overlapping subgraph decomposition and matrix partitioning. The part of the adjacency matrix associated with each submatrix is replaced by a low-rank approximation resulting to complexity savings. The proposed algorithm is also suitable for high-latency architectures as well as graphs whose entries are updated over time. Several practical and theoretical aspects are discussed while numerical experiments on real-world graphs demonstrate the potential of the proposed algorithm.

Index Terms— Triangle counting, graph, approximate counting, domain decomposition

1. INTRODUCTION

Graph triangles are simple yet important motifs [1] that capture the fundamental notion of transitivity in networks: if a vertex α is connected to a vertex β and β to a vertex γ, is it also the case that α is connected to γ? Counting triangles of graphs can provide invaluable information about the structural properties of networks and is an integral variable in graph analytics metrics such as the ubiquitous clustering coefficient and the transitivity ratio [2, 3, 4]. In [5] the idea of associating a common theme to subgraphs in the Web characterized by denser clusters of triangles is introduced. Such regions with high clustering coefficients, or high curvature, have been exploited to automatically cluster and annotate groups of genes or samples in the study of networks of co-expressed genes [6]. In [7], the distributions of clustering coefficients for spam and non-spam Web pages are found to be statistically different and so they are proposed as a tool for spam detection. Triangles are abundant in both social and biological networks: clustering coefficients in protein interaction networks can correlate their functional classes with topological properties [8]. Leskovec et al. [9] conduct a microscopic analysis of the edge-by edge evolution of large online social networks, discover that newly arriving edges tend to close triangles and faithfully reproduce this behavior in their network evolution model. In [10], the role of a user in online forum discussions is associated with triangle counts of its local network. Triangles also play a prominent role in topological signal processing with signals defined over simplices on the graph [11, 12].

Algorithms for *exact counting* are generally based on one of three ideas: (i) iterating through vertex triplets, (ii) linear algebra operations, and (iii) intersecting the adjacency lists of two connected vertices. For an n-vertex graph with m edges, a practical counting algorithm traverses all vertices (nodes) and for each pair of neighbors it checks whether they are connected by an edge, incurring a complexity $O(nd_{max}^2)$ for maximum degree d_{max}. Another option is to iterate over all m edges, and for each edge u, v acccumulate the cardinality of the intersection for the adjacency lists of its endpoints [13]. In [14] an algorithm with complexity $O(m^{3/2})$, based on rooted spanning trees is presented; similar complexities are attained in [15], leveraging the notion of graph arboricity. Direct Matrix-Matrix multiplications yield triangle counts along the diagonal of A^3, with A the graph adjacency matrix, and results in complexity $O(n^\omega)$ where ω is the matrix multiplication exponent; in [16] $\omega < 2.376$. A theoretical state-of-the-art algorithm, termed `AYZ`, for exact triangle counting which runs in $O(m^{\frac{2\,\omega}{\omega+1}}) = O(m^{1.41})$, where $\omega < 2.376$ is the matrix multiplication exponent, was advocated in [17]. Triangle counting implementations ranging from shared-memory machines to large distributed-memory clusters can be found in [18, 19, 20, 21, 22, 23, 24].

The computational complexity incurred by exact counting methods as well as the rise of applications based on streaming models motivates the need for *approximate counting* methods. Algorithms for *approximate counting* are often driven by the application scenario, such as the streaming provision or the distribution of edges and the particular spectral graph properties or processing budget (space/time) limitations. The work in [25] applies randomized projections for the numerical estimation of the trace of the third power of the graph adjacency matrix, using exclusively Matrix-Vector operations. In [26, 27] leading eigenvalues of the graph adjacency matrix are used to approximate triangle counts and this idea is further coupled with a preprocessing stage for uniform edge sampling and sparsification in [3]. The study of uniform edge sampling as a specialized sparsification framework (to which any triangle counting algorithm can then be plugged into) is developed in depth in [28]. This idea is revisited in [29] in conjunction with vertex grouping, which was also used in [17].

Triangle counting has been mapped to multiple programming environments and platforms, with carefully optimized computational primitives for maximizing its efficiency. It is a key benchmarking kernel[1] with implementations ranging from shared-memory machines to large distributed-memory clusters [30, 18, 31, 19, 20, 21, 22, 23, 24].

1.1. Contributions

In this paper we undertake a linear algebra viewpoint and propose an algorithm that leverages matrix partitionings by dividing the adjacency matrix into a 2-dimensional (2D) grid of submatrices. Counting of graph triangles can be then decomposed into a series of matrix trace

Emails: gkollias@us.ibm.com, vkal@ibm.com, lhoresh@us.ibm.com, shashanka.ubaru@ibm.com, traganit@msu.edu
The work of P. Traganitis is supported by NSF grant 2312546.

[1] `https://graphchallenge.mit.edu/champions`

computations involving matrix products. Instead of explicitly forming these matrix products, we rather approximate them via replacing one or more matrices via a rank-k truncated SVD. This converts sparse matrix multiplications to products between sparse matrices and dense vectors, potentially leading to computational cost savings. The main contributions of this paper are as follows:

- We analyze triangle counting via subgraph decomposition from a matrix-based perspective. Following this analysis, a novel approximate counting algorithm that leverages low-rank decomposition of the subgraph coupling is proposed. The proposed algorithm naturally lends itself for parallelization in 2D processor grids since the low-rank approximation of each submatrix can be performed embararrassingly parallel.
- Updates of the graph (adding/deleting new edges) do not require a partial spectral factorization update of the entire adjacency matrix. Instead, only the truncated SVD of the modified submatrices needs to be updated, e.g., [32], resulting to fast updates of the number of triangles. The same holds for graph augmentations (adding new vertices).

2. TRIANGLE COUNTING AND SUBGRAPH DECOMPOSITION

Consider a graph $\mathcal{G} := (V, E, A)$, where V denotes the set of vertices of the graph, $E := \{\{x, y\} : (x, y) \in V \times V \text{ and } x \neq y\}$ denotes the set of edges (without loops), and $A \in \mathbb{R}^{|V|\times|V|}$ denotes the so-called adjacency matrix whose (j, k) entry is equal to one for each edge $\{j, k\} \in E$ and zero otherwise. For convenience let $n = |V|$. We assume that the vertex set V of the graph $\mathcal{G}$ is partitioned into $p \in \mathbb{N}$ induced subgraphs $\mathcal{G}_{i=1,\ldots,p}$ with vertex sets $\{V_i\}_{i=1}^p$, such that $V_k \cap V_j = 0,\ k \neq j$, and $V_1 \cup \ldots \cup V_p = V$, i.e., each vertex of $\mathcal{G}$ belongs to one and only one subgraph. Such partitions of a graph can be either computed via an algebraic partitioner, e.g., METIS [33], or via performing breadth-first search and assigning each subgraph a set of roughly n/p contiguously labeled vertices.

Definition 1. *The matrix A can be decomposed as*

$$A = D + F,$$

where

$$D = \begin{pmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_p \end{pmatrix},\ F = \begin{pmatrix} 0 & E_{12} & \ldots & E_{1p} \\ E_{12}^T & 0 & \ldots & E_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ E_{1p}^T & E_{2p}^T & \ldots & 0 \end{pmatrix}.$$

The $n_i \times n_i$ matrix D_i denotes the coupling between the n_i vertices of subgraph $\mathcal{G}_i$. Similarly, the $n_i \times n_j$ matrix E_{ij} denotes the coupling between the vertices of subgraphs $\mathcal{G}_i$ and $\mathcal{G}_j$. The (α, β) entry of the matrix E_{ij} is nonzero if and only if vertex α of $\mathcal{G}_i$ is connected to vertex β of $\mathcal{G}_j$. Similarly, the (α, β) entry of the matrix D_i is nonzero if and only if vertex α of $\mathcal{G}_i$ is connected to vertex β of $\mathcal{G}_i$.

The k-th power A^k of the adjacency matrix A counts paths of length k between vertices, i.e., the (α, β)-th entry of A^k is equal to the number of paths of length k between vertices α and β. A graph triangle is a closed walk that crosses exactly three unique vertices. The total number of triangles that start/end at vertex i is equal to the i-th diagonal entry of the matrix A^3, and thus the total number of graph triangles is equal to $\mathbf{Tr}\left(A^3\right)/6$, where the operator $\mathbf{Tr}\left(.\right)$ denotes the matrix trace. The scalar division results from the fact that the term $\mathbf{Tr}\left(A^3\right)$ includes the triangle formed by vertices (α, β, γ) three times (each one starting from a separate vertex) and for both possible directions of each edge.

For each subgraph $\mathcal{G}_i$, three types of triangles can be defined depending on how many triangle vertices are located within its boundaries. Since a triangle has three vertices, we will denote by $T_{i,1},\ T_{i,2}$, and $T_{i,3}$, the number of triangles that have exactly one, two, and three vertices located in $\mathcal{G}_i$. A visualization of these three different types of triangles is shown in Figure 1.

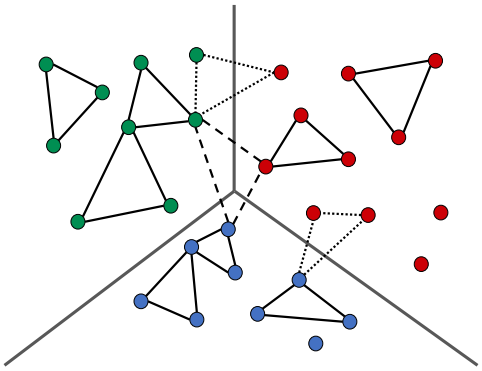

Fig. 1: Three different types of triangles. For each subgraph, $T_{i,1}$, $T_{i,2}$, and $T_{i,3}$ are represented by dashed, dotted, and solid lines, respectively.

Proposition 2. *Let A be written as in Definition 1. Then,*

$$\mathbf{Tr}\left(A^3\right) = \mathbf{Tr}\left(D^3\right) + \mathbf{Tr}\left(F^3\right) + 3\mathbf{Tr}\left(FDF\right).$$

Proof. Expanding $A^3 = (D + F)^3$ yields

$$\begin{aligned}\mathbf{Tr}\left(A^3\right) = \mathbf{Tr}\left(D^3\right) + \mathbf{Tr}\left(F^3\right) + \mathbf{Tr}\left(DFD\right) + \mathbf{Tr}\left(FD^2\right) \\ + \mathbf{Tr}\left(F^2D\right)\mathbf{Tr}\left(D^2F\right) + \mathbf{Tr}\left(FDF\right) + \mathbf{Tr}\left(DF^2\right).\end{aligned}$$

The proof follows by invoking the cyclic property of the trace operator $\mathbf{Tr}\left(DF^2\right) = \mathbf{Tr}\left(F^2D\right) = \mathbf{Tr}\left(FDF\right)$ and $\mathbf{Tr}\left(D^2F\right) = \mathbf{Tr}\left(FD^2\right) = \mathbf{Tr}\left(DFD\right) = 0$. □

Note that $\mathbf{Tr}\left(DFD\right) = 0$, since a random walker can cross $\mathcal{G}_i$ to $\mathcal{G}_j$ but can not return back to $\mathcal{G}_i$ to close the triangle. The matrix D_i couples vertices of subgraph $\mathcal{G}_i$, and thus the term $\mathbf{Tr}\left(D_i^3\right)$ counts the number of triangles that have all three vertices in $\mathcal{G}_i$. Therefore, the term $\mathbf{Tr}\left(D^3\right) = \sum_{i=1,\ldots,p} \mathbf{Tr}\left(D_i^3\right)$ is equal to six times the sum of triangles that have all three vertices located in the same subgraph, i.e., $\mathbf{Tr}\left(D^3\right) = 6\sum_{i=1}^p T_{i,1}$. Similarly, it follows $\mathbf{Tr}\left(FDF\right) = 6\sum_{i=1}^p T_{i,2}$ and $\mathbf{Tr}\left(F^3\right) = 6\sum_{i=1}^p T_{i,3}$.

Definition 3. *Let the set $\mathcal{N}(i)$ denote the list of all subgraphs connected to subgraph $\mathcal{G}_i$ through at least one edge, i.e., $\mathcal{N}(i)$ includes all $\mathcal{G}_j$ such that $E_{ij}\mathbf{1} \neq 0$, where $\mathbf{1}$ denotes the vector of all ones of appropriate dimension.*

Proposition 4. *Let F be expressed as in Definition 1. Then:*

$$\mathbf{Tr}\left(F^3\right) = 6 \sum_{\substack{i<j<k \\ j\in\mathcal{N}(i) \\ i\in\mathcal{N}(k) \\ k\in\mathcal{N}(j)}} \mathbf{Tr}\left(E_{ij}E_{jk}E_{ki}\right). \quad (1)$$

Proof. The term $\mathbf{Tr}\left(F^3\right)$ counts the number of triangles that have no more than one vertex per subgraph. Thus, for any three subgraphs $\mathcal{G}_i,\ \mathcal{G}_j$, and $\mathcal{G}_k$, that are pairwise connected, we need to compute the number of walks of length three that cross all three subgraphs and start/end at the same subgraph. Starting from a vertex in $\mathcal{G}_i,\ i < j < k$, a walker needs to pass to $\mathcal{G}_j$ (multiplication by E_{ij}), then pass to $\mathcal{G}_k$ (multiplication by $E_{jk} = E_{kj}^T$), and finally pass back to $\mathcal{G}_i$ to close the triangle (multiplication by $E_{ki} = E_{ik}^T$). Starting from $\mathcal{G}_j$ or $\mathcal{G}_k$ yields the same result, i.e.,

$$\mathbf{Tr}\left(E_{ij}E_{jk}E_{ki}\right) = \mathbf{Tr}\left(E_{ji}E_{ik}E_{kj}\right) = \mathbf{Tr}\left(E_{kj}E_{ji}E_{ik}\right).$$

The second part of the proof consists of a simple verification. □

Proposition 5. *Let matrices D and F be expressed as in Definition 1. Then,*

$$\mathbf{Tr}\left(FDF\right) = 2\mathbf{Tr}\left(\sum_{i=1}^{p}\sum_{\substack{i<j\\ j\in\mathcal{N}(i)}} E_{ij}D_jE_{ij}^T\right). \tag{2}$$

Proof. The term $\mathbf{Tr}\left(FDF\right)$ counts the number of triangles formed between exactly two adjacent subgraphs $\mathcal{G}_i$ and $\mathcal{G}_j$. Starting from a vertex in $\mathcal{G}_i$, a walker needs to pass to $\mathcal{G}_j$ (multiplication by E_{ij}), connect with another vertex in $\mathcal{G}_j$ (multiplication by D_j), and finally pass back to $\mathcal{G}_i$ to close the triangle (multiplication by E_{ij}^T). The multiplication by "2" stems by the constraint $i < j$ since we skip counting the same triangles starting from $\mathcal{G}_j$. □

3. APPROXIMATING THE NUMBER OF TRIANGLES VIA TRUNCATED SVD

Following the above discussion, computing $\mathbf{Tr}\left(A^3\right)$ requires the computation of $\mathbf{Tr}\left(D^3\right)$, $\mathbf{Tr}\left(F^3\right)$, and $\mathbf{Tr}\left(FDF\right)$. Since the matrix D is block-diagonal, the computation of $\mathbf{Tr}\left(D^3\right)$ breaks into p independent triangle counting problems with the i-th problem concerned with the computation of the number of triangles in $\mathcal{G}_i$,, i.e. $\mathbf{Tr}\left(D_i^3\right)$. The default option is to compute each $\mathbf{Tr}\left(D_i^3\right)$ via an exact counting algorithm such as the ones mentioned in Sec. 1. Indeed, when p is large relative to n one expects $n_i \ll n$ and thus computing the number of triangles associated with $\mathcal{G}_i, i = 1, \ldots, p$, introduces a relatively small overhead. Alternatively, each term $\mathbf{Tr}\left(D_i^3\right)$ can be approximated using the truncated eigendecomposition of D_i by $\mathbf{Tr}\left(U_{i,k}\Lambda_{i,k}^3U_{i,k}^T\right)$. $\Lambda_{i,k}$ is a diagonal matrix holding the k largest modulus eigenvalues of D_i and $U_{i,k}$ denotes the corresponding eigenvectors. Since the matrix D_i is symmetric, the leading k eigenpairs of D_i can be computed via the Lanczos algorithm [34]. This approach is known as `EigenTriangle` [3].

The next task is the computation of the expressions in (1) and (2). Without loss of generality, let $E_{ij} = \widehat{U}_{ij,k}\widehat{\Sigma}_{ij,k}\widehat{V}_{ij,k}^T$ denote the rank-k truncated SVD of the matrix E_{ij}. Then, we can approximate the number of triangles for which at least one edge connects subgraphs $\mathcal{G}_i$ and $\mathcal{G}_j$ as follows.

Proposition 6. *Let $E_{ij} = \widehat{U}_{ij,k}\widehat{\Sigma}_{ij,k}\widehat{V}_{ij,k}^T$, $\chi_{ijk} = \mathbf{Tr}\left(E_{ij}E_{jk}E_{ki}\right)$, and $\widetilde{\chi}_{ijk}$ denote the approximation of χ_{ijk} if E_{ij} is replaced by its rank-k truncated SVD, where $\widehat{\sigma}_{ij,k+1}$ denotes the largest non-computed singular value of E_{ij}. Then,*

$$\|\chi_{ijk} - \widetilde{\chi}_{ijk}\|_2 \leq \widehat{\sigma}_{ij,\kappa+1}\|E_{jk}\|_2\|E_{ki}\|_2.$$

Similarly, let $\psi_{ij} = E_{ij}D_jE_{ji}^T$ and replace E_{ij} by its rank-k truncated SVD. Then,

$$\|\psi_{ij} - \widetilde{\psi}_{ij}\|_2 \leq \widehat{\sigma}_{ij,\kappa+1}\|E_{ij}\|_2\|D_j\|_2.$$

Proof. Replacing E_{ij} by $\widehat{U}_{ij,k}\widehat{\Sigma}_{ij,k}\widehat{V}_{ij,k}^T$ gives:

$$E_{ij}E_{jk}E_{ki} = \widehat{U}_{ij,k}\widehat{\Sigma}_{ij,k}\widehat{V}_{ij,k}^TE_{jk}E_{ki}.$$

The proof concludes by taking norms on both sides and recalling that $\|\widehat{V}_{ij,k}\|_2 = \|\widehat{U}_{ij,k}\|_2 = 1$. The same idea applies to the approximation $\widetilde{\psi}_{ij} = \widehat{U}_{ij,k}\widehat{\Sigma}_{ij,k}\widehat{V}_{ij,k}^TD_jE_{ij}^T$. □

According to Proposition 6, the approximation of χ_{ijk} depends on how close to k the rank of E_{ij} is. Replacing either E_{ik} or E_{jk} with their respective rank-k truncated SVD instead of E_{ij} is equivalent to relabeling subgraphs $\mathcal{G}_i$ and $\mathcal{G}_j$. In principle, it is not possible to determine a priori which one of the three matrices should be replaced by its rank-k truncated SVD; in the proposed algorithm the matrix that provides the smallest upper error bound is chosen.

Algorithm 1 provides a high-level description of the proposed framework. The inputs of Algorithm 1 are the number of subgraphs p and the target rank k. These are assumed user-given; alternatively, both of these inputs can be set after performing greedy optimization on random snapshots (with replacement) of the input graph $\mathcal{G}$. The following theorem bounds the error of using Algorithm 1.

Theorem 7. *Let $\widehat{\sigma}_{ij,\kappa+1} \leq \mu \in \mathbb{R}^+$ for any $i, j = 1, \ldots, p,\ i \neq j$, and denote by $\tau \in \mathbb{N}$ the approximation returned by Algorithm 1. Moreover, let $\lambda_g(A)$ denote the g-th algebraically largest eigenvalue of A and set $\lambda_{\max} = \max_{g=1,\ldots,n}|\lambda_g(A)|$. Then, the absolute approximation error is asymptotically equal to $O(\mu\lambda_{\max}^2)$.*

Proof. Following the results of Proposition 6 and recalling that $\max_{i,j=1,\ldots,p,\ i\neq j}\{\|E_{ij}\|_2, \|D_i\|_2\} \leq \|A\|_2$, we get

$$\|\mathbf{Tr}\left(A^3\right) - \tau\| \leq \mu\left(2\sum_{i=1}^{p}\sum_{\substack{i<j\\ j\in\mathcal{N}(i)}}\|A\|_2^2 + 6\sum_{\substack{i<j<k\\ j\in\mathcal{N}(i)\\ i\in\mathcal{N}(k)\\ k\in\mathcal{N}(j)}}\|A\|_2^2\right).$$

The proof concludes by noticing that $\|A\|_2 = \lambda_{\max}$. □

Following Theorem 7, it follows that

$$\frac{\|\mathbf{Tr}\left(A^3\right) - \tau\|}{\mathbf{Tr}\left(A^3\right)} \leq \mu\left(2\sum_{i=1}^{p}\sum_{\substack{i<j\\ j\in\mathcal{N}(i)}}\lambda_{\max}^2 + 6\sum_{\substack{i<j<k\\ j\in\mathcal{N}(i)\\ i\in\mathcal{N}(k)\\ k\in\mathcal{N}(j)}}\lambda_{\max}^2\right)/\mathbf{Tr}\left(A^3\right),$$

which leads to the following Corollary.

Corollary 8. *The relative approximation error is asymptotically equal to $O\left(\mu\frac{\lambda_{\max}^2}{\sum_{i=1}^n\lambda_i^3}\right)$ and decreases either as μ becomes smaller or $\lambda_{\max}^2 \ll \sum_{i=1}^n\lambda_i^3$.*

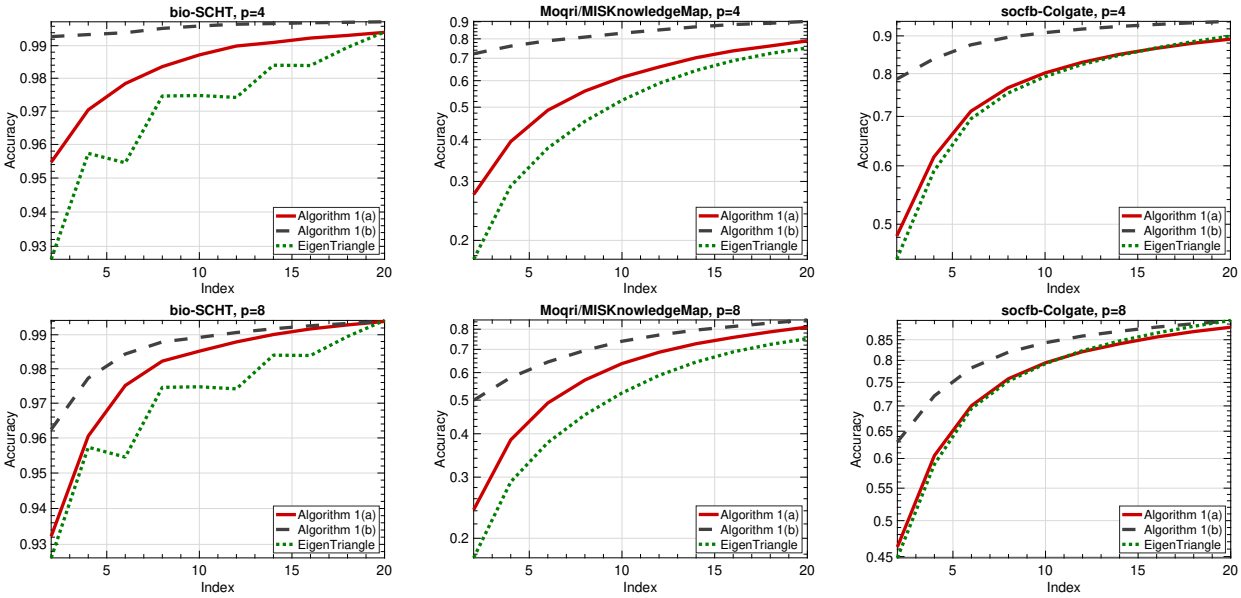

Fig. 2: *Triangle counting approximation accuracy. Top:* $p = 4$*. Bottom:* $p = 8$*.*

Algorithm 1 Triangle counting

1: **Input:** $A \in \{0,1\}^{n\times n},\ p \in \mathbb{N},\ k \in \mathbb{N},\ \tau = 0$
2: **Output:** estimated number of triangles $\tau \in \mathbb{N}$
3: [`In parallel:`] Update $\tau = \tau + \mathbf{Tr}\left(D_i^3\right),\ i = 1, \ldots, p$
4: [`In parallel:`] Compute $\widehat{U}_{ij,k}, \widehat{\Sigma}_{ij,k}, \widehat{V}_{ij,k},\ i \neq j$
5: **Option (a)**: Update τ with the sum of (1) and (2) after replacing E_{ij} by its rank-k truncated SVD; round up τ to the nearest integer
6: **Option (b)**: Update τ with the sum of (1) after replacing E_{ij} by its rank-k truncated SVD; update τ with the exact sum of (2) round up τ to the nearest integer

Algorithm 1 lists two separate options depending on whether (2) is computed exactly or approximately via replacing E_{ij} with its rank-k SVD. Indeed, in contrast to (1), the computations in (2) are embarrassingly parallel across each subgraph and thus the cost of computing the exact value of (2) can be amortized over p different processors. In particular, in our experiments we set k and p using greedy optimization of twenty randomly extracted induced subgraphs of $\mathcal{G}$ where each snapshot was an induced subgraph retaining about 5% of the total vertices of $\mathcal{G}$. We note that setting an optimal value for k is less crucial since it can be adjusted dynamically at no extra costs (i.e., iterative eigenvalue/SVD solvers can exploit previously computed rank-k factorizations).

3.1. Low-rank modifications

Algorithm 1 is highly suitable for updating the number of triangles subject to modifications of the adjacency matrix. Without loss of generality, assume that a modification of A leads to a modification only in the entries of E_{ij}. Then, triangle estimates need to be updated only for those triangles that have edges with endpoints on both $\mathcal{G}_i$ and $\mathcal{G}_j$. In addition to avoiding the update of the entire partial spectral factorization of A, the rank-k updated of each modified submatrix E_{ij} can take place in an embarrassingly parallel fashion. In summary:

1. Update the rank-k truncated SVD of E_{ij} on-the-fly, e.g., [32, 35].
2. Update the triangle estimates in (1) and (2) only for those indices that involve the submatrix E_{ij}, i.e., triangles that have edges with endpoints on both $\mathcal{G}_i$ and $\mathcal{G}_j$.

Another important point is that A need not be available since the SVD update mechanism does not directly involve the adjacency matrix; only the existing rank-k SVD and the location of the modified entries of A is needed.

4. NUMERICAL EXPERIMENTS

Experiments are conducted in a Matlab environment using 64-bit arithmetic on a single core of a computing system equipped with a 2.3 GHz Quad-Core Intel Core i9 processor and 64 GB of system memory. We consider the performance of Algorithm 1 on three separate graphs obtained by the Network Repository [36]: a) `bio-SCHT`, a gene association network with $n = 2,084$ vertices and $m = 63,027$ edges, b) `MISK`, an article similarity network with $n = 2,427$ vertices and $m = 28,511$ edges, and d) `socfb-Colgate`, a Facebook social network with $n = 3,482$ vertices and $m = 155,043$ edges. Note that the number of non-zero entries in the corresponding adjacency matrices is twice the number of edges.

Figure 2 plots the relative error accuracy percentage of the approximation returned by Algorithm 1 for $p = \{4, 8\}$ versus $k = 2, 4, \ldots, 20$. For illustration purposes, we also plot the accuracy percentage returned by `EigenTriangle` which approximates the number of graph triangles via $\sum_{i=1}^{k} \lambda_i^3$. We present the obtained accuracy using both options suggested in Algorithm 1. Naturally, option "Algorithm 1 (b)" achieves higher accuracy since (2) is computed exactly rather than approximated via truncated SVD as in option "Algorithm 1 (a)". For both options, we observe a decline of the obtained accuracy for larger p, since there are more computations that are performed approximately, i.e., the number triangles associated with $\mathbf{Tr}(F^3)$ now increases.

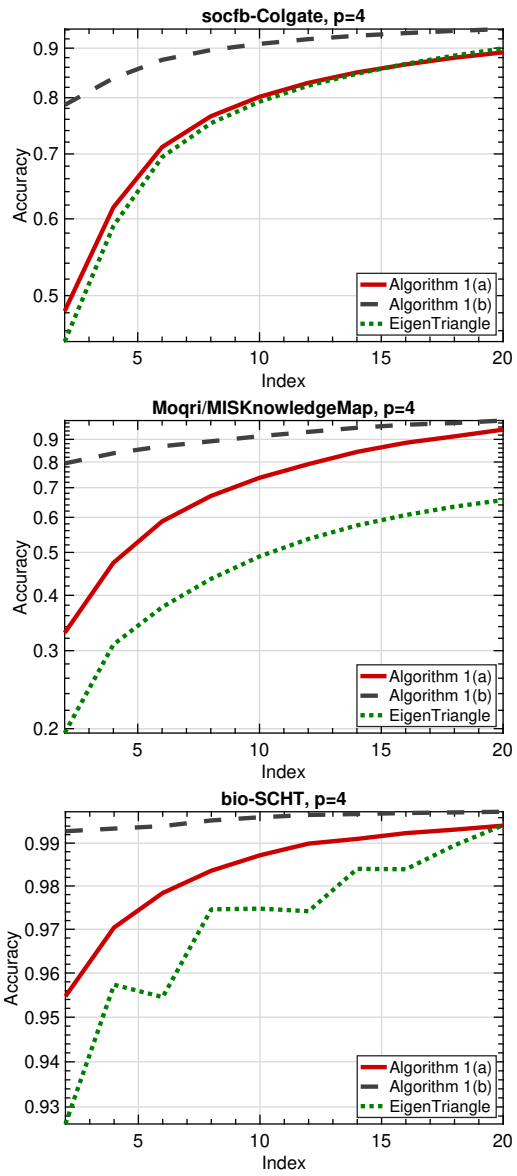

Fig. 3: *Triangle counting approximation accuracy for the graph perturbation scenario. The horizontal axis denotes the dimension of the low-rank approximation at each time-step.*

Next, we consider the problem of triangle counting under graph updates for the graphs `bio-SCHT`, `MISK`, and `socfb-Colgate`. Our experiment is set up as follows. First, we randomly choose half of the edges of the graph and remove them. We then run either Algorithm 1 ($p = 4$) or `EigenTriangle` using a rank $k = 2, 4, \ldots, 20$, and compute the approximate number of triangles of the graph formed by the remaining graph edges. Finally, we add back the edges removed in the previous phase in batches, with each batch containing $\frac{m}{10}$ edges.[2] Therefore, we have exactly five graph updates. For each update, instead of computing partial matrix factorizations from scratch, the rank-k truncated SVD of each E_{ij} is updated on-the-fly via the algorithm in [35]. Similarly, the spectral factorization of `EigenTriangle` is updated via the TRIP algorithm [37]. We repeat the same experiment for a total of ten times, each time with a new random set of removed edges. Figure 3 plots the (averaged) relative error accuracy percentage of the approximation returned by Algorithm 1 and `EigenTriangle` after the last (fifth) update. As expected, both schemes are able to deliver a better triangle counting update as the dimension k of the respective low-rank approximations become more accurate. Recall now, that the update of Algorithm 1 involves only the matrices E_{ij} who overlap with modified graph edges. This is in contrast to TRIP which requires a spectral factorization update of all matrix entries. As a result, sparse graph updates are more likely to favor Algorithm 1 since it becomes more likely that only a small portion of the submatrices of the matrix F will be updated. As in the previous experiment "Algorithm 1 (a)" is, as expected, less accurate than "Algorithm 1 (b)" since the latter computes the quantity in (2) exactly. Finally, both versions of Algorithm 1 are more accurate than `EigenTriangle`, although the latter does not require the use of the partitioning parameter p.

5. CONCLUSION

This paper presented and analysed a domain decomposition algorithm to approximate the number of triangles in graphs. The first step is to decompose the triangle counting problem into three separate tasks where each task is associated with how many triangle vertices reside within a unique subgraph. Computing the number of triangles involving exactly one or two vertices per subgraph can be accomplished independently and in trivial parallelism among each different subgraph. The number of triangles with at most one vertex per subgraph is approximated via low-rank approximation of the coupling matrices. The proposed algorithm can be efficient for networks with higher average degree or networks whose edges are modified over time. As part of our future work we plan to develop a distributed memory implementation of Algorithm 1 via the Message Passing Interface.

6. REFERENCES

[1] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[2] M. E. Newman, "The structure and function of complex networks," *SIAM review*, vol. 45, no. 2, pp. 167–256, 2003.

[3] C. Tsourakakis et al., "Spectral counting of triangles in power-law networks via element-wise sparsification," in *2009 International Conference on Advances in Social Network Analysis and Mining*. IEEE, 2009, pp. 66–71.

[2]The last batch might contain slightly more or less than $\frac{m}{10}$ edges.

[4] J. Gu, C. Song, H. Dai, L. Lu, and M. Liu, "Compact estimator for streaming triangle counting," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[5] J.-P. Eckmann and E. Moses, "Curvature of co-links uncovers hidden thematic layers in the world wide web," *Proceedings of the national academy of sciences*, vol. 99, no. 9, pp. 5825–5829, 2002.

[6] J. Rougemont and P. Hingamp, "DNA microarray data and contextual analysis of correlation graphs," *BMC bioinformatics*, vol. 4, no. 1, pp. 15, 2003.

[7] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient semi-streaming algorithms for local triangle counting in massive graphs," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 16–24.

[8] S.-H. Yook, Z. N. Oltvai, and A.-L. Barabási, "Functional and topological characterization of protein interaction networks," *Proteomics*, vol. 4, no. 4, pp. 928–942, 2004.

[9] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 462–470.

[10] H. T. Welser, E. Gleave, D. Fisher, and M. Smith, "Visualizing the signatures of social roles in online discussion groups," *Journal of social structure*, vol. 8, no. 2, pp. 1–32, 2007.

[11] S. Barbarossa and S. Sardellitti, "Topological signal processing over simplicial complexes," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2992–3007, 2020.

[12] M. T. Schaub, Y. Zhu, J.-B. Seby, T. M. Roddenberry, and S. Segarra, "Signal processing on higher-order networks: Livin' on the edge... and beyond," *Signal Processing*, vol. 187, pp. 108149, 2021.

[13] T. Schank, *Algorithmic Aspects of Triangle-Based Network Analysis*, Ph.D. thesis, 2007.

[14] A. Itai and M. Rodeh, "Finding a minimum circuit in a graph," *SIAM J. Comput.*, vol. 7, no. 4, pp. 413–423, 1978.

[15] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM Journal on computing*, vol. 14, no. 1, pp. 210–223, 1985.

[16] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of symbolic computation*, vol. 9, no. 3, pp. 251–280, 1990.

[17] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997.

[18] A. Azad, A. Buluç, and J. Gilbert, "Parallel triangle counting and enumeration using matrix algebra," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 2015, pp. 804–811.

[19] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, and C. Task, "Counting triangles in massive graphs with mapreduce," *SIAM J. Sci. Comp.*, vol. 36, no. 5, pp. S48–S77, 2014.

[20] T. M. Low et al., "First look: Linear algebra-based triangle counting without matrix multiplication," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–6.

[21] R. Pearce, "Triangle counting for scale-free graphs at scale in distributed memory," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 2017, pp. 1–4.

[22] A. S. Tom and G. Karypis, "A 2d parallel triangle counting algorithm for distributed-memory architectures," in *Proceedings of the 48th International Conference on Parallel Processing*, New York, NY, USA, 2019, ICPP 2019, pp. 45:1–45:10, ACM.

[23] M. M. Wolf et al., "Fast linear algebra-based triangle counting with kokkoskernels," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–7.

[24] Y. Zhang, H. Jiang, F. Wang, Y. Hua, D. Feng, and X. Xu, "LiteTE: Lightweight, communication-efficient distributed-memory triangle enumerating," *IEEE Access*, vol. 7, pp. 26294–26306, 2019.

[25] H. Avron, "Counting triangles in large graphs using randomized matrix trace estimation," in *Workshop on Large-scale Data Mining: Theory and Applications*, 2010, vol. 10, pp. 10–9.

[26] C. E. Tsourakakis, "Fast counting of triangles in large real networks without counting: Algorithms and laws," in *2008 Eighth IEEE International Conference on Data Mining*, Dec 2008, pp. 608–617.

[27] C. E. Tsourakakis, "Counting triangles in real-world networks using projections," *Knowledge and Information Systems*, vol. 26, no. 3, pp. 501–520, 2011.

[28] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 837–846.

[29] M. N. Kolountzakis et al., "Efficient triangle counting in large graphs via degree-based vertex partitioning," in *Algorithms and Models for the Web-Graph*. 2010, pp. 15–24, Springer Berlin Heidelberg.

[30] S. Arifuzzaman, M. Khan, and M. Marathe, "Patric: A parallel algorithm for counting triangles in massive networks," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 2013, pp. 529–538.

[31] J. Cohen, "Graph twiddling in a mapreduce world," *Computing in Science & Engineering*, vol. 11, no. 4, pp. 29, 2009.

[32] V. Kalantzis, G. Kollias, S. Ubaru, A. N. Nikolakopoulos, L. Horesh, and K. Clarkson, "Projection techniques to update the truncated svd of evolving matrices with applications," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5236–5246.

[33] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[34] Y. Saad, *Numerical methods for large eigenvalue problems: revised edition*, vol. 66, Siam, 2011.

[35] H. Zha and H. D. Simon, "On updating problems in latent semantic indexing," *SIAM Journal on Scientific Computing*, vol. 21, no. 2, pp. 782–791, 1999.

[36] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015.

[37] C. Chen and H. Tong, "Fast eigen-functions tracking on dynamic graphs," in *Proceedings of the 2015 SIAM international conference on data mining*. SIAM, 2015, pp. 559–567.