Intersection Representation of Big Data Networks and Triangle Enumeration

Wali Mohammad Abdullah, David Awosoga, and Shahadat Hossain

University of Lethbridge, Lethbridge, Alberta, Canada
{w.abdullah,david.awosoga,shahadat.hossain}@uleth.ca

Abstract. Triangles are an essential part of network analysis, representing metrics such as transitivity and clustering coefficient. Using the correspondence between sparse adjacency matrices and graphs, linear algebraic methods have been developed for triangle counting and enumeration, where the main computational kernel is sparse matrix-matrix multiplication. In this paper, we use an intersection representation of graph data implemented as a sparse matrix, and engineer an algorithm to compute the "k-count" distribution of the triangles of the graph. The main computational task of computing sparse matrix-vector products is carefully crafted by employing compressed vectors as accumulators. Our method avoids redundant work by counting and enumerating each triangle exactly once. We present results from extensive computational experiments on large-scale real-world and synthetic graph instances that demonstrate good scalability of our method. In terms of run-time performance, our algorithm has been found to be orders of magnitude faster than the reference implementations of the **miniTri** data analytics application [18].

Keywords: Intersection Matrix · Local Triangle Count · Forward Degree Cumulative · Forward Neighbours · Sparse Graph · k-count.

1 Introduction

The presence of triangles in network data has led to the creation of many metrics to aid in the analysis of graph characteristics. As such, the ability to count and enumerate these triangles is crucial to applying these metrics and gaining further insights into the underlying composition and distribution of these graphs. Generalizations aside, the applications of triangle counting are as ubiquitous as the triangles themselves, including transitivity ratio - the ratio between the number of triangles and the paths of length two in a graph - and clustering coefficient - the fraction of neighbours for a vertex i of a graph who are neighbours themselves. Other real-life applications of triangle counting include spam detection [4], network motifs in biological pathways [12], and community discovery [13]. However, before any network analysis can be undertaken, the underlying data structure of a graph must be critically examined and understood. An efficient representation of network data will dictate analysis capabilities and improve

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

algorithm performance and data visualization potential [5]. Note that large real-life networks are typically sparse in nature, so efficient computations of these graphs must be able to account for their sparsity and skewed degree distribution [3]. A consistent structure makes linear algebra-based triangle counting methods appealing, and most methods use direct or modified matrix-matrix multiplication, with a notable exception being the implementation of Low et al. [11]. This paper expands upon the preliminary ideas of a poster presentation from the 2021 IEEE Big Data Conference (Big Data) [2], and here we propose an "intersection" representation of network data obtained as a list of edges [17] and based on sparse matrix data structures [8]. Our triangle enumeration algorithm derives its simplicity and efficiency by employing matrix-vector product calculations as its main computational kernel. The local triangle count and edge support information are then acquired from the enumerated triangles obtained as the result of this matrix-vector multiplication.

1.1 K-count Distribution

Application proxies provide a simple yet realistic way to assess the performance of real-life applications' architecture and design. Below, we outline the main components of the miniTri data analytics proxy [18], which we use to demonstrate the effectiveness of our intersection-based graph representation and computation.

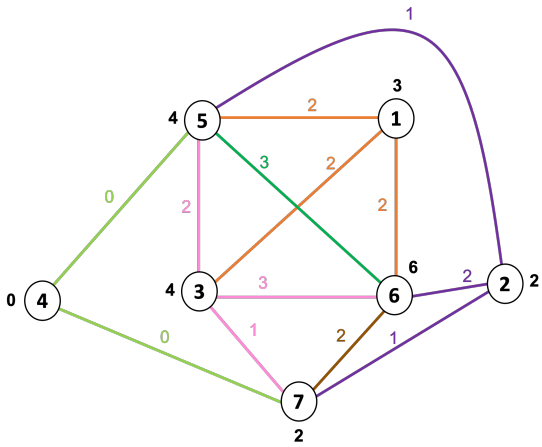
Let $G = (V, E)$ be a connected and undirected graph without multiple edges and self loops, where V denotes the set of vertices and E denotes the set of edges. For $v \in V$ a path of length 2 through v is a sequence of vertices $u-v-w$ such that $e_1 = \{u, v\} \in E$ and $e_2 = \{v, w\} \in E$. Such a length-2 path is termed a *wedge* at vertex v. Let $d(v)$ denote the number of edges *incident* on v, also defined as the number of vertices x such that $\{v, x\} \in E$. The number of wedges in G is then given by $\sum_{v \in V} \binom{d(v)}{2}$. A wedge $u - v - w$ is a *closed wedge* or a *triangle* if $e_3 = \{v, w\} \in E$. Let $\delta(v)$ and $\delta(e)$ denote the number of triangles incident on vertex v and edge $e = \{u, v\}$ respectively. In the literature $\delta(v)$ is known as the *local triangle count* or *triangle degree* of vertex v and $\delta(e)$ is known as the *support* or *triangle degree* of edge $e = \{u, v\}$. We denote by $\Delta(G)$ the number of triangles contained in graph G. Since a triangle is counted at each of its three vertices, we have $\Delta(G) = \frac{1}{3} \sum_{v \in V} \delta(v)$. Let $H = (V', E')$ be a subgraph of G where $|V'| = k$ and each pair of vertices are connected by an edge (H is a k-clique). Then H contains $\binom{k}{3}$ triangles and $\delta(v) = \binom{(k-1)}{2}$ and $\delta(e) = (k-2)$ for $v \in V'$ and $e \in E'$. Let t be a triangle in G and let $\delta(t_x) = \min_x \delta(x)$, where x is a vertex of t and $\delta(t_e) = \min_e \delta(e)$ where e is an edge of t. The *k-count* of triangle t is defined to be the largest k such that

1. $\delta(t_x) \geq \binom{(k-1)}{2}$ and
2. $\delta(t_e) \geq (k-2)$

The main computational task of miniTri is to compute the k-count distribution of the triangles of an input graph. Figure 1 displays an example input graph with 7 vertices and 13 edges. Each vertex i is circled and contains a label that represents

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

its identity. Beside each vertex i is an integer denoting its local triangle count $\delta(i)$, and there is an integer beside each edge $e = \{i, j\}$ denoting its support $\delta(e)$. The graph contains 7 triangles. The table of Figure 1 enumerates the triangles in the graph and displays the local triangle count and support of the vertices and edges together with the k-count of the triangles. Each row of the table lists the vertex labels of a triangle followed by the local triangle count, support, and k-count. There are 4 triangles with k-count value 4 and 3 triangles with k-count value 3. Let ω be the size of the largest clique in G. Then the graph contains at least $\binom{\omega}{3}$ triangles with k-count value of at least ω. Therefore, the k-count distribution can be used to obtain a bound on the size of the largest clique of a graph.



u	v	w	$\delta(u)$	$\delta(v)$	$\delta(w)$	$\delta(e_1)$	$\delta(e_2)$	$\delta(e_3)$	K-Count
1	3	5	3	4	4	2	2	2	4
1	3	6	3	4	6	2	2	3	4
1	5	6	3	4	6	2	2	3	4
2	5	6	2	4	6	1	2	3	3
2	5	7	2	6	2	2	1	2	3
3	5	6	4	4	6	2	3	3	4
3	6	7	4	6	2	3	1	2	3

Fig. 1. K-Count Table for the Example Input Graph

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

The remainder of the paper is organized as follows. In Section 2, we introduce the notion of the intersection representation of network data and our data structure, followed by an illustrative example describing the main ideas in our intersection matrix-based triangle enumeration method. Section 3 outlines the computing environment employed to perform numerical experiments and presents triangle enumeration results on three sets of representative network data. miniTri1 [18] and its successor, which we call miniTri2, are the reference implementations by which we present comparative running times and demonstrate that our method scales very well on massive network data, and can be very flexible in its extensions to the analysis of network characteristics such as truss decomposition [7] and triangle ranking [6]. The paper is summarized in Section 4 with a discussion on future research directions.

2 Intersection Representation of Network Data

Let the vertices in V be labelled $1, 2, \ldots, |V| = n$. Using the labels on the vertices, a unique label can be assigned to each edge $e_k = \{v_i, v_j\}, i < j, k = 1, 2, \ldots, |E| = m$.

The *intersection representation* of graph G is a matrix $X \in \{0,1\}^{m \times n}$ in which for each column j of X there is a vertex $v_j \in V$ and $\{v_i, v_j\} \in E$ whenever there is a row k for which $X(k,i) = 1$ and $X(k,j) = 1$. The rows of X represent the edge list sorted by vertex labels. Therefore, matrix X can be viewed as an assignment to each vertex a subset of m labels such that there is an edge between vertices i and j if and only if the inner product of the columns i and j is 1. Since the input graph is unweighted, the edges are simply ordered pairs, and can be sorted in $O(m)$ time. Unlike the adjacency matrix which is unique (up to a fixed labelling of the vertices) for graph G, there can be more than one *intersection matrix* representation associated with graph G [1]. We exploit this flexibility to store a graph in a structured and space-efficient form.

2.1 Adjacency Matrix-based Triangle Counting

Many existing triangle counting methods use the sparse representation of adjacency matrices in their calculations. The adjacency matrix $A(G) \equiv A \in \{0,1\}^{|V| \times |V|}$ associated with graph G is defined as,

$$A(i,j) = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E, i \neq j \\ 0 & \text{otherwise} \end{cases}$$

It is well known in the literature that the number of closed walks of length $k \geq 0$ are obtained in the diagonal entries of k^{th} power A^k. Therefore, the total number of triangles in a graph $G, \Delta(G)$, is given by the trace of A^3,

$$\Delta(G) = \frac{1}{6} Tr(A^3).$$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

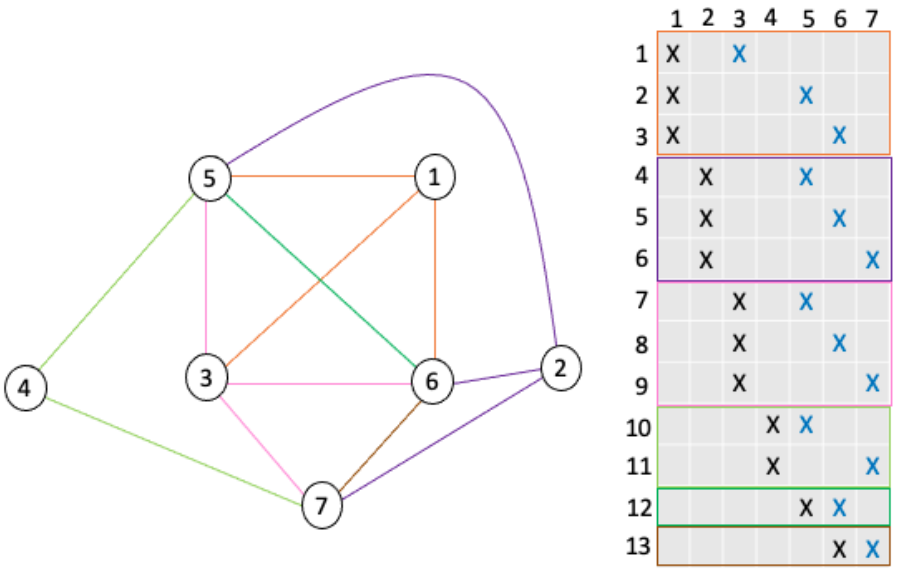


Fig. 2. Intersection Matrix Representation of the Example Input Graph

The factor of $\frac{1}{6}$ accounts for the multiple counting of a triangle (the number of ways closed walks of length 3 can be obtained is $3! = 6$). There is a large body of literature on sparse linear algebraic triangle counting methods based on adjacency matrix representation of the data [5]. miniTri's triangle counting implementation takes the adjacency matrix A of the input graph and creates an incidence matrix B from it [18]. The enumeration and counting of the triangles occur in the overloaded matrix multiplication $C = AB$, where entries in the resultant matrix C with a value of 2 correspond to a completed triangle. This method triple-counts each triangle, once for each vertex, so the final result is divided by 3 to give the total number of triangles in the graph. Since the multiplication of two sparse matrices usually results in a dense matrix, this is a memory intensive process.

2.2 Intersection Matrix-based Triangle Counting

Graph algorithms can be effectively expressed in terms of linear algebra operations [9], and we combine this knowledge with our proposed data representation to count the triangles in a structured three-step method. For vertex i we first find its neighbours $j > i$ such that $\{i, j\} \in E$ by multiplying the submatrix of X consisting of rows corresponding to edges incident on i (let us call them $(i - j)-$rows) by the transpose of the vector of ones of compatible length. A value of 1 in the vector-matrix product indicates that the corresponding vertex j is a neighbour of vertex i.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

Next, we multiply the submatrix of X consisting of columns j identified in the previous step and the rows below the $(i-j)-$rows by a vector of ones of compatible length. A value of 2 in the matrix-vector product indicates a triangle of the form (i, j, j') where j and j' are neighbours of vertex i with $j < j'$. Let k be the row index in matrix X for which the matrix-vector product contains a 2. Then it must be that $X(k, j) = 1$ and $X(k, j') = 1$. Since each row of X contains exactly 2 nonzero entries that are 1, it follows that $\{j, j'\} \in E$. This operation is identical to performing a set intersection on the forward neighbours of vertices j and j'.

The number of triangles in the graph is given by the sum of the number of triangles associated with each vertex as described. Since the edges are represented in sorted order in our algorithm, unlike many other triangle counting methods [18], each triangle is counted exactly once. Figure 2 displays the intersection matrix representation of the input graph X. The triangles of the form $(1, j, j')$ where $j, j' \in \{3, 5, 6\}$ are obtained from the product $X(7 : 13, [3\ 5\ 6]) * \mathbf{1}$, where $\mathbf{1}$ denotes the vector of ones. The product has a 2 at locations corresponding to rows 7, 8, and 12 of X and the associated triangles are $(1, 3, 5), (1, 3, 6)$, and $(1, 5, 6)$. Therefore, there are three triangles incident on vertex 1, and it can be easily verified that the graph contains a total of 7 triangles across all of the vertices.

2.3 Data Structure

In our preliminary implementation, we use two arrays to store useful information that can be computed after we sort the edges. `FDC` (Forward Degree Cumulative) is an array of size n, with elements corresponding to the total number of "forward neighbours" across the vertices of a graph. Forward neighbours are defined as the neighbours of a vertex that have a higher label than the vertex of interest. With the vertices of the graph labelled, finding the forward degree of a vertex j can be calculated as `fd(j) = FDC[j+1] - FDC[j]`. `FN` is an array of size m that stores *which* vertices are the forward neighbours of a vertex j. Using `FN` we can find these forward neighbours of j as `fn(j) = FN[k]`, where k ranges from `FDC[j]` to `FDC[j+1]-1`. The arrays `FDC` and `FN` thus save the vector-matrix products needed to find the forward neighbours. Figure 3 displays the arrays `FDC` and `FN` for the graph of Figure 2.

FN =	3	5	6	5	6	7	5	6	7	5	7	6	7
FDC =	1		4		7		10		12		13		14

Fig. 3. FN and FDC for the Example Graph.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

2.4 Local Triangle Count and Edge Support

As discussed in Section 1, there are many other metrics related to triangle computation that can be found using our intersection matrix data structure. The bases for these metrics are the triangle degrees, which are the number of triangles incident on an edge (edge support) or vertex (local triangle count) of a graph. This is illustrated in Figure 4 as **edgeDeg** and **vertDeg**, respectively, derived from Figure 1.

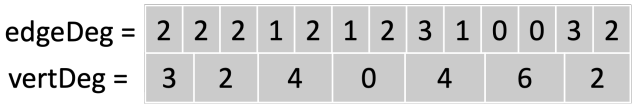


Fig. 4. vertDeg and edgeDeg for the Example Graph.

Let j be the column (vertex) of matrix X (graph G) currently being processed in the **fullCount** algorithm. For each pair of its forward neighbours j' and j'' there is an edge between them if and only if both of the corresponding columns contain a 1 in some row k identifying the triangle (j, j', j''). In terms of the matrix-vector multiplication in line 7 of algorithm **fullCount**, vector T will get updated as $T(k) \leftarrow 2$. Thus the triangle (j, j', j'') can be enumerated and stored instantly. The vertex triangle degrees of each triangle are dynamically updated with this same information, and stored in an array. The edge triangle degrees are stored in a separate array and updated by exploiting the structure of the `FN` and `FDC` arrays in tandem. The entries of the `FDC` array, while primarily used to store the forward degree of a vertex, also contain the edge number (edge id) that the forward neighbourhood of the vertex of interest begins and ends at. Since the sub-arrays in `FN` that correspond to the forward neighbourhood of the vertices are in the same order as the listed edges of the intersection matrix, any edge between two vertices can be identified by first finding the distance between the higher labelled vertex and the beginning of the forward neighbourhood in which it is found (using `FN`), and then adding this distance to the entry in `FDC` that corresponds to the edge of the lower numbered vertex. Finally, the k-count distribution of the triangles is used to give a bound on the maximum clique of a graph [18], and with the triangles enumerated and the edge and vertex triangle degrees computed and stored as shown in Figure 4, the k-count calculations can be quickly computed using the method described Section 1. The algorithm in its entirety is given in the next section.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

2.5 Algorithm

fullCount (X)
Input: Intersection matrix X
1: Calculate FDC ▷ Forward degree cumulative
2: Calculate FN ▷ Forward neighbour
3: $count \leftarrow 0$ ▷ Number of triangles
4: **for** $j = 1$ to $n - 1$ **do** ▷ $j \in V$, where V is the set of vertices
5: $fd \leftarrow FDC[j + 1] - FDC[j]$ ▷ fd is the forward degree of j
6: **if** $fd > 1$ **then** ▷ j has more than one forward neighbour
7: $T = X([FDC(j + 1) : m], fn_j) * \mathbf{1}$
8: $S = \{t \mid T[t] = 2\}$
9: **if** $S \neq \emptyset$ **then**
10: $count \leftarrow count + |S|$
11: **for** $t \in S$ **do**
12: update edgeDeg ▷ Array of triangle edge degrees
13: update vertDeg ▷ Array of triangle vertex degrees
14: $Triangles \leftarrow Triangles \cup t$ ▷ Array that stores enumerated triangles
15: $kCountTable \leftarrow$ computeKCounts($count$, $vertDeg$, $edgeDeg$, $Triangles$)
16: **return** $count$, $vertDeg$, $edgeDeg$, $kCountTable$, and $Triangles$

3 Numerical Results

This section contains experimental results from selected test instances. The first set comprises real-world social networks from the Stanford Network Analysis Project (`SNAP`), obtained from the Graph Challenge website [15]. SNAP is a collection of more than 50 large network datasets containing a large number of nodes and edges, including social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks [10]. The first set of experiments were performed using a Dell Precision T1700 MT PC with a 4th Gen Intel Core I7-4770 Processor (Quad Core HT, with 3.4GHz Turbo and 8GB RAM), running Centos Linux v7.9. The implementation language was *C++* and the code was compiled using $-O3$ optimization flag with a *g++* version 4.4.7 compiler. Performance times are reported in seconds and were averaged over three runs where possible, using the following implementation abbreviations: *mt1* for miniTri1, *mt2* for miniTri2, and *int* for our intersection algorithm.

Figure 5 shows the speedups of our algorithm versus the two reference miniTri implementations on these real-world instances. The speedups are a unitless measurement defined as the ratio of the miniTri counting time divided by that of our algorithm. For the triangle counting algorithms, our speedups ranged from 22× to an impressive 1383× over miniTri1, and from 16× to 642× over miniTri2, with two instances ("flickrEdges" and "Cit-Patents") failing to compute with miniTri2. Instances with an "*" had speedups greater than 650× against miniTri1 and were cut off from the figure for ease of viewing.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

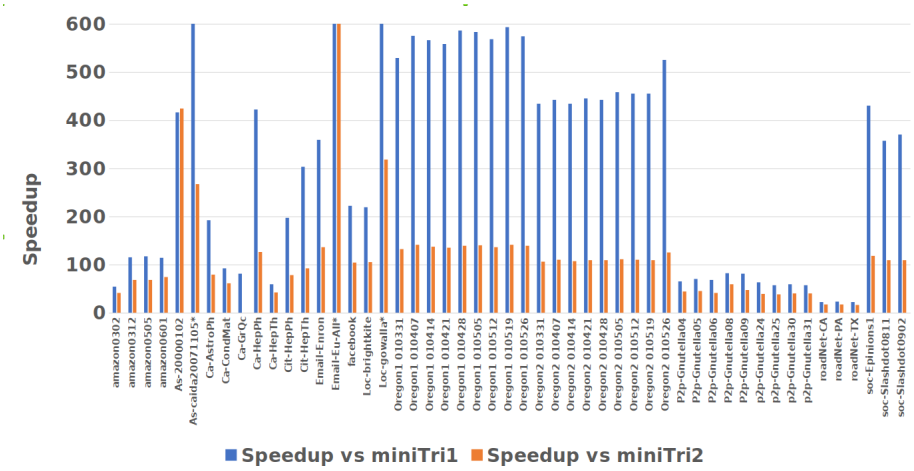


Fig. 5. Comparing our Intersection Algorithm with both miniTri implementations on Large Real World Networks

Table 1. Comparing Our Intersection Algorithm with miniTri on Large Synthetic Networks.

Graph Characteristics				Time in Seconds	
Name	$\|V\|$	$\|E\|$	$\Delta(G)$	mt1	int
graph500-scale18-ef16	262144	4194304	82287285	17440	9.357
graph500-scale19-ef16	524288	8388608	186288972	49211.8	25.21
graph500-scale20-ef16	1048576	16777216	419349784	197456	72.34
graph500-scale21-ef16	2097152	33554432	935100883	N/A	171.2
graph500-scale22-ef16	4194304	67108864	2067392370	N/A	481.43
graph500-scale23-ef16	8388608	134217728	4549133002	N/A	1340.05
graph500-scale24-ef16	16777216	268435456	9936161560	N/A	3317.15
graph500-scale25-ef16	33554432	536870912	21575375802	N/A	7959.39

Table 1 compares our algorithm performance on large synthetic test instances from GraphChallenge to miniTri1 (miniTri2 was only able to compute the first instance and thus omitted). "N/A"denotes instances where miniTri1 timed out after four days of computation. Due to the large sizes of this second set of instances, they were run on the large High Performance Computing system (Graham cluster) at Compute Canada. On the first 3 instances, our method is over 1800 times faster than miniTri1, and the relative performance improves with increasing instance size, further demonstrating the scalability of our triangle counting algorithm.

Figure 6 demonstrates our algorithm's performance on relatively dense brain networks from the Network Repository[14], back in the Linux environment. These graphs have between 15 and 268 million edges and up to 42 trillion triangles, and neither miniTri implementation was able to provide results for any

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

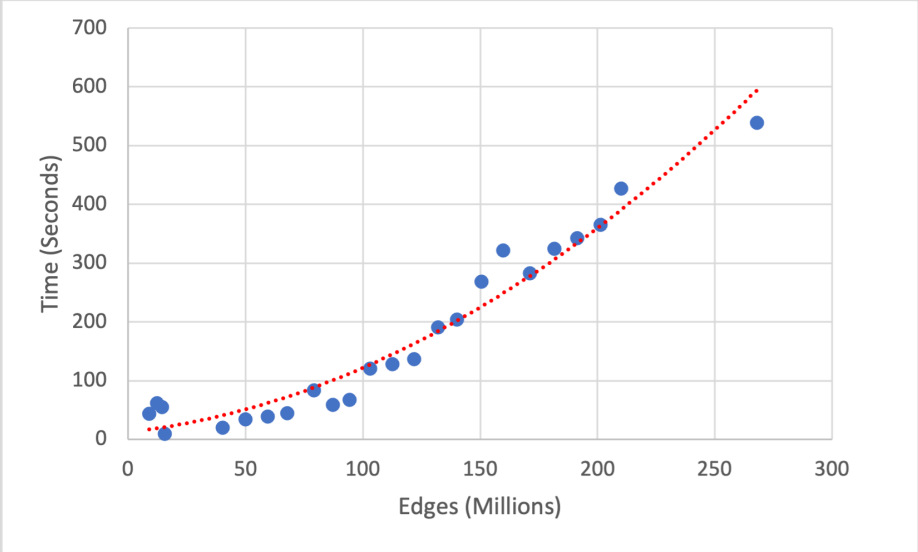


Fig. 6. Testing Our Intersection Algorithm on Networks with Billions of Triangles.

of the instances. The line of best fit is a polynomial of degree 2 and shows that our algorithm scales very well with graphs with massive amounts of triangles.

Our intersection-based implementation also produces competitive results when compared to the state-of-the art triangle counting algorithms [16]. Algorithms were analyzed and compared by fitting a model of graph counting times, T_{tri}, as a function of the number of edges $N_e = |E|$. This data was then used to estimate the parameters N_1 (the number of edges that can be processed in one second) and β:

$$T_{tri} = (N_e/N_1)^{\beta}$$

to compare different counting implementations. Implementations with a larger N_1 and smaller β perform the best, and the top entries from the 2019 review had N_1 values ranging from 5×10^5 to 5×10^8, and β values ranging from $\frac{1}{2}$ to $\frac{4}{3}$. For reference, our algorithm had $\beta = \frac{3}{4}$ and $N_1 = 1 \times 10^7$.

After examining the comparative performance of our triangle counting algorithm, we proceeded to expand the implementation to include the metrics described in Section 2.4 - triangle counting, triangle vertex degree, triangle edge degree, and k-count calculations. Similar to the basic counting experimental results, our intersection method of this "full count"was faster than miniTri1 and miniTri2 on every instance, with speedups ranging between 2× and 177× on the ten largest instances, displayed in Table 2. One noteworthy observation about these results is that due to the data structure that stored the enumerated triangles, the k-count calculation of our algorithm ran much faster than those of miniTri, even though the code implementation was nearly identical. This demon-

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

Table 2. Comparing Our Full Count Intersection Algorithm with miniTri1 and miniTri2 on Large Real World Networks.

Graph Characteristics				Time in Seconds			Speedup	
Name	$\|V\|$	$\|E\|$	$\Delta(G)$	mt1	mt2	int	mt1/int	mt2/int
Loc-gowalla	196591	950327	2273138	156	106.4	0.882	177	121
roadNet-PA	1090920	1541898	67150	2.067	1.792	0.076	27	24
roadNet-TX	1393383	1921660	82869	2.544	2.207	0.070	36	32
flickrEdges	105938	2316948	107987357	1112	N/A	553.1	2	∞
amazon0312	400727	2349869	3686467	26.8	22.29	0.932	29	24
amazon0505	410236	3356824	3951063	28.71	23.39	0.997	29	23
amazon0601	403394	3387388	3986507	28.24	25.09	0.998	28	25
roadNet-CA	1965206	5533214	120676	3.706	3.212	0.134	28	24
Cit-Patents	3774768	33037894	7515023	157.21	N/A	3.502	45	∞

strates the versatility of *FDC* and *FN* in their ability to perform a wide range of network analytics.

4 Conclusion

Network data is usually input as a list of edges which can be preprocessed into a representation such as an adjacency matrix or adjacency list, suitable for algorithmic processing. We have presented a simple, yet flexible scheme based on intersecting edge labels, the intersection matrix, for the representation of and calculation with network data. A new linear algebra-based method exploits this intersection representation for triangle computation – a kernel operation in big data analytics. By using sparse matrix-vector products instead of the memory-intensive matrix-matrix multiplication, our implementation has the capacity to enumerate and extend triangle analysis in graphs so that important information such as triangle vertex and edge degree can be gleaned in a fraction of the time of reference implementation of miniTri on large benchmark instances. The computational results from a set of large-scale synthetic and real-world network instances clearly demonstrate that our basic implementation is efficient and scales well. The two arrays `FDC` and `FN` together constitute a compact representation of the sparsity pattern of network data, requiring only $n + m$ units of storage. This is incredibly useful in the exchange of network data, with the potential to allow for the computation of many additional intersection matrix-based network analytics such as rank and triangle centrality [6]. A shared memory parallel implementation of this method using OpenMP is being developed, with very optimistic preliminary results. This algorithm can still be tuned, and cache efficiency is being studied for additional optimizations, exploring temporal and spatial locality to analyze the memory footprint and provide further improvements. A natural extension of the research presented in this paper is to use the intersection representation in *graphlet* counting methods. Similar to the k-count distribution, graphlet frequency distribution (a vector of the frequency

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

of different graphlets in a graph) provides local topological properties of graphs [17].

Acknowledgments This research was supported in part by NSERC Discovery Grant (Individual), NSERC Undergraduate Student Research Award, and the AITF Graduate Student Scholarship. A part of our computations were performed on Compute Canada HPC system (http://www.computecanada.ca), and we gratefully acknowledge their support.

References

1. Abdullah, W.M., Hossain, S., Khan, M.A.: Covering large complex networks by cliques—a sparse matrix approach. In: Kilgour, D.M., Kunze, H., Makarov, R., Melnik, R., Wang, X. (eds.) Recent Developments in Mathematical, Statistical and Computational Sciences. pp. 117–127. Springer International Publishing, Cham (2021)
2. Abdullah, W.M., Awosoga, D., Hossain, S.: Intersection representation of big data networks and triangle counting. In: 2021 IEEE International Conference on Big Data (Big Data). pp. 5836–5838 (2021). https://doi.org/10.1109/BigData52589.2021.9671349
3. Al Hasan, M., Dave, V.S.: Triangle counting in large networks: a review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(2), e1226 (2018)
4. Becchetti, L., Boldi, P., Castillo, C.: Efficient algorithms for large-scale local triangle counting. In: ACM Trans Knowl Discovery Data. pp. 1–28 (2010)
5. Burkhardt, P.: Graphing trillions of triangles. Information Visualization **16**(3), 157–166 (2017)
6. Burkhardt, P.: Triangle centrality. ArXiv **abs/2105.00110** (2021)
7. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. National security agency technical report **16**(3.1) (2008)
8. Hasan, M., Hossain, S., Khan, A.I., Mithila, N.H., Suny, A.H.: DSJM: a software toolkit for direct determination of sparse Jacobian matrices. In: International Congress on Mathematical Software. pp. 275 – 283. Springer (2016)
9. Kepner, J., Gilbert, J.: Graph algorithms in the language of linear algebra. SIAM (2011)
10. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014), accessed: 2019-10-02
11. Low, T.M., Rao, V.N., Lee, M., Popovici, D., Franchetti, F., McMillan, S.: First look: Linear algebra-based triangle counting without matrix multiplication. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6 (2017). https://doi.org/10.1109/HPEC.2017.8091046
12. Milo, R., Shen-Orr, S., Itzkovitz, S.: Network motifs: simple building blocks of complex network. Science pp. 824–827 (2002)
13. Palla, G., Dereny, I., Frakas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature pp. 814–818 (2005)
14. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015), https://networkrepository.com

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34

15. Samsi, S., Gadepally, V., Hurley, M., Jones, M., Kao, E., Mohindra, S., Monticciolo, P., Reuther, A., Smith, S., Song, W., Staheli, D., Kepner, J.: Static graph challenge: Subgraph isomorphism. http://graphchallenge.mit.edu/data-sets (2017), accessed: 2021-07-09
16. Samsi, S., Gadepally, V., Hurley, M., Jones, M., Kao, E., Mohindra, S., Monticciolo, P., Reuther, A., Smith, S., Song, W., Staheli, D., Kepner, J.: Graphchallenge.org triangle counting performance (2020)
17. Szpilrajn-Marczewski, E.: A translation of sur deux propriétés des classes d'ensembles by. Fund. Math **33**, 303–307 (1945)
18. Wolf, M.M., Berry, J.W., Stark, D.T.: A task-based linear algebra building blocks approach for scalable graph analytics. In: 2015 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2015)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_34