Triangle Centrality in Arkouda

Joseph Patchett, Zhihui Du, Fuhuan Li, David A. Bader
Department of Data Science
New Jersey Institute of Technology
Newark, NJ, USA
{jtp47,zd4,fl28,bader}@njit.edu

***Abstract*—There are a wide number of graph centrality metrics. Further, the performance of each can vary widely depending on the type of implementation. In this work we present our implementation of triangle centrality in Arkouda with several different triangle counting methods. Triangle Centrality is a robust metric that captures the centrality of a vertex through both a vertex's own connectedness and that of its neighbors. Arkouda is an open-source framework for data science at the scale of terabytes and beyond. These methods are compared against each other and another shared memory implementation.**

***Index Terms*—triangle counting, graph theory, large scale graphs**

I. Introduction

Ideally, a centrality metric will have several characteristics. The first is accuracy; the algorithm should return vertices that are central in the graph. The method should be quick; processing large graphs is time consuming and even optimized algorithms are still bound by Amdahl's law. Finally, methods should be robust; considering both highly connected vertices and vertices that are connected to highly connected vertices. There are many other metrics that are commonly used and have been extensively studied including PageRank [1], closeness centrality [2], betweenness centrality [3], harmonic centrality [4] and many others. For reference, the algorithmic complexity of closeness centrality and harmonic centrality is $O(mn)$. The first parallel implementation of betweenness centrality was done by Bader and Madduri [5].

The main computations in triangle centrality [6] are done in triangle counting. Using an adjacency matrix as a representation for a graph in main memory, this can be completed in $O(m\sqrt{m})$ time complexity. However, for real-world graphs, the practical performance can be very different even if they have the same number of edges because such graphs are often very sparse with very different topologies. Arkouda works with a double index data structure [7] but this demonstrates an ideal bound for performance comparison. Triangle centrality considers three properties for a vertex; the open set (including the respective vertex) of triangle counts for the neighbors of which a vertex is a part of, the triangle counts of its adjacency set that do not include that vertex in its set and the total count of triangles in a graph. The first captures the connectedness of a vertex, the second captures the connectedness of its neighbors, and the third bounds this into an interpretable range of $[0, 1]$.

This research is supported in part by the NSF grant CCF-2109988.

There has been significant research in triangle counting. Depending on the storage of the graph whether that be an adjacency matrix, adjacency lists or some other format, there are many variations. In distributed systems there has been research into using GPUs [8] [9] [10] to optimize triangle counting through bitmaps or optimizing graph partitioning methods. In a MapReduce implementation, Burkhardt achieves an optimal time complexity for triangle counting at $O(m\sqrt{m})$ [11]–[13] for adjacency matrices [12].

II. Algorithms

We implement several Arkouda methods to compare against. All of these methods utilize the double index data structure [14]. This data structure maintains a neighbor vector for which the indices are vertices in the graph. The elements in the vector are indices in the edge vectors which hold each respective vertex's edges, the source and the destination. For each vertex, the number of edges and its start in the edge vector are held in a separate vector. Because of this, a vertex's edges can be accessed in $O(1)$ time.

Many Real world graphs have a skewed degree distribution meaning that a small number of vertices have a very high degree and many vertices have a low degree. Our first triangle counting method takes advantage of this distribution by only searching vertices with the smallest degree. The degree of each vertex can be accessed in $O(1)$ making this efficient.

Algorithm 1 Minimum Search Triangle Counting Method

```
Require: Graph G
  for edges < u, v > ∈ G do
    let sv be the vertex with the smaller adjacency list
    let lv be the vertex with the larger adjacency list
    for vertices w in N(sv) do
      if N(w) > N(lv) then
        edgesearch(lv)
      else
        edgesearch(w)
      end if
    end for
  end for
```

Our other method avoids the multiple for loops necessary in most edge intersection methods and takes advantage of the

TABLE I
RUN TIMES FOR EACH METHOD (SEC)

Graph	GraphBLAS	Minimum Search	Path Merge
as-caida20071105	0.00013	0.111	0.085
ca-AstroPh	0.396	0.176	0.249
ca-CondMat	0.108	0.088	0.047
ca-GrQc	0.017	0.014	0.0079
email-enron	0.00012	0.171	0.409
loc-brightkite	0.0057	0.204	0.27

sorting in the preprocessing in Arkouda. The adjacency lists of two vertices are compared.

Algorithm 2 Path Merge Triangle Counting Method

Require: Graph G
  for edges $< u, v > \in G$ **do**
    LNI and RNI point to the indices in the edge list of the left and right vertex
    while $LNI \in N(u)$ and $RNI \in N(v)$ **do**
      if vertex at $RNI ==$ vertex at LNI **then**
        Increment Count
        Increment RNI, LNI
      else
        if $dst[RNI] < dst[LNI]$ **then**
          Increment RNI
        else
          Increment LNI
        end if
      end if
    end while
  end for

Comparisons are done such that no searches are required and the extraneous for loops are not required. All three aforementioned components of triangle centrality can be evaluated during the triangle counting step; the vertices and total number of triangles in the graph are immediately updated and the existence of a triangle is updated for each neighbor edge. Then, adjacent vertices without this moniker can be added to the respective closed set of triangles.

III. RESULTS

The experiments for both the GraphBLAS SuiteSparse library [15] version 5.1.5 and the Arkouda implementations were run on a single locale on an Ubuntu 20.04 operating system with an Intel i7-10700K CPU.

It is expected that the GraphBLAS outperforms Arkouda because it is a shared memory implementation and lacks the overhead in Arkouda but in certain graphs the Arkouda methods perform better. This could be because the data structures in Arkouda handle certain graph topologies better than GraphBLAS which was written entirely with linear algebra methods. Within the Arkouda methods there is not a clear fastest algorithm; sometimes the path merge method is twice as fast as the minimum search method but on other graphs the opposite is true.

IV. CONCLUSION

We integrate several triangle counting methods and compare each to a shared memory implementation in GraphBLAS. Despite being designed for a distributed memory implementation, the Arkouda methods run comparably or better than the GraphBLAS implementations. Due to the different performances of the path merge and the minimum search method, it may be fruitful to determine which method may be faster at run time because this can be twice as fast. Ensembling these methods could yield an efficient method that generalizes to many different graph types. Our research is open-source and available on GitHub at https://github.com/Bears-R-Us/arkouda-njit. We would like to acknowledge both the Arkouda and Chapel community.

REFERENCES

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[2] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[3] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[4] P. Boldi and S. Vigna, "Axioms for centrality," 2013.

[5] D. A. Bader and K. Madduri, "Parallel algorithms for evaluating centrality indices in real-world networks," in *2006 International Conference on Parallel Processing (ICPP'06)*, 2006, pp. 539–550.

[6] P. Burkhardt, "Triangle centrality," *CoRR*, vol. abs/2105.00110, 2021. [Online]. Available: https://arxiv.org/abs/2105.00110

[7] Z. Du, O. A. Rodriguez, and D. A. Bader, "Enabling exploratory large scale graph analytics through Arkouda," in *The 25th Annual IEEE High Performance Extreme Computing Conference (HPEC)*, 2021.

[8] M. Bisson and M. Fatica, "Static graph challenge on GPU," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–8.

[9] ——, "Update on static graph challenge on GPU," in *2018 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–8.

[10] L. Hoang, V. Jatala, X. Chen, U. Agarwal, R. Dathathri, G. Gill, and K. Pingali, "DistTC: High performance distributed triangle counting," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2019, pp. 1–7.

[11] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick, "Listing triangles," in *ICALP*, 2014.

[12] P. Burkhardt, "Graphing trillions of triangles," *Information Visualization*, vol. 16, no. 3, pp. 157–166, 2017.

[13] S. Chu and J. Cheng, "Triangle listing in massive networks and its applications," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 672–680. [Online]. Available: https://doi.org/10.1145/2020408.2020513

[14] M. Merrill, W. Reus, and T. Neumann, "Arkouda: interactive data exploration backed by Chapel," in *Proceedings of the ACM SIGPLAN 6th on Chapel Implementers and Users Workshop*, 2019, pp. 28–28.

[15] T. A. Davis, "Graph algorithms via SuiteSparse: GraphBLAS: Triangle counting and k-truss," in *2018 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–6.