Incremental Graph Clustering in Parallel

Md Taufique Hussain
Ariful Azad
mth@indiana.edu
azad@iu.edu
Indiana University
Bloomington, Indiana, USA

ABSTRACT

We develop a distributed memory graph clustering algorithm to find clusters in a graph where new nodes and edges are being added incrementally. At each stage of the algorithm, we maintain a summary of the clustered graph computed from all incremental batches received thus far. As we receive a new batch of nodes and edges, we cluster the new graph and merge new clusters with the previous summary clusters. We use sparse linear algebra to perform these operations. Our algorithm would make it possible to find clusters in very large graphs for which regular graph clustering algorithms could not run due to computation/communication bottlenecks.

1 PROBLEM DEFINITION

An incremental graph is a graph where new nodes and edges are being added incrementally. Every time a batch of nodes and edges are added we call that an incremental step. Let M_{all} be the adjacency matrix of the graph in an incremental step. We can divide M_{all} into four regions as shown in the Fig. 1. We name the regions as - M_{11} representing the edges between old nodes only (red in the figure), M_{12} and M_{21} representing the edges between old and new nodes (green in the figure) as well as M_{22} representing the edges between new nodes only. Our goal is to find clusters in the entire graph(M_{all}) assuming we know the clusters(shaded regions) in the old graph (M_{11}).

Simple solution is to apply an existing clustering algorithm on the entire graph(M_{all}) ignoring the clustering structure. Let $C = \{c_1, c_2, ..., c_x\}$ be the set of clusters detected in this way. With $nnz(.)$ we represent number of edges present in the graph. This solution has computational cost and memory requirement more than $O(nnz(old) + nnz(new)))$ in every incremental step - which is very high for very large graphs. In our target solution we want to remove $O(nnz(old)$ from both computational complexity and memory requirement. Let $B = \{b_1, b_2, ..., b_y\}$ be the set of clusters in this solution. We can measure success by computing the agreement between B and C with F-score.

2 MOTIVATION

Our motivation to study this problem comes from the application of metagenomic protein family detection. In this application, a protein similarity graph is built from the database of genome sequences coming from different environmental samples. Each of the sequences are represented as a node. Similarity between the sequences are measured by alignment which becomes edge weight. Clustering is performed on this graph to detect protein families.

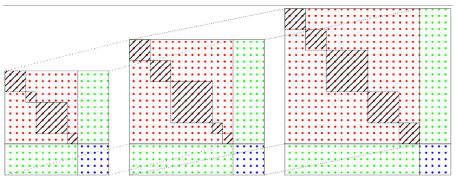

Figure 1: Incremental clustering pipeline using adjacency matrix representation. Each matrix represents an incremental step. In each incremental step, red region represents edges between old nodes, green region represents the edges between old and new nodes, blue region represents the edges between new nodes. Shaded region represents clusters.

There are several factors that make proetin family detection a challenging task. Firstly, these graphs are very big. For example, to the best of our knowledge, largest such graph on which anyone could successfully detect clusters is *Metaclust50* having 282 million nodes and 37 billion edges with average degree 131 (https://metaclust.mmseqs.com/). Around 1 terabytes is required just to store it in memory, intermediate memory requirement for computation is even higher. So, very large scale distributed computing resources are required to find clusters in such graphs. For example, clustering *Metaclust50* using *HipMCL* required using 2048 compute nodes (131,072 cores) on NERSC Cori supercomputer for 3+ hours - which was one quarter of entire Cori [1]. Secondly, Even larger graphs are available. For example, *PASTIS* have successfully generated graphs of 405 million proteins [7] [9]. Even larger graphs than that are possible. For example, JGI IMG/M Database contains currently 77 billion sequences [4]. A graph generated using the entire database would contain 7 trillion edges (considering average degree 100). No supercomputer is capable of meeting the memory requirement of problems of such scale. Even if there is any, no graph clustering algorithm will scale due to communication cost being the bottleneck. Hence, it is impossible to find clusters of such large graphs on any supercomputer without changing the approach. On top of that, thirdly, these graphs are incremental in-nature. For example, 65 billion sequences in August 2021, 70 billion sequences in October 2022, 77 billion sequences in August 2023 [4]. It is huge waste of resources to compute everything from the scratch in every incremental step. Our two big aims of this work are -

- **Reduce memory requirement:** Find clusters of the new graph using smaller resources.

Conference'17, July 2017, Washington, DC, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

- **Reduce resource wastage:** Find clusters of new graph faster than doing it again

Both without loosing much accuracy.

3 ALGORITHM

Our solution to find clusters in a single incremental step consists of three substeps.

Subset-1: We detect clusters in M22 using regular clustering algorithm. For our target application we use markov clustering(MCL) [8].

Subset-2: We combine four pieces($M_{11}, M_{12}, M_{21}, M_{22}$) to create an incremental graph(M_{inc}). Using full M_{11} and M_{22} is equivalent to full graph clustering. Simple solution is to remove inter cluster edges from M_{11} and M_{22}. But we can do better as we explain later. To combine four pieces, we use parallel sparse matrix assign(*SpAssign*) [3] operation. To reduce the cost, we assign each of the four matrices to four empty matrices and apply sparse matrix addition(*SpAdd*) [5] operation. Load balancing these parallel operations is tricky due to the block structure. To solve that we permute the order of vertices at each incremental step. This brings the complexity of keeping track of cluster assignments.

Substep-3: We merge or split the clusters detected in M_{11} and M_{22}. Simple solution is to apply regular clustering algorithm on the incremental graph M_{inc}. We choose MCL again for our target application.

3.1 Sparsification of M_{11} and M_{22}

Instead of using entire M_{11} in the substep-2 of our algorithm, we make use of some properties of MCL to maintain a sparse summary of M_{11}. MCL runs the following steps until convergence - simulate random walk using sparse matrix-matrix multiplication(SpGEMM), inflate/deflate edge weights using elementwise power, discard low weight edges by prunning. Matrix first become denser, then after first few iterations keeps getting sparser. When it gets sparser, clustering structure starts to emerge. In our solution, we keep the MCL state when number of nonzeros(nnz) drops to a certain threshold as the sparse summary of M_{11} for the next incremental step. To give an idea, in our dataset at around fifth iteration the matrix gets 30% sparse when we save the summary, but it varies from dataset to dataset. Ours is not completely a random choice as some other variants of MCL have used the similar idea [6].

We can analytically show that the incremental graph M_{inc} stays significantly sparser than the full graph M_{all} which helps cluster detection. If average degree of the entire graph is d, it is being clustered in k incremental steps, at each incremental step our sparse summary keeps fraction r of the edges that it starts with, then average degree of the incremental graph at k^{th} incremental step can be expressed as a sum of the geometric series $\frac{d}{k}+\frac{d}{k}(r+r^2+...+r^{k-1})$. When r is smaller than 1, the term is smaller than d. If there are n nodes in the graph, SpGEMM on M_{all} has complexity $O(nd^2)$ while M_{inc} has complexity $O(n(\frac{d}{k} \times \frac{1-r^k}{1-r})^2)$. Because simulating random walk with SpGEMM is the most expensive operation of MCL, detection of clusters on M_{inc} is faster than on M_{full}. Also, sparsifying this way ensures that more meaningful sparsification happens for the MCL iterations in the next incremental step. Only removing inter-cluster edges can not ensure that.

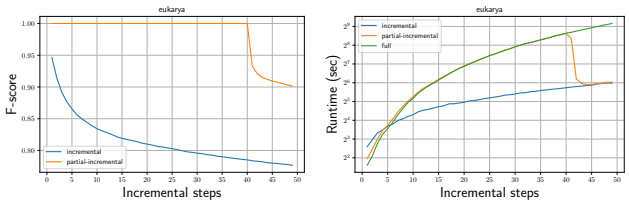

Figure 2: Preliminary experiment result on Eukarya using 50 incremental steps. For the first 40 incremental steps *partial-incremental* is equivalent to *full* hence F-score is shown as 1.0.

4 IMPLEMENTATION AND RESULTS

We implement our algorithm in C++ using the Combinatorial BLAS library [2]. To test our idea preliminarily we use a comparatively smaller metagenomic protein similarity network, named Eukarya, consisting of 3.2 million nodes and 360 million edges. We use 16 compute nodes (1024 cores) on NERSC Cori supercomputer to do that. We assign 4 processes per node and 16 threads per process. For incremental clustering we simulate 50 incremental steps by randomly sampling equal number of vertices and corresponding edges from the entire graph in each step. To evaluate, we use two variants of incremental setting. One is starting incremental clustering from the beginning, we call it *incremental*. Another is running incremental clustering for the last 20% incremental steps, we call it *partial-incremental*. In each incremental step, we also run full graph clustering using HipMCL on the entire graph received thus far, we call it *full*. To compute accuracy of output produced by incremental variants, we compute F-score against the output produced by *full*. The result is shown in Fig. 2.

On the left side of Fig. 2 we see that we are not loosing accuracy much by engineering all these steps in our algorithm. One would probably want to use incremental clustering only when the graph gets large, rather than using it from the beginning. From that perspective, accuracy of *partial-incremetal* is even more exciting as it produced more than 90% accurate result.

On the right side of Fig. 2 we see that, on the last incremental step, our algorithm is producing this accurate result while being 10× faster than detecting clusters on the entire graph. On larger graphs and higher parallelism, we expect to see this gap to be even more significant.

5 CONCLUSION

For graphs that are not so massive our algorithm is a trade off between accuracy and computational resources but for massive graphs that are incremental in-nature this is the only scalable way to detect clusters. We see promising results for metagenomic protein similarity graphs. Hence, our algorithm has potential to enable unprecedented scale of graph clustering. We believe our work will accelerate new scientific discovery in two possible ways - first by enabling analysis of new metagenomic data, second by enabling participation of scientists who do not have access to large computational resources in order to do this kind of analysis.

REFERENCES

[1] Ariful Azad, Aydin Buluç, Georgios A Pavlopoulos, Nikos C Kyrpides, and Christos A Ouzounis. 2018. HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks. *Nucleic Acids Research* 46, 6 (01 2018), e33–e33.

[2] Ariful Azad, Oguz Selvitopi, Md Taufique Hussain, John R Gilbert, and Aydın Buluç. 2021. Combinatorial BLAS 2.0: Scaling combinatorial algorithms on distributed-memory systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2021), 989–1001.

[3] Aydin Buluç and John R Gilbert. 2012. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal on Scientific Computing* 34, 4 (2012), C170–C191.

[4] I-Min A Chen, Ken Chu, Krishnaveni Palaniappan, Anna Ratner, Jinghua Huang, Marcel Huntemann, Patrick Hajek, Stephan J Ritter, Cody Webb, Dongying Wu, et al. 2023. The IMG/M data management and analysis system v. 7: content updates and new features. *Nucleic Acids Research* 51, D1 (2023), D723–D732.

[5] Md Taufique Hussain, Guttu Sai Abhishek, Aydin Buluç, and Ariful Azad. 2022. Parallel Algorithms for Adding a Collection of Sparse Matrices. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 285–294.

[6] Venu Satuluri and Srinivasan Parthasarathy. 2009. Scalable graph clustering using stochastic flows: applications to community discovery. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 737–746.

[7] Oguz Selvitopi, Saliya Ekanayake, Giulia Guidi, Muaaz G Awan, Georgios A Pavlopoulos, Ariful Azad, Nikos Kyrpides, Leonid Oliker, Katherine Yelick, and Aydin Buluç. 2022. Extreme-scale many-against-many protein similarity search. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[8] S. van Dongen. 2000. *Graph clustering by flow simulation.* Ph. D. Dissertation. Utrecht University.

[9] Linda Vu. [n. d.]. ExaBiome Brings Metagenomics into the Exascale Era. *Exa-News* ([n. d.]). https://www.exascaleproject.org/exabiome-brings-metagenomics-into-the-exascale-era/