Community Detection Using Parallel Computing

Lokesh Sreenath (ljakka2)

Nidhish Kamath (nkamath5)

Taaha Kazi (tnkazi2)

Utsav Majumdar (utsavm2)

In partial requirement for the course

IE 533 : Big Graphs & Social Networks

Prof. Rakesh Nagi

Date: 2nd May, 2023

Abstract

Community detection is the process of identifying cohesive groups, or communities within a network or graph. The goal of community detection is to identify groups of nodes that are densely connected to each other but sparsely connected to nodes in other groups. However, devising an algorithm to detect these communities using traditional/ established sequential programming algorithms demands a time-complexity of at least O(m) [1] and is often inefficient for large graphs. In this work we have proposed a parallel implementation of Memory Based Label Propagation Algorithm. We have also benchmarked both the quality and the performance of this algorithm against datasets of varying size.

Key words: Community Detection, Parallel implementation on GPU, Multi-threading, Label Propagation Algorithm (LPA), Memory LPA.

Introduction

Due to the enormity in the amount of data available today, it is often challenging for us to estimate correlations between various entities and get to know the underlying distribution and connection from large graphs. Community detection is an important problem in the context of large graphs. We want to be able to find subgraphs in the form of clusters or groups that have a dense network of edges within themselves as compared to the sparse number of edges that connect separate groups. This can be achieved by attributing nodes based on multiple characteristics that would allow a node to belong to a certain community. This extrapolates to a variety of applications for different kinds of datasets. For example, in a dataset of research paper citations, it will form communities of researchers with shared research domains which will help people find and network with other people outside of their known network but having the same interests. This could also help in word sense disambiguation in Natural Language applications, where certain words can be grouped together based on their occurrences in similar contexts. For e-commerce websites like Amazon, it will help in detecting groups of individuals purchasing similar items which would translate to better targeted recommendation and marketing strategies.

There are many algorithms for detecting communities in graphs. We want to implement one that will converge fast given larger datasets and also provide meaningful clustering, forming communities as close to the ground truth as possible. Interestingly, though each approach evaluates its own performance and time complexity, not many of them talk about or focus on parallelizing for faster convergence. Below are a few hand-picked approaches suggested for the community detection problem in the literature.

SCD: Under a parallel paradigm, Prat-Pérez et al. [5] proposed a novel disjoint community detection algorithm called Scalable Community Detection (SCD). In this work, they used Weighted Community Clustering (WCC) as a metric to optimize iteratively. They measured the change in WCC caused by shifting the communities of each node and they did so in a parallel fashion.

BigClam: Using a generative model approach of forming a cluster was discussed in [4]. Here, Non-Negative Matrix Factorization technique is used to decompose the adjacency matrix of a network into a product of two matrices, where each matrix represents the affliction of each node to each community. This model, called the BigClam, is computationally expensive and also very sensitive to hyperparameters such as the probability of each cluster strength.

Louvain Algorithm: The Louvain Algorithm is a hierarchical algorithm that aims to find the optimal partition of a node in a network into communities or modules, based on the “Modularity Metric”. It consists of Two phases: The First phase involves optimizing modularity locally for each node by iteratively moving it to the community that results in the maximum rise in Modularity. Then the second phase involves constructing a new network whose nodes represent the communities found in the first phase and edges correspond to the sum of weights of edges between the nodes in different communities. First Phase will only be relabeled if there is a rise in modularity.

One of the primary advantages of the Louvain algorithm is its scalability to very Big Network. It's flexible to form a range of communities for a wide range of communities.

Naim et al. [6] talk about the Parallel implementation of the Louvain Method of solving Community Detection Algorithms. This algorithm is the first for this problem that parallelizes the access to individual edges, allowing for load balancing when the processing networks with the node according to its degree. They have achieved the speed of 270x compared to the sequential algorithm in parallel implementation. A unique feature this paper implemented was to "parallelize the hashing of individual edges both in the modularity optimization and also in the aggregation phase".

MemLPA: Fiscarelli et al. [2] augment the classical Label Propagation Algorithm [1] by implementing a memory mechanism that "remembers" the previous labels for the nodes in the network as the algorithm evolves. All the labels "collected" by a node are scored, and the scores are updated in an iteration. Nodes choose the label with the maximum score at the end of each iteration. They show that their algorithm is able to mitigate issues seen with the classical LPA, such as over-propagation of a single label over the network. Further, in their paper Fiscarelli et.al. present a thorough evaluation of their algorithm against other established community detection methods (both LPA-based and otherwise) on real and synthetic datasets such as the LFR benchmark [3]. They show that their algorithm, called the MemLPA, is one the most performant algorithms when it comes to capturing the ground truth as indicated by classical metrics such as the Normalized Mutual Information (NMI), Adjusted Rand Index (ARI) as an indication of the quality of clustering and also topological metrics like community size, internal transitivity, scaled density and others. MemLPA also has a time complexity comparable to the classical LPA, which is near linear in network size (O(m)).
Though not explicitly mentioned as a merit in the paper, we note that this algorithm is highly parallelizable owing to independent calculations for each node in the network, and a tendency of the algorithm to work better with a synchronous update rule - where the labels for each node at the end of each iteration are updated at the same time.

For reasons mentioned above, we choose to adopt the memory-based label propagation algorithm, MemLPA [2], for our implementation going ahead.

We have divided the sections ahead into six parts: Section 1 talks about the formulation of the community detection problem, Section 2 details how the MemLP algorithm approaches the community detection algorithm, Section 3 delves into our implementation of MemLPA and Section 4 provides results of the same. Section 5 gives a generic benefit analysis of performing community detection. Section 6 concludes this study and provides direction for extending this work in the future.

Section 1 : Problem Formulation/ Objectives

We aim to provide a solution to the community detection problem using the Memory based Label Propagation Algorithm **[2]** and the power of parallel programming (CUDA). Generated communities are tested against the ground truth using the NMI (Normalized Mutual Information) score, which is a normalization of the Mutual Information score; which in turn quantifies the the amount of information obtained about one clustering (usually ground truth) from the information observed from another clustering (proposed clustering). Mathematically these quantities are defined as follows:

$$\boxed{NMI(C,T) = \frac{I(C,T)}{\sqrt{H(C)\cdot H(T)}}}$$

$$\boxed{I(C,T) = \sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij}\log(\frac{p_{ij}}{p_{C_i}\cdot p_{T_j}})} \quad \boxed{H(\mathbf{c}) = -\sum_{i=1}^{n} p_i \log_2 p_i}$$

Here, I(C,T) is the Mutual Information score; H(C) and H(T) are quantities called the entropy (Shannon's entropy) for the types of clustering C (proposed) and T (ground truth). P_{ij} is the joint probability of the various clusters in C (indexed by i) and T (indexed by j). It is calculated by counting the number of nodes that belong both to clusters C_i & T_j and dividing by the total number of nodes. Pc_i and Pt_j are marginal probabilities of the clusters C_i & T_j, measured as a ratio of their individual sizes divided by the total number of nodes in the graph. 'n' denotes the number of nodes in the graph, 'r' denotes the total number of communities in the proposed clustering, and 'k' denotes the total number of ground truth communities.

Our objective is to maximize the NMI score, i.e propose communities as closer to the ground truth as possible in minimal time reducing from O(m) complexity. Upon building the pipeline for execution of the algorithm, we plan to run the following datasets through it:

1) Zachary's Karate Club

2) LFR bench-marking dataset generated with varying mixing parameters.

Section 2 : Approach - MemLPA

Sequential Algorithm:

The sequential implementation of the algorithm consists of the following steps:

1. Initialization:
    a. Initialize an adjacency matrix of a weighted graph G=(N,E).
    b. Independent values are assigned as labels to each node in the graph.
    c. Memory is allocated for each node to maintain a history of label changes and neighborhood edge weight summation tuples for every iteration.
    d. An Active List is initialized for each node as false, which is used to keep track of nodes that have already been finalized and integrated into a community.

   e. The edge weight can be augmented to include a neighborhood fraction score (NFS). Neighborhood fraction score is explained in Section 3.

2. Iteration:

   The following steps are iterated as long as there are nodes remaining in the Active List:

   a. Check each node in the graph for its neighbors and add each neighbor and its edge weight to the memory.
   b. Select the neighbor with the highest edge weight and update the label of the current node with the label of the selected neighbor.
   c. If the label of the selected neighbor already has a non-zero value, add the new edge weight to the label weight.

3. Termination:

   Currently, the termination criterion for the loop is based on a heuristic where we check if the label of a node changes over n iterations. If it doesn't change, the node is removed from the Active List. We have chosen a value of n=3 for the label history implemented on a smaller graph, which gives satisfactory results in terms of community detection. However, the value of n can be increased to change the criteria for label change over n iterations, if needed.

Parallel Algorithm:

From a mathematical perspective, a parallel algorithm works identically to the sequential one. But from an implementation perspective, there are many changes in the type of data structures used and the operations carried out to maximize parallelism.

1. Initialization:
   a. An adjacency matrix of a weighted graph G=(N,E) We define it a Compressed Sparse Row (CSR) matrix.
   b. Independent values are assigned as labels to each node in the graph.
   c. A CSR matrix called MemoryMat is initialized to keep a history of label changes and neighborhood edge weight summation.

   d. The edge weight can be augmented to include a neighborhood fraction score (NFS).

2. Iteration
   a. We map the neighbors to their current labels using a parallelised gather operation.
   b. We calculate the current score of each label for each node by building a CSR matrix.
   c. This current score is added to the history score
   d. We then calculate the argument maximum of each row of the MemoryMat. This value is now the label of the corresponding node.
   e. We add a very small value in the MemoryMat at each label corresponding to each node.
3. Termination criteria is the same as that of the sequential algorithm.

Mathematical Formulation for MemLPA:

$Let\ N(u)\ \&\ N(v)\ denote\ the\ set\ of\ neighbors\ of\ node\ u\ \&\ v\ respectively.$

$Let\ N.F.(u, v)\ denote\ the\ entries\ in\ row\ u\ \&\ column\ v\ of\ the\ matrix\ N.F.,$

$and\ N.F.(v, u)\ denote\ the\ entries\ in\ row\ v\ \&\ column\ u\ of\ the\ matrix\ N.F.$

$Then,\ the\ entries\ of\ the\ Neighborhood\ Fraction\ matrix\ N.F.,\ are\ defined\ as\ -$

$$N.F.(u, v) = (N(u) \cap N(v)) \div N(v)$$

$$\&\ N.F.(v, u) = (N(u) \cap N(v)) \div N(u)$$

$The\ adjusted\ adjacency\ matrix\ A*\ is\ given\ by\ -$

$$A* = N.F. \odot A$$

$where\ \odot\ denotes\ element\ wise\ matrix\ multiplication$

$and\ A\ is\ the\ adjacency\ matrix\ describing\ the\ graph.$

$$CurrentScore\ (v, i) = \sum_{u \in N(v)} [l_u == i]$$

$We\ interpret\ this\ as:\ The\ Current\ score\ of\ the\ v^{th}\ node\ for\ label\ i\ is\ equal\ to\ the\ summation\ of\ the\ distance\ scores\ of\ all\ its\ neighbors\ where\ their\ label\ is\ equal\ to\ i.$

$CurrentLabel_v = argmax\ MemoryMatrix_v$

$Here,\ v\ stands\ for\ a\ specific\ node\ in\ the\ graph.$

$MemoryMatrix_{i+1} = MemoryMatrix_i + CurrentScore_{i+1}\ ;\ i: \{0, 1, ..., k\}$

$where\ i\ is\ the\ iteration\ number\ \&\ k\ is\ the\ total\ number\ of\ iterations\ till\ convergence$

Section 3 : MemLPA - Implementation Details

At present, we have thoroughly investigated the memory label propagation algorithm, its implementation and developed both sequential and parallel codebases for C++ and Python.

Codebases developed :

1. Sequential :
    a. Python → MemLPA, Louvain, NMI
    b. C++ → MemLPA

2. Parallel :
    a. PyCUDA → MemLPA
    b. CUDA → Neighborhood fraction calculation

We have implemented our algorithm on multiple datasets. However, we focused our algorithms primarily on the LFR benchmark (Lancichinetti et al. [3]), an established benchmark in the literature for community detection, that allows us to generate networks with properties similar to real-world networks.

The LFR model is a popular generative model for synthetic networks with communities which makes it useful for our purposes for the following reasons. Firstly, LFR graphs have a modular structure where the nodes are organized into groups or communities. This means that they have well-defined communities with clear boundaries, which makes it easier to evaluate the performance of community detection algorithms. Secondly, the LFR model allows for the generation of graphs with different levels of community structure, which makes it possible to test algorithms under different scenarios. Thirdly, LFR graphs are generated with a degree distribution that closely resembles that of real-world networks. This means that our algorithms can be tested on graphs that have similar properties to real-world networks, which is important for evaluating their performance in practical applications. Finally, LFR graphs are generated with a tunable mixing parameter that controls the degree of overlap between communities. This allows for the generation of graphs with varying levels of community overlap, which is useful for testing algorithms that are designed to handle overlapping communities. Since our algorithm is designed to cater to detecting disjoint communities with non-overlapping nodes, we preferred using the LFR generated graphs.

The parameter of LFR generated graph that can be controlled are :

1. Nodes : Total number of nodes in the graph
2. $\tau 1$: The exponent of the power law degree distribution for nodes in the network. This parameter controls the degree distribution of the network and affects the number of nodes with high degree. Higher values of $\tau 1$ lead to fewer high-degree nodes.
3. $\tau 2$: The exponent of the power law distribution for the community size. This parameter controls the distribution of community sizes and affects the number of small and large communities. Higher values of $\tau 2$ lead to more small communities and fewer large communities.
4. κ : The average degree of nodes within their own community. This parameter controls the internal connectivity of communities and affects the density of edges within communities. Higher values of κ lead to denser communities.
5. μ : The fraction of edges that connect nodes in different communities. This parameter controls the external connectivity of communities and affects the degree of inter-community connectivity. Higher values of μ lead to more inter-community edges and more overlap between communities.

For our purposes, we have maintained the following parameters :

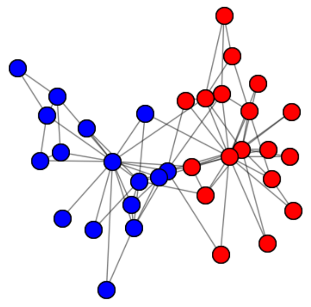
$\tau 1 = 3$, $\tau 2 = 1.5$, $\kappa = 5$

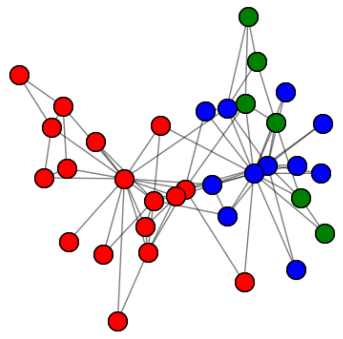
We varied the number of nodes from 1000 to 1000000 and checked the values of μ ranging from 0.1 to 0.4. We also set the minimum number of communities at 40.

Datasets implemented on :

1. Toy graph of 10 nodes → Sequential C++, Python, PyCUDA (MemLPA, Louvain)
2. Zachary's Karate Club → Sequential C++, Python, PyCUDA (MemLPA, Louvain)
3. LFR generated graphs :
    a. 1000 nodes, 0.1 μ index → Sequential C++, Python, PyCUDA (MemLPA, Louvain)
    b. 1000 nodes, 0.2 μ index → Sequential C++, Python, PyCUDA (MemLPA, Louvain)
    c. 1000 nodes, 0.3 μ index → Sequential C++, Python, PyCUDA (MemLPA, Louvain)
    d. 1000 nodes, 0.4 μ index → Sequential C++, Python, PyCUDA (MemLPA, Louvain)
    e. 10000 nodes, 0.4 μ index → Sequential C++ (MemLPA)
    f. 100000 nodes, 0.1 μ index → Sequential Python, PyCUDA (MemLPA)
    g. 100000 nodes, 0.3 μ index → PyCUDA (MemLPA)
    h. 100000 nodes, 0.4 μ index → PyCUDA (MemLPA)
    i. 1000000 nodes, 0.4 μ index → PyCUDA (MemLPA)

Table 1 below, shows the preliminary results we obtained on the Zachary Karate Club dataset.

Method	NMI	Graph
Ground Truth	1.0	

MemLPA	0.74	

Table[1]: NMI using MemLPA with visualizations on the Karate Club Dataset

Section 4 : Results and Discussion

We tested the performance of the algorithm for the following parameters:

1. **Quality of Results**: We measured how effective the algorithm was in predicting the ground truth on graphs generated using the LFR benchmark. We did so by measuring the NMI. We also benchmarked the score against a varying number of nodes and a varying number of μ values. The μ parameter dictates the mixing parameter of the nodes in the graph. We also benchmarked the performance of our algorithms against an off the-shelf Louvain implementation [7]. Note that this implementation doesn't make use of parallel computations.

Louvain Algorithm (on 1K nodes)	NMI score ∈ [0,1]		MemLPA (on 1K nodes)	NMI score ∈ [0,1]
μ = 0.1	0.971		μ = 0.1	0.5530
μ = 0.3	0.246		μ = 0.3	0.4513
μ = 0.4	0.187		μ = 0.4	0.4383

Table [2] : Comparison of NMI scores for the Louvain algorithm and MemLPA

We also noticed that the number of communities generated by the MemoryLPA was higher in number. On further inspection, we noticed that this higher number of communities was because of a large number of ‘singleton’ communities, i.e. a lot of communities were generated that contained only one node. We address this issue in Section 6.

2. **Performance**: We measured the speed of implementation of the algorithm. For this, we benchmarked the performance of the algorithm against its sequential implementation and calculated the boost in performance. As seen in the tables below, we see that for a large graph (~100K nodes), the parallel implementation gives a 78x - 100x speedup over a sequential implementation.

MemLPA (Sequential) (on 1K nodes)	**Time (in seconds) averaged over 3 runs**		**MemLPA (Parallel) (on 1K nodes)**	**Time (in seconds) averaged over 3 runs**
$\mu = 0.1$	5.953		$\mu = 0.1$	5.174
$\mu = 0.3$	6.456		$\mu = 0.3$	5.143
$\mu = 0.4$	6.983		$\mu = 0.4$	5.188

Table[3] : Comparison of the time taken: MemLPA-Sequential against MemLPA-Parallel

MemLPA (Sequential) (on 100K nodes)	**Time (in seconds) averaged over 3 runs**		**MemLPA (Parallel) (on 100K nodes)**	**Time (in seconds) averaged over 3 runs**
$\mu = 0.1$	522.291		$\mu = 0.1$	6.692
$\mu = 0.3$	680.941		$\mu = 0.3$	6.746
$\mu = 0.4$	650.463		$\mu = 0.4$	6.607

Table[4] : Comparison of the (time taken): MemLPA-Sequential against MemLPA-Parallel

Section 5 : Discussion / Benefit Analysis

We can use community detection on a variety of applications that involve connected datasets in the form of graphs. Community detection helps us understand the underlying structure of complex networks, such as social networks, biological networks, and transportation networks. By identifying communities, we can gain insights into how nodes are organized and connected within a network, revealing patterns of interaction, hierarchy, and organization. It can also be used to identify anomalies or outliers in a network. Some common applications are as follows :

1. Social Network Analysis: Community detection can be used to identify groups of people who are connected to each other within their social network, once the communities are formed.
2. Recommendation systems: Community detection can be used to identify groups of users who are likely to be interested in the same products using a nested network, where nested nodes can act as products.
3. Web Search: Community detection can be used to identify groups of web pages that are related to each other.

Section 6 : Conclusion and Further Work

The memory label propagation algorithm is embarrassingly parallelizable as it involves performing the same operations on all individual nodes in the graph for each iteration. But in spite of its simplicity, its implementation involves transforming the fundamental data structures used and optimizing them for a parallel implementation. We have successfully implemented a parallel implementation for memory label propagation and have also benchmarked the quality of the results, and the performance of the algorithm.

Our current implementation can further be optimized to increase performance as well as quality. For improving performance we plan to undertake the following steps:

1. Build out the solution entirely in CUDA i.e. not use CuPy as our interface in CUDA. We believe building a native solution in C++ and CUDA will deliver further speed ups.
2. We plan to work on faster pruning of nodes that will not contribute to further iterations, in order to improve scalability and freeing up memory. We believe this should speed up per iteration performance.
3. Efficient Termination: Currently, our algorithm terminates when the last three iterations have the same label values. This condition, although simple, tends to lead to certain edge cases where the label values end up oscillating and prevent the algorithm from terminating. We plan to build a termination check that is robust to this condition. For improving the quality of the algorithm, we intend to mitigate the 'singleton' community problem noticed. One possible solution to this problem could be assigning singleton nodes to the communities that are closest to it. We believe this will substantially improve the NMI.

In conclusion, we anticipate that our efforts in improvements in quality and performance will lead to improved efficiency and scalability of our algorithm, making it more suitable for real-world applications with large-scale datasets. This will contribute to the advancement of memory label propagation techniques and have potential implications in various domains such as social network analysis, recommendation systems, and community detection.

References

[1] Raghavan, UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106. http://dx.doi.org/10.1103/PhysRevE.76.036106

[2] Fiscarelli, A.M., Brust, M.R., Danoy, G. et al. Local memory boosts label propagation for community detection. Appl Netw Sci 4, 95 (2019). https://doi.org/10.1007/s41109-019-0210-8

[3] Lancichinetti, A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110. https://link.aps.org/doi/10.1103/PhysRevE.78.046110

[**4**] J. Yang and J. Leskovec, “Overlapping community detection at scale: a nonnegative matrix factorization approach,” in Proceedings of the sixth ACM international conference on Web search and data mining. ACM, 2013, pp. 587–596 https://doi.org/10.1145/2433396.2433471

[**5**] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. 2014. High quality, scalable and parallel community detection for large real graphs. In Proceedings of the 23rd international conference on World wide web (WWW '14). Association for Computing Machinery, New York, NY, USA, 225–236. https://doi.org/10.1145/2566486.2568010

[**6**] Md. Naim, Fredrik Manne, Mahantesh Halappanavar, and Antonino Tumeo. Community Detection on the GPU. https://bora.uib.no/bora-xmlui/bitstream/handle/1956/16753/Paperill.pdf

[**7**] CDlib - Community Discovery Library

https://cdlib.readthedocs.io/en/latest/reference/cd_algorithms/algs/cdlib.algorithms.louvain.html

Appendix

Our codebase can be accessed here:

https://github.com/nkamath5/MemLPA-Community-Detection

Links to relevant datasets:

Zachary Karate Club: http://konect.cc/networks/ucidata-zachary/

LFR:https://networkx.org/documentation/stable/reference/generated/networkx.generators.community.LFR_benchmark_graph.html