Using Concurrency For A Community Detection Algorithm

Raghid Abdeljawad and Sina Mohammadi

Department Of Information Technology Uppsala University

UPPSALA UNIVERSITET

Objectives

Since networks contains a lot of dimensions as well as edges (connections between nodes), the performance of algorithms that act on these networks is pretty bad, specially the computational time. A community detection algorithm "Generalized Louvain" is one of these algorithms that perform badly with respect to dimensions and number of edges. The objective of this project will be to use concurrency in order to increase the performance gain of the community detection algorithm (Generalized Louvain).

Introduction

The data today is far greater than the data available in previous years, due to the growth of social media and the internet. Hence it has become more interesting to use the data by analyzing and mining it to extract useful information. Some of these data sets can be visualized as a **Multilayer Network**. These networks usually contains "communities", where a group of nodes are more connected to each other than the rest of the network. Detecting these communities is an important task in social network analysis, allowing us to identify and understand the communities within the social structures. There are several algorithm that are used in order to do this, and one of these algorithms is the "Generalized Louvain Algorithm". Since this algorithm is very time consuming, the objective has been to increase performance using concurrency.

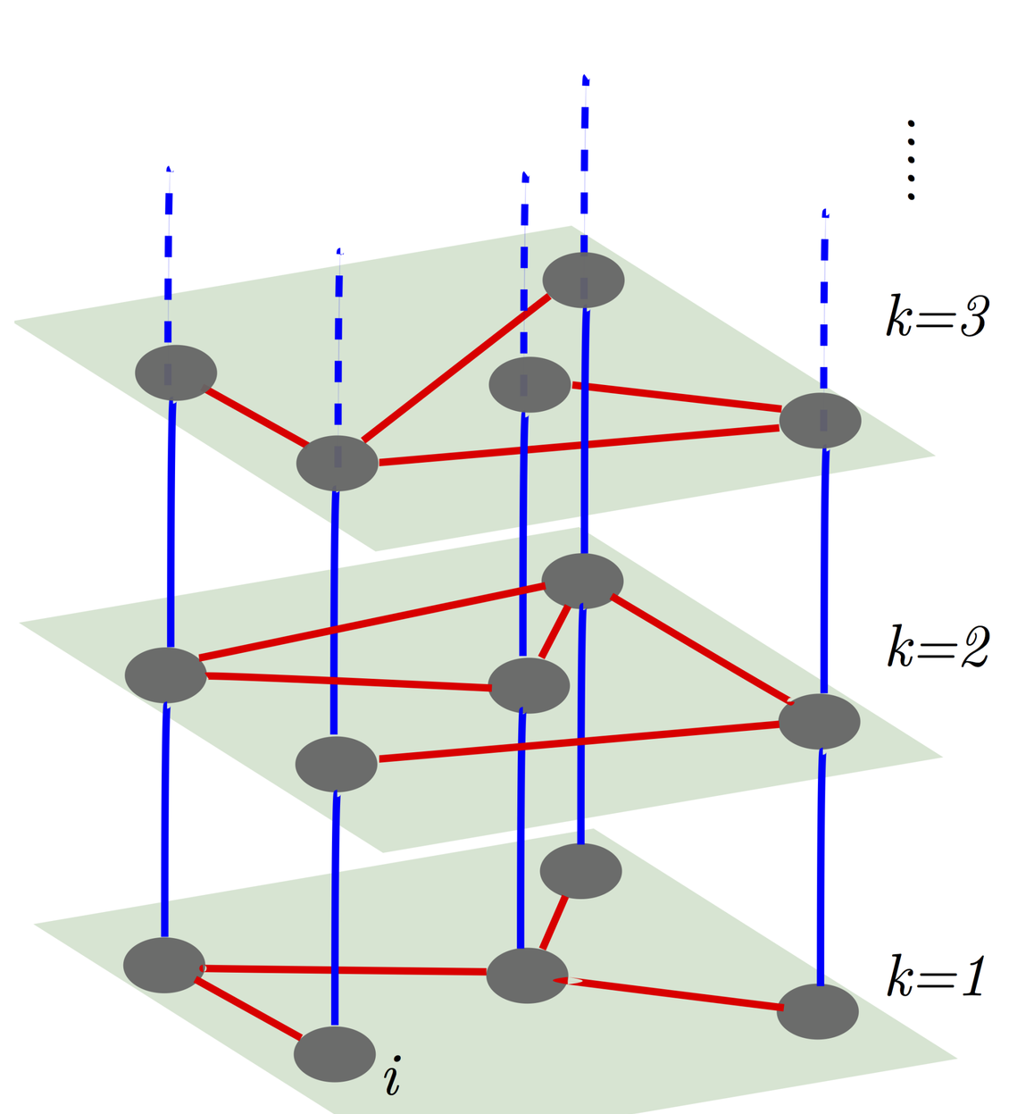

Figure 1:3D Multilayer Network [3]

Main work

The three main parts in this project are the **Network generation**, **Algorithm Testing** and **Optimizing with concurrency** (see Methods). The networks/benchmarks has been generated using MATLAB, a prepared framework provided has been been used to generate data sets [1]. The algorithm itself is written in C++ programming language. Finally the concurrency is implemented in C++. OpenMP is the interface that has been used to parallelize the algorithm.

- **Network Generation → MATLAB**
- **Algorithm Testing → C++**
- **Concurrency → OpenMP**

Methods

Benchmarks are needed to test the Generalized Louvain algorithm. These Benchmarks has been generated with a framework in MATLAB. As mentioned before, the networks could be of different sizes and different densities, hence one need to specify the parameters that characterise the network. Many benchmarks with different characteristics has been generated in order to find the bottlenecks of the algorithm. The next step was to test the algorithm on these benchmarks. By doing these tests, one could figure out the performance of the algorithm and therefore try to parallelize the part of the code that are most time consuming.

Important Result

- The computational time is most sensitive to the number of edges in the network (Density of the network).
- Computing the modularity is the most time consuming part of the code.

Results

As mentioned before, the algorithm is most sensitive to the number of edges the network has. When the networks have been generated, two of the parameters controls the "Powerlaw distribution", which in turn controls the number of edges. These parameters has been changed in order to get benchmarks with different edge densities.

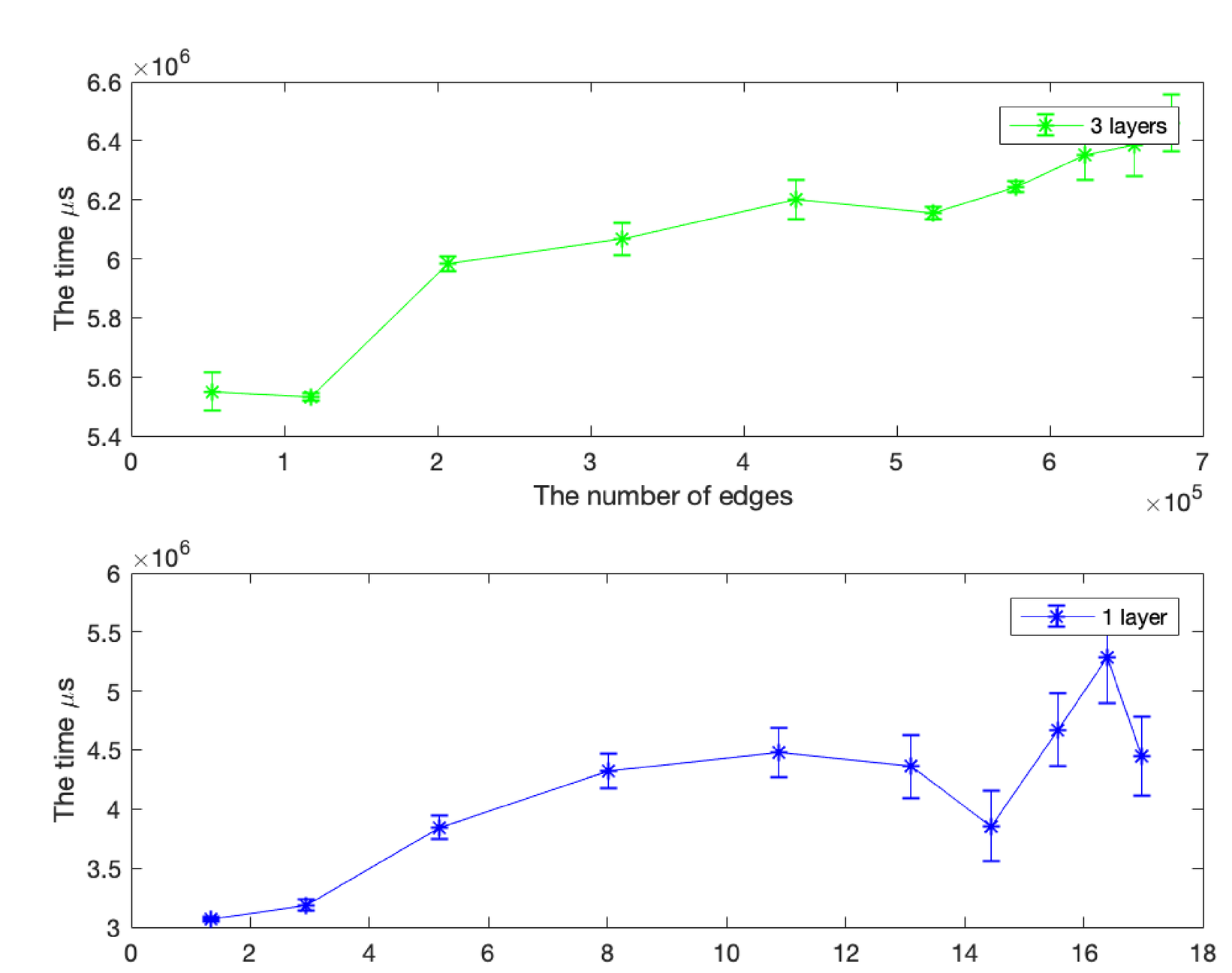

Figure 2:The performance of Glouavin

Results

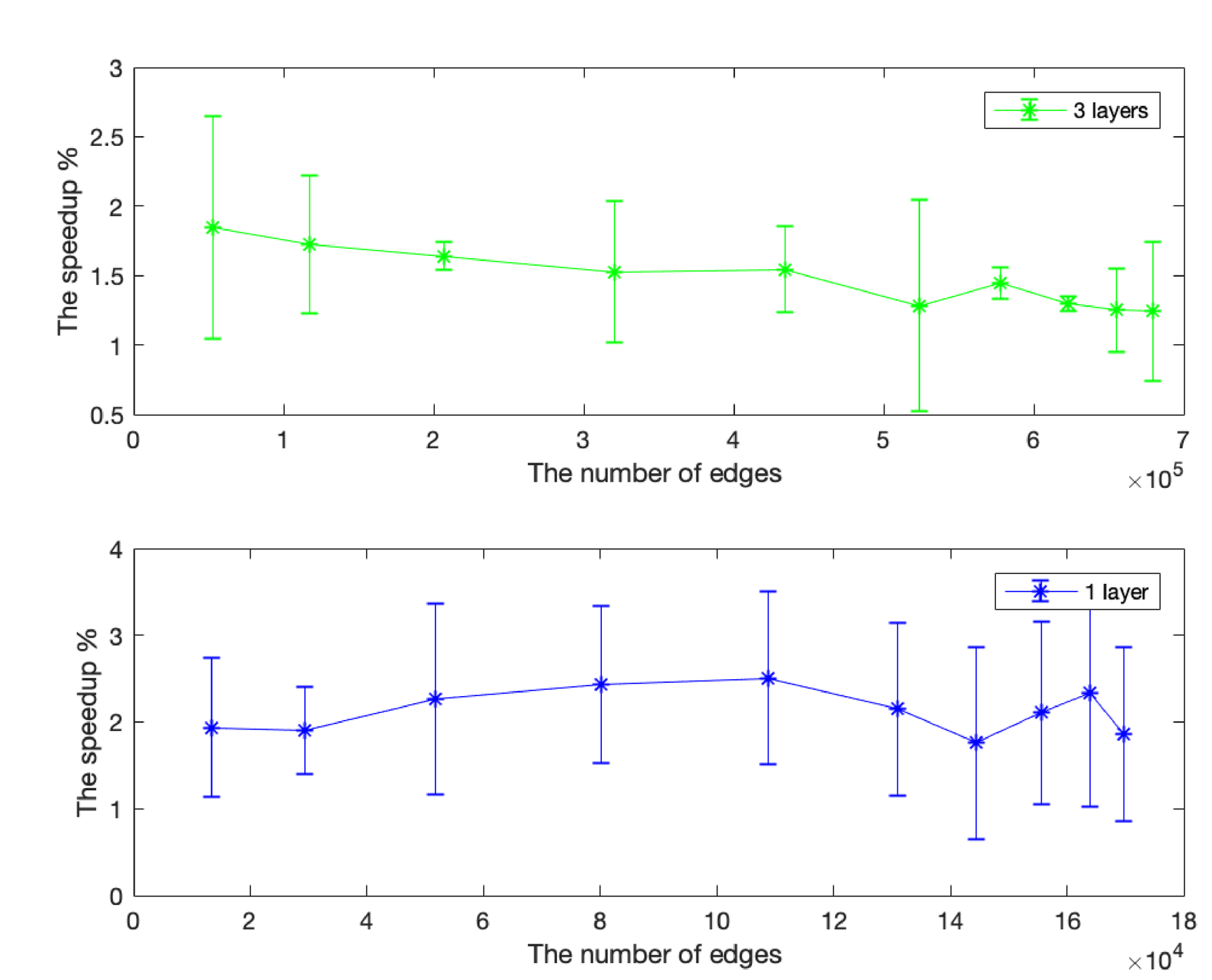

Figure 3:The speedup using openmp

Using openmp gives a pretty good speedup accourding to figure 3. From the standard division we see that the speedup vary a little bit and this is caused by the varying amount of work of the parts that has been parallelized.

Conclusion

After the algorithm had been parallelized we got a small amount of performance increment. Since the testing and characterization of the bottlenecks has taken more time than anticipated, there are some areas to consider going forward with this algorithm. When working with data mining the amount of data is usually quite large. Since the algorithm consumes a lot of memory in order to analyze the small data sets that we have been testing, one should look into how to reduce memory complexity of the algorithm.

Additional Information

- The MATLAB code for the generation of the networks, see reference [1]
- The Generalized Louvain algorithm is a part of the CRAN package Multinet, but written in C++.

References

[1] Lucas G. S. Jeub Marya Bazzi.
A framework for the construction of generative models for mesoscale structure in multilayer networks.
2019.

[2] Kevin Macon Mason A. Porter Peter J. Mucha1, Thomas Richardson and Jukka-Pekka Onnela.
Community Structure in Time-Dependent, Multiscale, and Multiplex Networks.
2010.

[3] Nikos E. Kouvaris.
http://nikos.techprolet.com/.
2013.

Acknowledgements

Contact Information

- Email: Raghid.Namir@gmail.com
sina.sa.mohammadi@gmail.com