Parallelization of the Louvain Algorithm Using a Distributed Framework

Finn Archinuk*, Ezra MacDonald*, Alison Ziesel*

Abstract

A wide variety of data can be represented as network graphs, and the analysis of these graphs for underlying structure is an important step in characterization of the underlying data. Massive datasets typically require excessive computational power or distributed analysis. In this project, we implemented a well-regarded community detection algorithm, the Louvain algorithm, using the PySpark distributed computing framework. We were able to successfully apply our implementations to both a modestly sized dataset and an extremely large dataset. We found that distributed computing has distinct design requirements relative to non-distributed approaches. We also found that for extremely large datasets, community detection is an important first step, not the only step, to data characterization.

Introduction

Network graphs can be used to describe a wide variety of disparate phenomena, including social contacts, cellular metabolism networks, structures of predator-prey relationships, communities of related and interlinked web pages, and many more (Zachary 1977; Newman 2006; Martinez-Antonio, Janga, and Thieffry 2008; Dunne, Williams, and Martinez 2002). This representation of a complex organization allows for quantification of certain features of the network, including degree, weight, and directionality of relationships, and these quantitative characterizations are especially well-suited for a computationally-based analysis.

Not all features of a network graph are strictly quantitative, however. The concept of 'community' within a network, while often easy to recognize visually, can be difficult to describe objectively. A commonly accepted definition of community within a larger network is that nodes within a community are more densely linked with one another than they are with nodes outside of said community (Figure 1) (Newman 2006). In 2004, Newman and Girvan published a pivotal paper that among other contributions, provided the numerical notion of modularity by which to identify community status within a network graph (Newman and Girvan 2004). While their initial formulation was more complicated, later authors simplified their proposed modularity as a quantification of actual in-group graph edges less the number of expected edges for a graph of the same size and with the same characteristics (Brandes 2008).

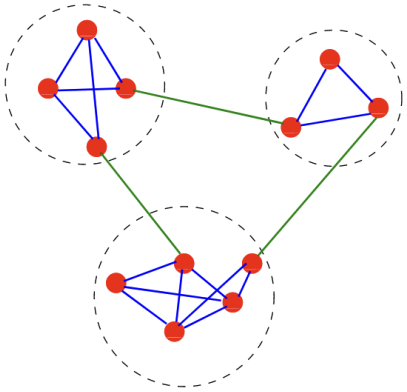

Figure 1: An example of a graph with three separate communities, each exhibiting dense internal edges and sparse inter-community edges.

Communities are of special interest in graph analysis, as they may represent local subfeatures within the larger network; highly connected nodes within a community may represent a locus of control or an influential individual, and so identifying communities and their membership can help reveal underlying characteristics of the interaction network. In the early 2000s, several methods for identifying communities were proposed, with two hierarchical approaches including divisive algorithms (the stepwise removal of edges to identify subgraphs or modules) and its conceptual inverse, agglomerative algorithms (the restoration of edges sequentially to a set of nodes) (Newman and Girvan 2004; Clauset, Newman, and Moore 2004).

However, these approaches were typically very computationally costly, with time complexities as high as $O(n^3)$, limiting their utility to relatively small networks. Following the introduction of modularity as a feature of communities within networks, algorithms that sought to maximize modularity were developed; these however are doomed to be only approximations, as it was demonstrated in 2006 that modularity maximization is an NP-complete problem (Brandes

*These authors contributed equally.

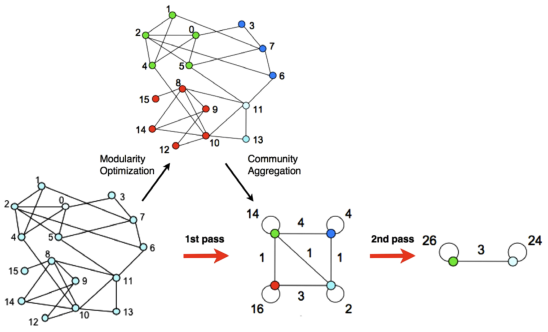

Figure 2: From Blondel *et al.*, a graphical representation of the two phases of the Louvain algorithm. Additional compaction of communities is possible with additional application of the algorithm.

et al. 2006; Brandes 2008).

Louvain Algorithm

In 2008, Blondel and colleagues published their algorithm, called the Louvain algorithm for their academic location, which they showed was capable of identifying community structure in even very large sparse networks with a comparatively low time complexity, described by a later paper as "essentially linear in the number of links [edges] of the graph" (Blondel et al. 2008; Lancichinetti and Fortunato 2009). The Louvain algorithm relies on a number of heuristics and as such is an inexact algorithm, but it was remarkable at the time for its relative speed and capability compared to contemporary methods.

The Louvain algorithm is an iterative approach with two distinct phases. Initially, all nodes of a graph are assigned to their own unique clusters. In the first phase or moving phase, for each node, the impact on modularity if that node were moved to the cluster containing one of its neighbour nodes is assessed, with the node under consideration being placed in the cluster where modularity is maximized. In the second or contraction phase, the nodes contained within a given cluster are reduced to a single node in a network of clusters, with edges between nodes in that cluster converted to a single self linked edge with weight equal to the sum of the intra-node edges, and outbound edges between clusters being set to the weight of the sum of those inter-cluster edges in the resultant network. These steps are performed iteratively, creating a further condensed network of clusters at each step, until modularity no longer increases, or a set number of passes has been performed. Figure 2 depicts one pass of the two phases of the Louvain algorithm.

The Louvain algorithm was embraced by the network analysis community for its speed, capabilities with even very large networks, its unsupervised nature, and intuitive results. Notably, each pass of the Louvain algorithm produces a 'meta-community' of increasing density, which can reveal a hierarchical association between clusters within the greater network, a feature that was relatively novel compared to contemporary algorithms.

Originally written in C++, the algorithm has since been implemented in other popular graph libraries like Networkx in Python and has been made available in graph databases like Neo4j (Hagberg, Swart, and S Chult 2008; Needham and Hodler 2019). Attempts to further augment the large network capabilities of this algorithm have led to research groups parallelizing and distributing the algorithm, with a 2015 conference paper describing its parallelization and a later 2018 paper detailing its conversion to a distributed algorithm using Thrill, a C++-based data processing framework (Que et al. 2015; Hamann et al. 2018). Additional work has been performed since these two papers were published, including efforts to distribute the algorithm using Spark (Makris, Pettas, and Pispirigos 2019; Shirazi et al. 2019; Liu et al. 2021).

A recent paper describes the Louvain algorithm distributed using PySpark, although there is no publicly available code repository cited (Makris and Pispirigos 2021). As there is no currently available public repository of code implementing the Louvain algorithm in PySpark, this represents an open challenge. Implementing it in PySpark introduces a distributed version of this algorithm into the Python data science ecosystem, rather than the relatively less rich Java data science community. Further, such an undertaking is especially well suited to the learning objectives laid out in this class, and for these reasons, our project team has proposed to implement a distributed version of the Louvain algorithm in the PySpark big data processing framework.

Implementation

Our initial project proposal was to implement the Louvain algorithm as detailed in the paper written by Hamann *et al.* using PySpark (Hamann et al. 2018). On implementing the algorithm, we were able to explore multiple techniques and tools for implementing graph algorithms in Python, and each team member gained considerable experience while developing an implementation of the Louvain algorithm. The accompanying code repository that details these efforts can be found at https://github.com/aziesel/CSC502_Project.

Datasets

Les Misérables Character Network

During the development, debugging and validation phases, we chose to use the Les Miserables character co-appearance network created by Donald Knuth (Knuth 1993). The graph is connected and consists of a vertex set of characters, while the undirected edge set consists of character co-appearances where the weights count the respective number of co-appearances. The network with one version of clustering applied appears in figure 3. The synchronous Louvain algorithm implemented in the `python-louvain` library took less than a second to analyze this network using a standard, non-subscription tier Google Colaboratory notebook allotted 12Gb of RAM, 107.7Gb of disk storage and no additional hardware acceleration (ie., no GPU).

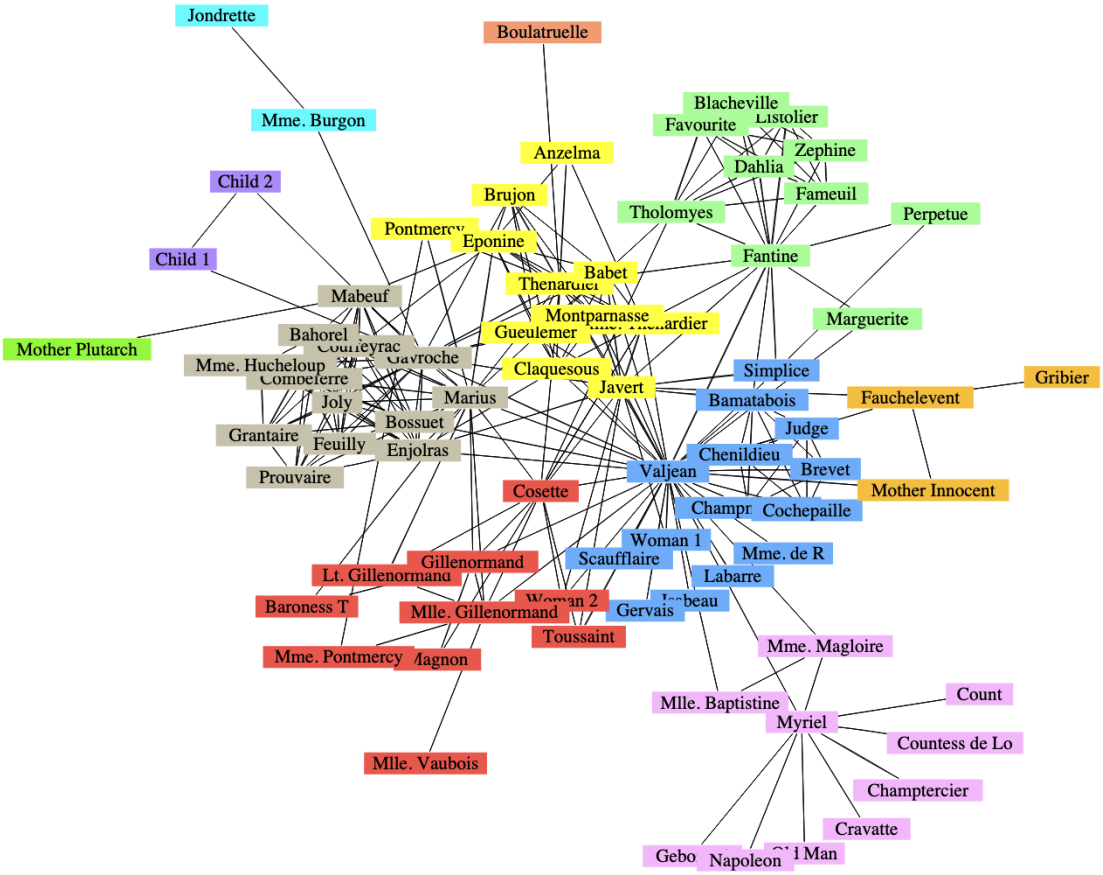

Figure 3: From Knuth 1993. The *Les Misérables* character interaction network, with one clustering produced by a non-Louvain clustering algorithm.

ArXiv Co-Authorship Network

We also analyzed the metadata describing a large collection of 1.7 million ArXiv preprint manuscripts, spanning submissions from 2007 through 2023 (Tricot et al. 2023). A Jupyter notebook (`toy_network_and_data_preprocessing.ipynb`) was prepared to parse out author information per preprint manuscript, and a dictionary of dictionaries structure was generated to store information regarding primary authors, their co-authors, and the number of manuscripts that author-co-author pair wrote together. Extracting this reduced dataset resulted in 1,712,999 authors and 8,478,247 connections. Edges were exported as a JSON file for further analysis. This notebook was also used to perform some preliminary exploratory analysis of the dataset, and to test the existing `python-louvain` library in a non-distributed fashion on the dataset. Using a Google Colaboratory notebook, partitioning the authorship network into communities took 49 minutes.

In addition to use of the free Google Colaboratory service, we also made use of our personal computers for some analysis. Some analyses conducted by Alison Ziesel were performed on a Mac Studio M1 Max with 64 Gb of RAM, and by Ezra MacDonald on a PC with 64 GB of RAM and an AMD Ryzen 7 3700X 8-Core Processor. Finn Archinuk used a 2012 MacBookPro with 8GB RAM with a core i5 intel processor.

To simplify this dataset for visualization and limit less informative connections, authors were filtered for exclusion. Authors with too few (less than 10) or too many (over 50) co-authorships were removed. The choice to remove highly connected authors was intended to increase the importance of small local clusters, and it was hypothesized that highly connected nodes would complicate local clustering. It was discovered that the most highly connected author was “The LIGO Collaboration”, which is a very large astronomical organization, with over 5000 connections. Institutions and organizations were not directly filtered out, though we believe this may be a reasonable reduction since human co-authors would remain connected. Removing the less connected authors reduced the number of nodes to be interrogated, and it was believed they would have limited impact on clustering.

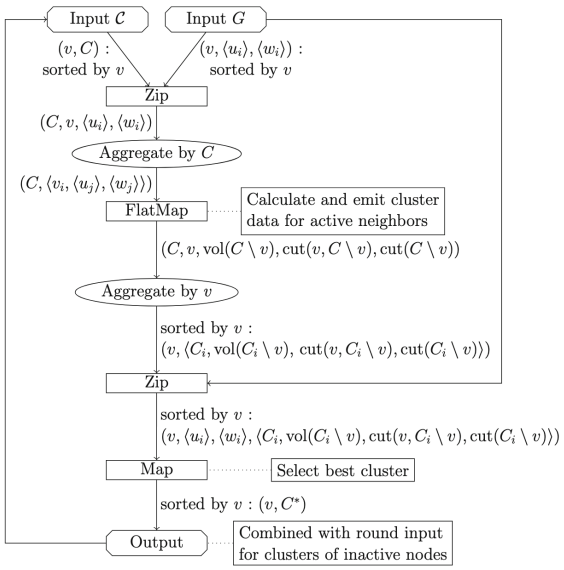

Figure 4: From Hamann *et al.*, the directed acyclic graph describing their implementation. This was used as a guide for developing our implementations.

Results

Development

Our team undertook parallel investigations, exploring suitable data structures and organization for distribution of the Louvain algorithm. Two implementations of the primary cluster generation phase of the algorithm, and one capable of the secondary compaction phase were developed, each with a slightly different approach to the problem. These implementations can be referred to as `PySpark Louvain` and `Threaded Louvain`.

All attempts at implementation aimed to follow the directed acyclic graph describing information flow from Hamman *et al.* (Figure 4). While our implementations were informed by the Hamann paper, Hamann's modularity equation was unclear, specifically how they made use of the volume and cut functions of clusters to calculate changes in modularity. Both of our developed implementations instead performed an alternative modularity score using the data aggregated by cluster (Blondel et al. 2008).

PySpark Louvain

This PySpark implementation made use of dataframes organized in the same manner as those laid out in Hamman *et al.*, specifically the 'C' and 'G' tables corresponding to cluster membership and graph organization respectively. Rather than using custom object types to store information, all pertinent information is stored within these two tables, referred to as `input_C` and `input_G` in the implementation. Following the workflow outlined in Figure 4, a dataframe aggregating cluster membership, along with neighbour node information and edge weights, was also generated, called `aggregated` and used in all of the analysis steps subsequently.

Dataframe transformations are used to develop and query the `aggregated` dataframe, but many functions are still sequential in nature. Cluster membership for the entire graph was assessed three times, observing the membership table `input_C` for changes, and ceasing iterations when no further reassignments occurred. While this was conducted manually, automation of this step is trivial. Using the final cluster assignments, a graph representation of the *Les Misérables* character interaction network was produced and compared to the version produced by the non-distributed Louvain implementation. This work is detailed in the notebook `apply_louvain_toy.ipynb`.

Another noteworthy exploration was done using the GraphFrames library which provides a dataframe-based graph solution for Apache Spark. This framework was not pursued to the point of implementing the entire algorithm as joining information about nodes and their edges could not be done in a reasonable or computationally timely fashion. This implementation attempted to implement the Louvain algorithm using a PySpark RDD according to the specification provided in the paper by Hamann *et al.* and is included in the Jupyter notebook `mod_map_louvain.ipynb`.

Threaded Louvain

The `Threaded Louvain` implementation used Python's concurrent library for parallelism. Data was structured using simple built-in data structures (lists and dictionaries) and simple aggregative classes for nodes and clusters to be able to quickly reference neighbours and weights. Following Hamann, current cluster information was distributed to machines (here, threads), emitting bids for the nodes in or adjacent to the cluster. These bids were collected, sorted by node, then the node would join the cluster that resulted in the highest change to modularity. Nodes were only updated in their corresponding subround, and these subrounds were determined using a hash function based on their identifier (character name or author name).

Results

Les Misérables Dataset

We compared the results of our `PySpark Louvain` and `Threaded Louvain` implementations against the clustering results generated by a non-distributed implementation, the `python-louvain` library (Aynaud 2009). We quantitatively measured the concurrence rate between each of our implementations and that non-distributed version; our `Threaded Louvain` implementation achieved a 70.1% concurrence rate, finding 52 of the 77 characters clustered in the same fashion as the non-distributed implementation. Our `PySpark Louvain` achieved a 77.9% concurrence rate, finding 60 of the 77 characters clustered in the same fashion. Comparing our two implementations, we found 72.7% concurrence, with 56 of 77 characters clustered in the same manner. Figure 5 shows the cluster assignments for the non-distributed and our two implementations.

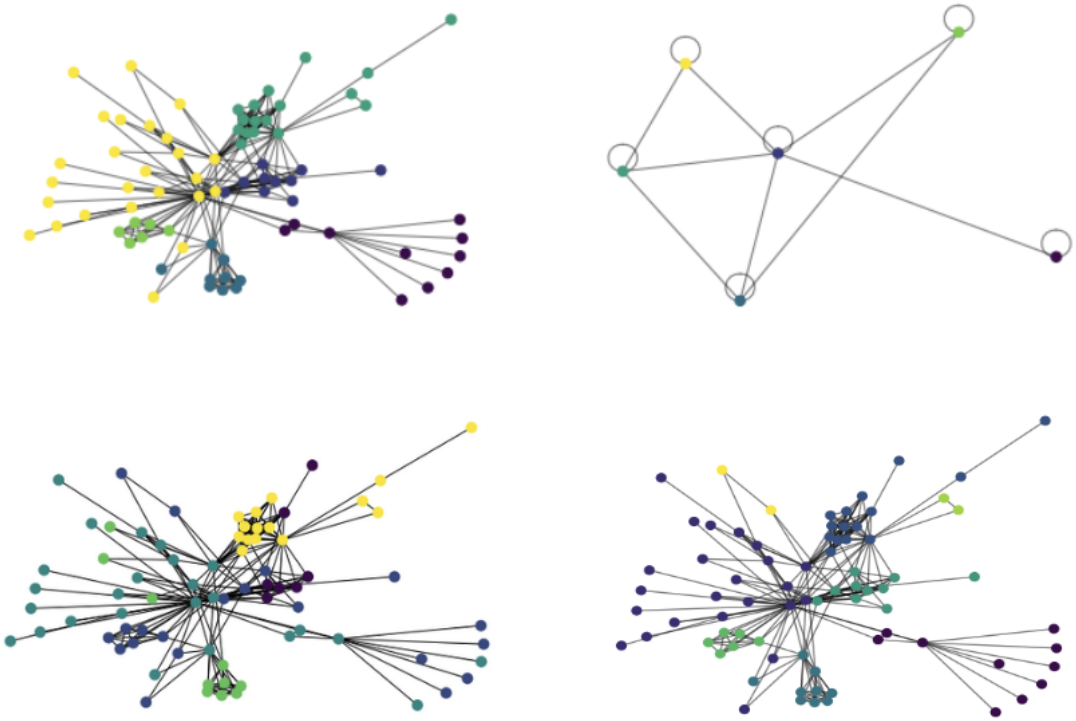

Figure 5: Comparisons of clustering produced by `python-louvain` (top left), `Threaded Louvain` (lower left) and `PySpark Louvain` (lower right). The compacted communities network produced by `python-louvain` is present in the upper right.

ArXiv Dataset

The `Threaded Louvain` implementation was used to analyze our ArXiv co-authorship network. Application of the `PySpark Louvain` implementation was attempted, but a number of warnings coupled with a cripplingly slow run time led to the abandonment of that approach. `Threaded Louvain` reduced the number of clusters from singletons (173,091 authors) to 15,573 clusters over the course of 4 iterations, with 10 subrounds each. While the Louvain algorithm performs compaction of nodes when the local moving is stabilized, our experiments with the `Threaded Louvain` on the *Les Misérables* dataset suggested that it would converge but not stabilize. It was decided to perform the compaction step at this point, where the 15,573 clusters of authors became 15,573 nodes of clusters, with self loops indicating the amount of internal connection. A single iteration of distributed local moving was performed on this compacted graph, bringing the number of clusters down to 7,278 clusters (10 subrounds). At this point we stopped our algorithm to attempt to visualize and quantify our efforts. The time elapsed was $\approx$ 85 minutes; 1 hour for the 4 local-moving iterations, 15 minutes for compaction, and 10 minutes for a local moving iteration on the compacted graph.

We identified a number of features in the co-authorship network, including 15,573 clusters within the larger network. Due to the massive size and interconnectedness of the ArXiv network, it was prohibitively difficult to conduct further quantitative analyses on our results. However, we did undertake a number of qualitative assessments regarding the clustering assignment. Figure 6 tracks the number of clusters from singleton initialization through 4 iterations (40 subrounds). Tracking the number of clusters is a very inexpensive way to ensure the algorithm is identifying dense neighbourhoods.

The `Threaded Louvain` implementation was also capable of conducting the second, compaction phase of the Louvain algorithm, something that was not achieved with the `Pyspark Louvain` implementation. Figure 7 shows a representative cluster and its neighbouring clusters post-compaction. Due to the influence of highly connected authors, compaction causes a very large number of "neighbouring" compacted clusters. This visualization demonstrates the influence of highly connected authors and begins to outline the limits of this algorithm to this particular dataset.

We produced additional visualizations, including circular graphs of a random subset of 5000 edges, as well as with a subset of the first 5000 edges (Figure 8). Since the number

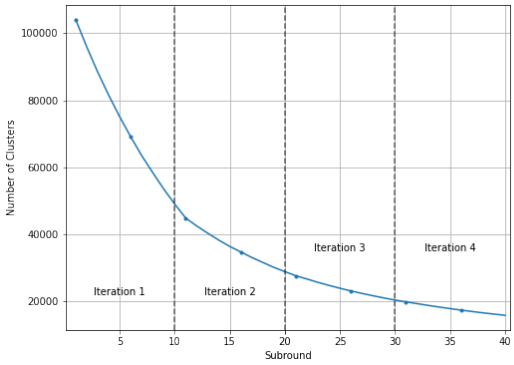

Figure 6: Progression of community identification produced by `Threaded Louvain` showing a convergence towards to fewer than 20000 clusters over 4 local moving iterations.

of edges is fixed at 1,122,714 after compaction, 5000 is a small subset. By visualizing the first 5000, we can see some structure as an artifact of how the author connections were compiled and this structure remains after a round of compaction. Seeing structure in a subset of edges indicates that the compaction step maintains community information from the previous round, though the means of visualization are challenging.

Discussion

Implementation of network clustering algorithms is a natural fit for distributed computing. In this current era of big data, many datasets naturally conform to a network modality and exploration for communities or modules within those massive networks can reveal a great deal of pertinent information. There are existing, successful implementations of the Louvain algorithm already publicly available, several of which were produced during the course of graduate studies (Sürücü 2020; Chilakamarri 2020; Trevisan and Vassio 2022). It appears that while the Louvain algorithm is relatively intuitive, implementing it in a distributed fashion is more complex than anticipated. Nevertheless, a distributed implementation of this algorithm is an important advance as it enables the analysis of ever larger networks of data in a timely manner.

However, implementation in PySpark was not completely straightforward. Throughout this project we made a number of key observations, including the importance of devising data structures appropriate to distributed analysis, organizing data in such a way to minimize repeated data access, and whether or not the use of custom object types was more appropriate than using primitive data types. By its nature, the Louvain algorithm repeatedly queries neighbour nodes for their cluster membership, and these repeated queries can pose a steep computational cost that may be alleviated by careful dataframe or RDD design. Simply caching the membership table is not sufficient, as it is repeatedly updated as the algorithm progresses and so is not a static object. Additionally, implementation of functions with an eye for distribution is also critical, as non-distributed function design does not necessarily translate well to parallelization.

Other notable observations arose during the course of our project work. In some cases, 'oscillation' in which a given node alternated between two different cluster memberships in subsequent cycles of the first phase of the Louvain algorithm was observed. In some cases this was attributable to a deviation from the formal algorithm (in particular, not updating cluster membership immediately after identifying the cluster for which maximum modularity increase was observed for the PySpark implementation), but in other instances it appeared that these oscillations may be related to the order in which nodes were assessed for membership (observed in the Threaded implementation). Related to this, the original Blondel paper does acknowledge that the algorithm is especially susceptible to the order in which nodes are queried. The Hamann paper further partitions nodes into subgroups that are active for a given subround of the first phase of the Louvain algorithm, both for parallelization reasons and possibly to reduce the number of nodes with which a given node may exhibit sub-optimal cluster assignment behaviour.

Choice of toy dataset is also a critical decision to be made when testing an algorithm implementation. While interesting, the *Les Misérables* character interaction network does not have a ground truth, and exhibits a complex organization of its seventy seven nodes. A less culturally relevant but far more practical choice for toy dataset would be a smaller, entirely artificially constructed set of nodes with obvious communities exhibiting high degrees of internal connections, with sparse and minimally weighted connections between those communities. This would have enabled us to pursue a more rapid and confident prototyping phase of implementation. It also would have made the use of free but limited resources such as Google Colaboratory more appropriate for initial phases of analysis; the free tier offered by that service has limited capabilities for distributed computing, although it's an attractive option for collaborative work among project team members.

To quantify performance of our implementations, we compared cluster assignments with assignments produced by the `python-louvain` library. We found reasonable concurrence, with 70.1 and 77.9% concurrence of assignment. The Louvain algorithm is known to be highly sensitive to the order in which nodes are assessed for cluster membership. It is very possible that most of the discrepancies in cluster assignment are due to the order of nodes queried; no attempt was made to control for this possibility. Future work should include a control of node order, as well as replicate runs of each implementation with shuffled node order to measure the contribution of node order to final cluster assignment.

We also successfully undertook analysis of our dataset of interest, the ArXiv co-authorship interaction network we produced from publication metadata, using our threaded implementation. This dataset was incredibly noisy, with superconnected, frequently publishing authors having an outsized effect on the network organization and cluster assign-

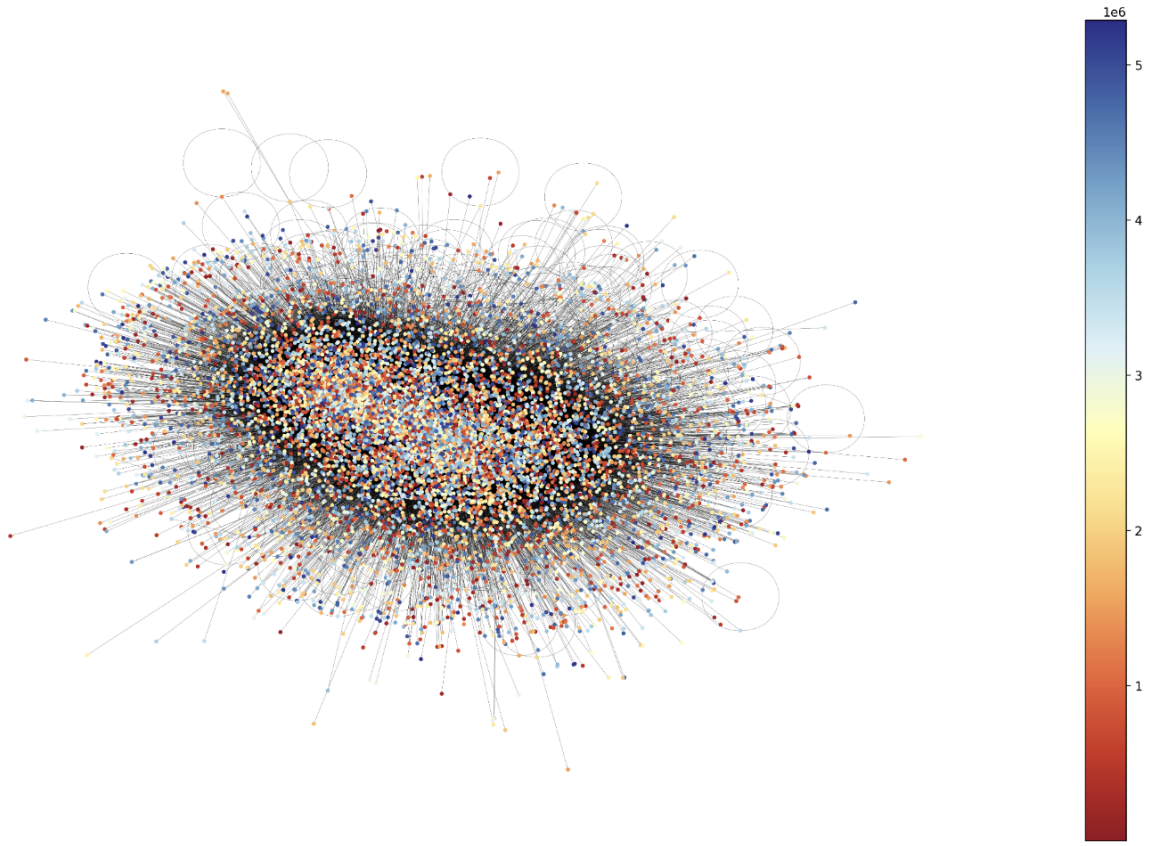

Figure 7: A single cluster and immediate neighbouring clusters from the ArXiv dataset after compaction with `Threaded Louvain`. Due to highly connected authors and fuzzy communities in interdisciplinary academia, the graph structure of this neighbourhood is impossible to parse. The nodes in this region have almost 200 000 edges to their neighbours and are connecting 263 other unique clusters. The challenge of visualizing clusters increases with additional layers of compaction.

ment. Quantification of this network was very difficult, so we instead undertook qualitative assessments of the network's structure. Distinctive patterns, described as 'fireworks', were observed, where the previously mentioned superconnected authors accumulated clusters of less connected authors around them, with connections between those clusters. In future work, authors with only one connection could be more intelligently removed by adding a self loop to their corresponding co-author, which should have a similar effect for clustering while greatly reducing extraneous edges/nodes. We additionally observed that compaction, the second phase of the Louvain algorithm in which clusters are reduced to single nodes with self-links reflecting internal cluster edges and network edges indicating inter-cluster connections, did not help in visualization of the co-authorship network. Additional rounds of compaction may possibly produce a more readily appreciable organization of co-authors, but at the expense of sharply decreased resolution of individual scientists' contributions.

Conclusions

For this project, we undertook to produce a distributed implementation of the Louvain algorithm, an effective method for identifying communities within a larger interaction network. We explored three parallel avenues to pursue this goal, and produced two implementations capable of carrying out the first phase of the Louvain algorithm, and one capable of performing the second compaction phase of that algorithm. We found reasonable concurrence of cluster assignment between our two implementations and a non-distributed implementation on our toy dataset, and found an extremely complex network and cluster organization with our experimental dataset, the ArXiv co-authorship network. During our work, we learned a great deal about implementation within distributed frameworks, including key differences in optimal data structure and method design relative to non-distributed implementation. Given that many different types and sources of massive data can be expressed as network graphs, distributed implementation of the Louvain algorithm represents an important contribution to ongoing analysis of increasingly larger datasets. The work of this project has made each of us better able to pursue future studies of mas-

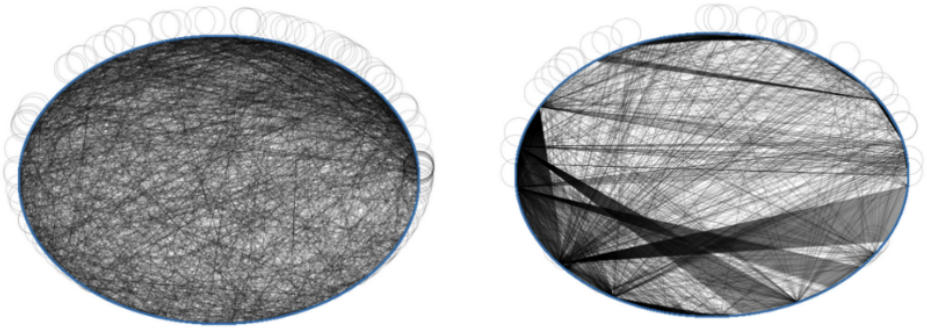

Figure 8: After compaction of the ArXiv dataset there are 1.1 million edges. A random subset of 5000 edges (left) and the first 5000 (right) are depicted on circular graphs. Only when we visualize a limited selection of the edges can local structures become discernable.

sive datasets and their analysis via distributed computing.

References

Aynaud, T. 2009. Python-Louvain GitHub repository. Https://github.com/taynaud/python-louvain.

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10): P10008.

Brandes, U. 2008. On variants of shortest-path betweenness centrality and their generic computation. *Social networks*, 30(2): 136–145.

Brandes, U.; Delling, D.; Gaertler, M.; Görke, R.; Hoefer, M.; Nikoloski, Z.; and Wagner, D. 2006. Maximizing modularity is hard. *arXiv preprint physics/0608255*.

Chilakamarri, S. 2020. *ONLINE COMMUNITY DETECTION USING TWITTER DATA*. Ph.D. thesis, San Francisco State University.

Clauset, A.; Newman, M. E.; and Moore, C. 2004. Finding community structure in very large networks. *Physical review E*, 70(6): 066111.

Dunne, J. A.; Williams, R. J.; and Martinez, N. D. 2002. Food-web structure and network theory: the role of connectance and size. *Proceedings of the National Academy of Sciences*, 99(20): 12917–12922.

Hagberg, A.; Swart, P.; and S Chult, D. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Hamann, M.; Strasser, B.; Wagner, D.; and Zeitz, T. 2018. Distributed graph clustering using modularity and map equation. In *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings 24*, 688–702. Springer.

Knuth, D. E. 1993. *The Stanford GraphBase: a platform for combinatorial computing*, volume 1. AcM Press New York.

Lancichinetti, A.; and Fortunato, S. 2009. Community detection algorithms: a comparative analysis. *Physical review E*, 80(5): 056117.

Liu, D.; Huang, K.; Zhang, C.; Wu, D.; and Wu, S. 2021. Study on Discovery Method of Cooperative Research Team Based on Improved Louvain Algorithm. *Scientific Programming*, 2021: 1–13.

Makris, C.; Pettas, D.; and Pispirigos, G. 2019. Distributed community prediction for social graphs based on Louvain algorithm. In *Artificial Intelligence Applications and Innovations: 15th IFIP WG 12.5 International Conference, AIAI 2019, Hersonissos, Crete, Greece, May 24–26, 2019, Proceedings 15*, 500–511. Springer.

Makris, C.; and Pispirigos, G. 2021. Stacked community prediction: a distributed stacking-based community extraction methodology for large scale social networks. *Big Data and Cognitive Computing*, 5(1): 14.

Martinez-Antonio, A.; Janga, S. C.; and Thieffry, D. 2008. Functional organisation of the Escherichia coli transcriptional regulatory network. *Journal of Molecular Biology*, 381.1: 238–247.

Needham, M.; and Hodler, A. E. 2019. Graph Algorithms in Neo4j: Louvain Modularity.

Newman, M. E. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103.23: 8577–8582.

Newman, M. E.; and Girvan, M. 2004. Finding and evaluating community structure in networks. *Physical review E*, 69(2): 026113.

Que, X.; Checconi, F.; Petrini, F.; and Gunnels, J. A. 2015. Scalable community detection with the louvain algorithm. In *2015 IEEE International Parallel and Distributed Processing Symposium*, 28–37. IEEE.

Shirazi, S.; Baziyad, H.; Ahmadi, N.; and Albadvi, A. 2019. A new application of louvain algorithm for identifying disease fields using big data techniques. *Journal of Biostatistics and Epidemiology*, 5(3): 183–193.

Sürücü, S. 2020. *Measuring political polarization using big data: The case of Turkish elections*. Master's thesis.

Trevisan, M.; and Vassio, L. 2022. Identification and Clustering of Anomalies in Online Social Networks.

Tricot, J.; Rishi, D.; Maltzan, B.; and Brinn, S. 2023. arXiv Dataset. Https://www.kaggle.com/dsv/5290013.

Zachary, W. W. 1977. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33.4: 452–473.