The Journal of Open Source Software

CM++ - A Meta-method for Well-Connected Community Detection

Vikram Ramavarapu[1], **Fábio Jose Ayres**[2], **Minhyuk Park**[1], **Vidya Kamath Pailodi**[1], **João Alfredo Cardoso Lamy**[2], **Tandy Warnow**[1], and **George Chacko**[1¶]

1 Department of Computer Science, University of Illinois Urbana-Champaign, IL 61801, USA **2** Insper Institute, Sao Paulo, Brazil ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Introduction

Community detection methods help uncover the meso-scale structure of networks and have broad applications (Dey et al., 2022; Haggerty et al., 2013; Karatas & Sahin, 2018; Waltman & van Eck, 2012). While communities can be defined in different ways (Coscia et al., 2011), a common expectation is one of greater edge-density within and lesser edge density between communities (Fortunato & Newman, 2022). A related expectation is that communities also should be well-connected (Bonchi et al., 2021; Traag et al., 2019).

In Park et al. (2023), we describe Connectivity Modifier (CM), a meta-method that enforces well-connectedness in communities. As input, CM takes a network, a clustering of the network generated by an algorithm, and a user-specified connectivity threshold. For each community (cluster), CM uses VieCut (Henzinger et al., 2018) to find a small edge cut, and if the edge cut size is below the specified connectivity threshold, then the cut is removed, and the partitions are reclustered using the clustering algorithm. This process repeats until all clusters are well connected.

We now present CM++ (Ramavarapu et al., 2023), which uses parallelism for scalability and has new features. CM++ provides support for additional clustering paradigms and is designed to be extensible by other developers. Concurrently, we present the CM++ Pipeline, a modular and extensible workflow that automates CM++ operations. The pipeline consists of clustering, pre-processing, connectivity modifier (CM++), and post-processing stages with generation of cluster statistics.

Statement of Need

We have demonstrated that several widely-used codes do not generate well-connected clusters (Park et al., 2023). A tool to enforce user-specified levels of well-connectedness to clusterings from multiple community detection methods is not presently available.

The Journal of Open Source Software

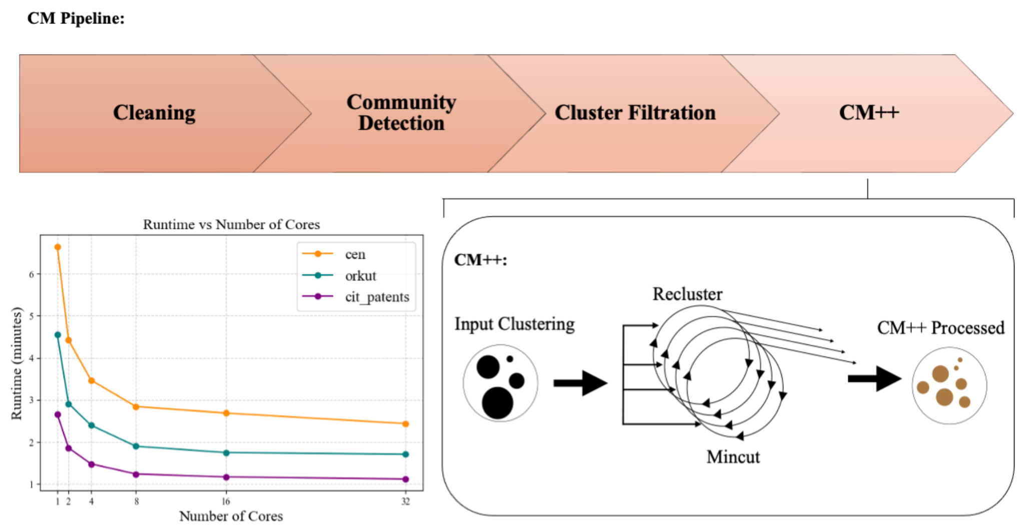

Figure 1: (Top) A visualization of a workflow created from the CM++ Pipeline. (Bottom right) Algorithmic schematic of CM++. CM++ splits the queue of clusters evenly between the spawned processes. Each process runs an instance of CM++ on its share of clusters (recursive mincutting and reclustering until well connected). (Bottom left) Runtime curve with respect to the number of parallel cores running CM++. CEN (14 million nodes, 1.3 billion edges) is the Curated Exosome Network. orkut (3.1 million nodes, 117 million edges) and cit_patents (3.7 million nodes, 16 million edges) are both from the SNAP database (Leskovec & Krevl, 2014) and processed through the removal of parallel edges and self-loops (Park et al., 2023).

CM++: Enforcing Well-Connectedness

Key Features:

- **Flexibility** For users to accompany their definition of a good community with well-connectedness, CM++ is designed to work with any clustering algorithm and presently provides built-in support for the Leiden algorithm (optimizing either the Constant Potts Model or modularity) (Traag et al., 2019), Iterative K-core Clustering (IKC) (Wedell et al., 2022), and Infomap (Rosvall & Bergstrom, 2008).
- **Dynamic Thresholding:** In order to allow the enforcement of connectivity to be flexible, connectivity thresholds can be constants, or functions of the number of nodes in the cluster, or the minimum node degree of the cluster.
- **Multi-processing:** For better performance, users can specify a larger number of cores to process clusters concurrently.

CM++ Pipeline: A Flexible and User-Friendly Community Detection Pipeline

Key Features:

- **Graph Cleaning:** Removal of parallel and duplicate edges as well as self loops.
- **Community Detection:** Clusters an input network with one of Leiden, IKC, and InfoMap.
- **Cluster Filtration:** A pre-processing stage that allows users to filter out clusters that are trees or have size below a given threshold.
- **Community Statistics Reporting:** Generates node and edge count, modularity score, Constant Potts Model score, conductance, and edge-connectivity at multiple stages.
- **Extensibility:** Developers can remove stages and design new ones.

Limitations

The current version of CM++ offers a limited range of built-in clustering options, but is designed to simplify extension by other developers. With IKC, CM++ has failed to complete on clusters on the order of a million nodes due to very high memory usage. CM++ is limited to community detection algorithms that yield disjoint communities, so algorithms that yield overlapping communities are not supported by CM++.

Parallel Strategy

In its current form, CM++ distributes the input clusters roughly equally across available cores. Each core runs an instance of CM++, and outputs are aggregated at the end. This strategy may suffer from load balancing issues if there are large outliers in cluster size. A version is being developed that uses a shared memory queue of clusters that each core can fetch from.

Conclusions

CM++ offers performant improvements over its predecessor CM. The accompanying pipeline provides additional functionality and is customizable, allowing users to re-order modules and add custom modules.

Acknowledgements

This work was supported in part by the Insper-Illinois Collaboration and by Oracle Research Awards to Tandy Warnow and George Chacko. The authors thank Nathan Bryans and Christine Ballard from Oracle Research for their assistance with the Oracle Cloud Infrastructure.

References

Bonchi, F., García-Soriano, D., Miyauchi, A., & Tsourakakis, C. E. (2021). Finding densest k-connected subgraphs. *Discrete Applied Mathematics*, *305*, 34–47. https://doi.org/10.1016/j.dam.2021.08.032

Coscia, M., Giannotti, F., & Pedreschi, D. (2011). A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *4*(5), 512–546. https://doi.org/10.1002/sam.10133

Dey, A. K., Tian, Y., & Gel, Y. R. (2022). Community detection in complex networks: From statistical foundations to data science applications. *WIREs Computational Statistics*, *14*(2), e1566. https://doi.org/10.1002/wics.1566

Fortunato, S., & Newman, M. E. J. (2022). 20 years of network community detection. *Nature Physics*, *18*(8), 848–850. https://doi.org/10.1038/s41567-022-01716-7

Haggerty, L. S., Jachiet, P.-A., Hanage, W. P., Fitzpatrick, D. A., Lopez, P., O'Connell, M. J., Pisani, D., Wilkinson, M., Bapteste, E., & McInerney, J. O. (2013). A pluralistic account of homology: Adapting the models to the data. *Molecular Biology and Evolution*, *31*(3), 501–516. https://doi.org/10.1093/molbev/mst228

Henzinger, M., Noe, A., Schulz, C., & Strash, D. (2018). Practical minimum cut algorithms. *ACM Journal of Experimental Algorithmics*, *23*.

Karatas, A., & Sahin, S. (2018, December). Application areas of community detection: A review. *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*. https://doi.org/10.1109/ibigdelft.2018.8625349

Leskovec, J., & Krevl, A. (2014). *SNAP Datasets: Stanford large network dataset collection*. http://snap.stanford.edu/data.

Park, M., Tabatabaee, Y., Ramavarapu, V., Liu, B., Pailodi, V. K., Ramachandran, R., Korobskiy, D., Ayres, F., Chacko, G., & Warnow, T. (2023). Well-connected communities in real-world and synthetic networks. *To Appear, Proceedings of COMPLEX Networks 2023*. https://arxiv.org/abs/2303.02813

Ramavarapu, V., Kamath, V., Park, M., Ayres, F., & Chacko, G. (2023). *Connectivity modifier pipeline*. https://github.com/illinois-or-research-analytics/cm_pipeline.

Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, *105*(4), 1118–1123. https://doi.org/10.1073/pnas.0706851105

Traag, V. A., Waltman, L., & van Eck, N. J. (2019). From Louvain to Leiden: Guaranteeing well-connected communities. *Scientific Reports*, *9*(1), 5233. https://doi.org/10.1038/s41598-019-41695-z

Waltman, L., & van Eck, N. J. (2012). A new methodology for constructing a publication-level classification system of science. *Journal of the American Society for Information Science and Technology*, *63*(12), 2378–2392. https://doi.org/10.1002/asi.22748

Wedell, E., Park, M., Korobskiy, D., Warnow, T., & Chacko, G. (2022). Center–periphery structure in research communities. *Quantitative Science Studies*, *3*(1), 289–314. https://doi.org/10.1162/qss_a_00184