VieClus v1.00 – Vienna Graph Clustering
User Guide

Sonja Biedermann, Monika Henzinger, Christian Schulz and Bernhard Schuster
University of Vienna, *Vienna, Austria*
Email: {christian.schulz}@univie.ac.at

Abstract

This paper severs as a user guide to the graph clustering framework VieClus (Vienna Graph Clustering). We give a rough overview of the techniques used within the framework and describe the user interface as well as the file formats used.

Contents

1 Introduction

Graph clustering is the problem of detecting tightly connected regions of a graph. Depending on the task, knowledge about the structure of the graph can reveal information such as voter behavior, the formation of new trends, existing terrorist groups and recruitment [18] or a natural partitioning of data records onto pages [9]. Further application areas include the study of protein interaction [17], gene expression networks [21], fraud detection [1], program optimization [14, 8] and the spread of epidemics [15]—possible applications are plentiful, as almost all systems containing interacting or coexisting entities can be modeled as a graph.

It is common knowledge that there is no single best strategy for graph clustering, which justifies a plethora of existing approaches. Moreover, most quality indices for graph clusterings have turned out to be NP-hard to optimize and are rather resilient to effective approximations, see e.g. [2, 7, 20], allowing only heuristic approaches for optimization. The majority of algorithms for graph clustering are based on the paradigm of intra-cluster density versus inter-cluster sparsity. One successful heuristic to cluster large graphs is the *multi-level* approach [6], e.g. the Louvain method for the optimization of modularity [5]. Here, the graph is recursively *contracted* to obtain smaller graphs which should reflect the same general structure as the input. After applying an *initial clustering* algorithm to the smallest graph, the contraction steps are undone and, at each level, a *local search* method is used to improve the clustering induced by the coarser level w.r.t some objective function measuring the quality of the clustering. The intuition behind this approach is that a good clustering at one level of the hierarchy will also be a good clustering on the next finer level. Hence, depending on the definition of the neighborhood, local search algorithms are able to explore local solution spaces very effectively in this setting. However, these methods are also prone to get trapped in local optima. The multi-level scheme can help to some extent since local search has a more global view on the problem on the coarse levels and a very fine-grained view on the fine levels of the multi-level hierarchy. In addition, as with many other randomized meta-heuristics, several repeated runs can be made in order to improve the final result at the expense of running time.

Still, even a large number of repeated executions can only scratch the surface of the huge search space of possible clusterings. In order to explore the global solution space extensively, we need more sophisticated meta-heuristics. This is where memetic algorithms (MAs), i.e. genetic algorithms combined with local search [13], come into play. Memetic algorithms allow for effective exploration (global search) and exploitation (local search) of the solution space. The general idea behind genetic algorithms is to use mechanisms inspired by biological evolution such as selection, mutation, recombination, and survival of the fittest. A genetic algorithm (GA) starts with a population of individuals (in our case clusterings of the graph) and evolves the population over several generational cycles (rounds). In each round, the GA uses a selection rule based on the fitness of the individuals of the population to select good individuals and combines them to obtain improved offspring [10]. When an offspring is generated an eviction rule is used to select a member of the population to be replaced by the new offspring. For an evolutionary algorithm it is of major importance to preserve diversity in the population [3], i.e., the individuals should not become too similar in order to avoid a premature convergence of the algorithm. This is usually achieved by using mutation operations and by using eviction rules that take similarity of individuals into account.

Here we release a memetic algorithm, VieClus (Vienna Graph Clustering), to tackle the graph clustering problem. A key component of our contribution are natural recombine operators that employ ensemble clusterings as well as multi-level techniques. In machine learning, ensemble methods combine multiple weak classification (or clustering) algorithms to obtain a strong algorithm for classification (or clustering). More precisely, given a number of clusterings, the *overlay/ensemble clustering* is a clustering in which two vertices belong to the same cluster if and only if they belong to the same cluster in each of the input clusterings. Our recombination operators use the overlay of two clusterings from the population to decide whether pairs of vertices should belong to the same cluster [16, 19]. This is combined with a local search algorithm to find further improvements and also embedded into a multi-level algorithm to find even better clusterings. Our general principle is to randomize tie-breaking whenever possible. This diversifies the search and also improves solutions. Lastly, we combine these techniques with a scalable communication protocol, producing a system that is able to compute high-quality solutions in a short amount

of time. In our experimental evaluation, we show that our algorithm successfully improves or reproduces all entries of the 10th DIMACS implementation challenge under consideration in a small amount of time. In fact, for most of the small instances, we can improve the old benchmark result *in less than a minute*. Moreover, while the previous best result for different instances has been computed by a variety of solvers, our algorithm can now be used as a single tool to compute the result. For more details, we refer the reader to [4].

2 Graph Format

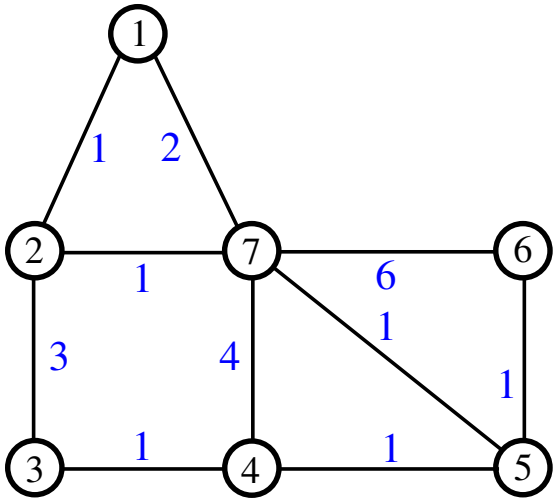
2.1 Input File Format

2.1.1 Metis File Format

The graph format used by our clustering programs is the same as used by Metis [12], Chaco [11] and the graph format that has been used during the 10th DIMACS Implementation Challenge on Graph Clustering and Partitioning. The input graph has to be undirected, without self-loops and without parallel edges.

To give a description of the graph format, we follow the description of the Metis 4.0 user guide very closely. A graph $G = (V, E)$ with n vertices and m edges is stored in a plain text file that contains $n + 1$ lines (excluding comment lines). The first line contains information about the size and the type of the graph, while the remaining n lines contain information for each vertex of G. Any line that starts with % is a comment line and is skipped. The first line in the file contains either two integers, $n\ m$, or three integers, $n\ m\ f$. The first two integers are the number of vertices n and the number of undirected edges of the graph, respectively. Note that in determining the number of edges m, an edge between any pair of vertices v and u is counted *only once* and not twice, i.e. we do not count the edge (v, u) from (u, v) separately. The third integer f is used to specify whether or not the graph has weights associated with its vertices, its edges or both. If the graph is unweighted then this parameter can be omitted. It should be set to 1 if the graph has edge weights. The remaining n lines of the file store information about the actual structure of the graph. In particular, the ith line (again excluding comment lines) contains information about the ith vertex. Depending on the value of f, the information stored in each line is somewhat different. In the most general form (when $f = 1$, i.e. we have edge weights) each line has the following structure:

$$v_1\ w_1\ v_2\ w_2 \ldots v_k\ w_k$$




```
7  10  11
2  1  7  2
1  1  3  3  7  1
2  3  4  1
3  1  7  4  5  1
4  1  7  1  6  1
5  1  7  6
1  2  2  1  4  4  5  1  6  6
```

Figure 1: An example graph and its representation in the graph format. The IDs of the vertices are drawn within the cycle, the vertex weight is shown next to the circle (red) and the edge weight is plotted next to the edge (blue).

where $v_1, \ldots, v_k$ are the vertices adjacent to this vertex, and $w_1, \ldots, w_k$ are the weights of the edges. Note that the vertices are numbered starting from 1 (not from 0). Furthermore, the vertex-weights must be integers greater or equal to 0, whereas the edge-weights must be strictly greater than 0.

2.2 Output File Formats

2.2.1 Clustering

The output format of a clustering is similar to the output format of a partition provided by Metis. It is basically a text file named *tmpclustering*. This file contains n lines. In each line the block ID of the corresponding vertex is given, i.e. line i contains the block ID of the vertex i (here the vertices are numbered from 0 to $n-1$). The block IDs are numbered consecutively from 0 to $n-1$.

2.3 Troubleshooting

VieClus should not crash! If VieClus crashes it is mostly due to the following reasons: the provided graph contains self-loops or parallel edges, there exists a forward edge but the backward edge is missing or the forward and backward edges have different weights, or the number of vertices or edges specified does not match the number of vertices or edges provided in the file. Please use the *graphchecker* tool provided in our package to verify whether your graph has the right input format. If our graphcheck tool tells you that the graph that you provided has the correct format and VieClus crashes anyway, please write us an email.

3 User Interface

VieClus contains the following programs: vieclus, graphchecker, evaluator. To compile these programs you need to have Argtable, g++, OpenMPI and scons installed (we use argtable-2.10, g++-4.8.0, OpenMPI-1.4.1 and scons-1.2). Once you have that you can execute *compile.sh* in the main folder of the release. When the process is finished the binaries can be found in the folder *deploy*. We now explain the parameters of each of the programs briefly.

3.1 VieClus

Description: This is the distributed memetic algorithm to tackle the graph clustering problem.

Usage:
mpirun -n P vieclus file [--help] [--seed=<int>] [--time_limit=<double>] [--output_filename=<string>]

Options:

P	Number of processes to use.
file	Path to graph file that you want to cluster.
--help	Print help.
--seed=<int>	Seed to use for the random number generator.
--time_limit=<double>	Time limit in seconds s. The default value is set to 0s, i.e. one clustering call will be made by each PE. In order to use combine operations you *have to* set a time limit $t > 0$. VieClus will return the best solution after the time limit is reached. A time limit $t = 0$ means that the algorithm will only create the initial population.
--output_filename=<string>	Specify the output filename (default tmpclustering).

3.2 Graph Format Checker

Description: This program checks if the graph specified in a given file is valid.

Usage:
graphchecker file

Options:

file Path to the graph file.

3.2.1 Evaluation

Description: This is the program to compute the modularity of a clustering.

Usage:
./evaluator file --input_partition=<string>

Options:

file	Path to graph file
--input_partition=file	Path to clustering file to evaluate.

References

[1] L. Akoglu, H. Tong, and D. Koutra. Graph Based Anomaly Detection and Description: A Survey. *Data Min. Knowl. Discov.*, 29(3):626–688, May 2015.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*. Springer Science & Business Media, 2012.

[3] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. PhD thesis, 1996.

[4] Sonja Biedermann, Monika Henzinger, Christian Schulz, and Bernhard Schuster. Memetic graph clustering. *CoRR*, abs/1802.07034, 2018.

[5] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[6] U. Brandes. *Network Analysis: Methodological Foundations*, volume 3418. Springer Science & Business Media, 2005.

[7] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008.

[8] J. Demme and S. Sethumadhavan. Approximate Graph Clustering for Program Characterization. *ACM Trans. Archit. Code Optim.*, 8(4):21:1–21:21, January 2012.

[9] A. A. Diwan, S. Rane, S. Seshadri, and S. Sudarshan. Clustering Techniques for Minimizing External Path Length. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 342–353, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[10] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[11] B. Hendrickson. Chaco: Software for Partitioning Graphs. `http://www.cs.sandia.gov/~bahendr/chaco.html`.

[12] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[13] J. Kim, I. Hwang, Y. H. Kim, and B. R. Moon. Genetic Approaches for Graph Partitioning: A Survey. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 473–480. ACM, 2011.

[14] S. McFarling. Program Optimization for Instruction Caches. *SIGARCH Comput. Archit. News*, 17(2):183–191, April 1989.

[15] M. E. J. Newman. Properties of Highly Clustered Networks. *Physical Review E*, 68(2):026121, 2003.

[16] M. Ovelgönne and A. Geyer-Schulz. An Ensemble Learning Strategy for Graph Clustering. In *Graph Partitioning and Graph Clustering*, number 588 in Contemporary Mathematics. 2013.

[17] J. B. Pereira-Leal, A. J. Enright, and C. A. Ouzounis. Detection of Functional Modules from Protein Interaction Networks. *Proteins: Structure, Function, and Bioinformatics*, 54(1):49–57, 2004.

[18] S. E. Schaeffer. Survey: Graph Clustering. *Comput. Sci. Rev.*, 1(1):27–64, August 2007.

[19] C. L. Staudt and H. Meyerhenke. Engineering High-Performance Community Detection Heuristics for Massive Graphs. In *Proceedings 42nd Conference on Parallel Processing (ICPP'13)*, 2013.

[20] D. Wagner and F. Wagner. Between Min Cut and Graph Bisection. In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, pages 744–750. Springer, 1993.

[21] Y. Xu, V. Olman, and D. Xu. Clustering Gene Expression Data using a Graph-Theoretic Approach: an Application of Minimum Spanning Trees. *Bioinformatics*, 18(4):536–545, 2002.