

# A Modular Overlapping Community Detection Algorithm: Investigating the “From Local to Global” Approach

Maximilien Danisch<sup>1</sup>, Noé Gaumont<sup>2</sup>, Jean-Loup Guillaume<sup>3</sup>

<sup>1</sup> Sorbonne Université, CNRS, Laboratoire d’Informatique de Paris 6, LIP6, Paris F-75005, France

<sup>2</sup> Complex Systems Institute of Paris Ile-de-France (ISC-PIF), Paris, France, CAMS, CNRS - EHESS, Paris, France

<sup>3</sup> Laboratoire Informatique, Image et Interaction (L3i), Université de La Rochelle, La Rochelle, France

## Abstract

We propose an overlapping community detection algorithm following a “from local to global approach”: our algorithm finds local communities one by one by repetitively optimizing a quality function that measures the quality of a community. Then, as some extracted local communities can be very similar to each-other, a cleaning procedure is applied to obtain the global overlapping community structure. Our algorithm depends on three modules: (i) a quality function, (ii) an optimization heuristic and (iii) a cleaning procedure. Various such modules can be independently plugged in. We show that, using default modules, our algorithm improves over a state-of-the-art method on some real-world graphs with ground truth communities. In the future we would like to study which combination of modules performs best in practice and make our code parallel.

## 1 Introduction

The Web graph (web pages connected by hyperlinks), Facebook (profiles connected by friendships), Internet (computers connected by Internet connections) and a human brain (neurons connected by synapses) are only a few examples of graphs extracted from the real world.

Designing practical algorithms to find relevant groups of nodes in such graphs has applications ranging from web search to drug design. However, designing such *community detection* algorithms is an extremely challenging task, indeed most real-world graphs are huge making any quadratic time algorithm not practical. In addition, community detection is an ill-defined problem, indeed there is no clear definition of what is a community, i.e. a relevant set of nodes.

In this paper we propose a generic and modular algorithm, called MOCDA for Modular Overlapping Community Detection Algorithm, that allows to compute a set of overlapping communities. This algorithm is based on a "local to global" approach (seed-centric approach) [4] where local communities are expanded around seeds by the repeated addition of nodes. A function is used to assess the quality of each community and an optimisation heuristic is used to optimize it. As two local communities could differ by only a small number of nodes, all the local communities are cleaned to remove the noise caused by similar communities and provide the global overlapping structure of the network. All steps are modular: the quality function, the optimization itself and the cleaning procedures can be modified to fit the needs of the user.

The rest of the paper is organized as follows: in Section 2 we present the algorithm and the different modules, in Section 3 we benchmark our algorithm against existing state-of-the-art method and we conclude in Section 4.

## 2 Algorithm

We detail here our modular algorithm for detecting overlapping communities that relies on three modules: (i) the definition of a function that evaluates the quality of a community, (ii) an optimization heuristic and (iii) a cleaning procedure. An efficient C implementation is available at <https://github.com/maxdan94/mocda> in which options are available to chose different strategies for the three modules.

## 2.1 Quality functions

A quality function is a function that evaluates if a given set of nodes is a good community or not. As there are several definitions of a *good* community, there are several ways to evaluate a set of nodes, such as the clustering coefficient or the conductance. We narrow the set of possible quality functions in MOCDA to those relying on local features to evaluate a given set of nodes or very simple global features. Indeed, a quality function actually does not need a complete knowledge of the graph to evaluate to what extent a set of nodes is a good community.

Formally, we consider quality functions that can be expressed as a function  $f(\phi)$  where  $\phi$  is a set of features among the following:

- $n$ : number of nodes in the graph.
- $m$ : number of links in the graph.
- $t$ : number of triangles in the graph.
- $s$ : number of nodes in the community.
- $l_2$ : number of links with both end nodes in the community.
- $l_1$ : number of links with exactly one node in the community.
- $t_3$ : number of triangles with three nodes in the community.
- $t_2$ : number of triangles with exactly two nodes in the community.
- $t_1$ : number of triangles with exactly one node in the community.

Many quality functions of the literature can be written under that form such as conductance  $\phi = \frac{l_1}{\min(2 \cdot l_2 + l_1, 2 \cdot m - 2 \cdot l_2 - l_1)}$  or cohesion  $C = \frac{t_3}{\binom{s}{3}} \times \frac{t_3}{t_3 + t_2}$  [3].

The reason we bound our algorithm to these features is practical: we will optimize the input function in a greedy way by adding or removing nodes one at a time starting from a small set of nodes, we thus need to be able to evaluate the increase or decrease of the function extremely quickly and these parameters allows to do it. In particular, for the features related to triangles, we rely on the compact-forward algorithm detailed in [5] which allows to list all triangles in very-large real world graphs.

## 2.2 Optimization heuristics

We focus only on greedy and stochastic approaches. Given a node of interest  $u$  and a quality function  $f$ , a possible optimization heuristic consists in the following three steps that can be changed independently in our program.

- **Initialization:** start from a community containing only one node, or two linked nodes, or a node and all its neighbors.
- **Optimization:** at each iteration, add a randomly chosen node, neighbor of the community  $C$ , that increases the quality  $f$ . It is also possible to add the node that leads to the highest increase and/or to authorize the removal of a node.
- **Stop:** stop when the quality function can no longer be increased, i.e., when a local maximum has been reached. It is also possible to add the least quality-decreasing node with the hope that it will improve even more afterwards and return the set of nodes of highest quality obtained.

Therefore, given a quality function and a seed community, the optimization grows this seed to a full community. Since the optimization step is stochastic, two execution starting from the same seed community can give different results.

## 2.3 Cleaning procedures

Since the optimization is to be repeated several times using different seed communities and as two different optimizations can lead to similar final communities, it is important to clean these obtained sets of nodes. There are several ways to do that, which all require to compute the similarity between two sets of nodes. For this we use the  $F_1$  similarity: given two sets  $a$  and  $b$  the similarity is given by  $F_1(a, b) = 2 \frac{|a \cap b|}{|a| + |b|}$ . We say that two sets are similar if their similarity is higher than a given threshold. We use the following cleaning procedures depending on the order the communities are examined: (i) process the obtained sets of nodes on-the-fly or (ii) process the obtained sets of nodes in decreasing order of quality. In both cases a new set of nodes is kept if and only if it is not similar to a previously processed set of nodes.

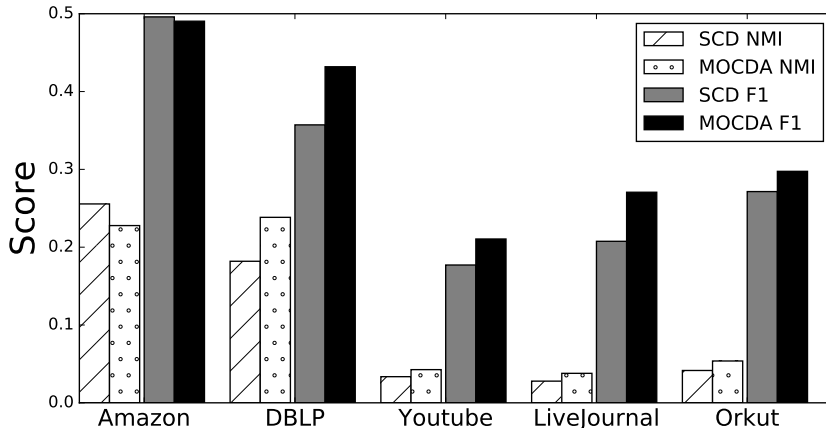


FIG. 1:  $\mathcal{F}_1^{avr}$  and NMI scores using ground truth

Note that other cleaning procedures could also be investigated such as doing the intersection or union of found communities in some way, rather than simply removing sets.

Once the communities are cleaned, the algorithm only outputs the communities that were found to be similar to another one  $k$  or more times. This redundancy test states that a community found only once may be less relevant than a community found a large number of times.

**Implementation details.** Several solutions had to be found in order to obtain a program efficient in terms of time and memory. We were able to check whether a quality function of the type  $f(n, m, t, s, l_1, l_2)$  (resp.  $f(n, m, t, s, l_1, l_2, t_1, t_2, t_3)$ ) increases or decreases under the addition or deletion of a node  $u$  in constant time and proceed to the actual addition or deletion in  $O(d_u)$  time (resp.  $O(\sum_{v \in N(u)} d_v)$ ) in case  $f$  indeed increases while using linear memory. For the cleaning procedure, we were able to compute the maximum  $F_1$ -score between the new community and an already found one in time  $O(s \cdot t)$  in the worst case, where  $s$  is the size of the found community and  $t$  is the maximum number of communities a node belongs to. Note that  $s$  and  $t$  are, in practice, very small compared to the size of the graph leading to a nearly linear cleaning procedure in terms of the number of communities.

### 3 Experimental evaluation

A wide range of methods have been designed to detect communities in graphs and we refer to the following reviews of existing methods [1, 9, 4, 2]. We compared our algorithm using default options to several algorithms including the Louvain method, BIGCLAM and OSLOM but we report the results only for SCD [8] as it performed better and faster than the other methods. SCD has been shown to lead to a non-overlapping and non-exhaustive community structure that is more similar to the real overlapping community structure of some real-world networks than the overlapping community structure unfolded by some state-of-the-art algorithms.

We tried 16 quality functions such as conductance, average degree, edit distance to an isolated clique and cohesion among others. The quality function leading to the best results on all networks was  $\frac{l_2}{n^{1.5}}$ . We also tried several optimization heuristics and cleaning procedures and the best trade-off between time and accuracy is obtained by a stochastic optimization allowing removal of nodes (we do  $2 \cdot n$  optimizations, each time starting from a randomly chosen node) and on-the-fly cleaning (simply outputting all unique communities found at least twice).

To test our algorithm we applied our method on networks with a known community structure [6]. In order to compare the ground truth and the structure found by MOCDA and SCD, we used the Normalized Mutual Information (NMI) [7] and the  $\mathcal{F}_1^{avr}$  [8]. Figure 1 shows the NMI and the  $\mathcal{F}_1^{avr}$  [8]. The larger are these metrics the more similar are the two community structures. Table 1 shows the running time on a laptop with a CPU Intel i7-6500U at 2.50GHz and 16Go of RAM. As we can see, even though our algorithm is slower than SCD, it outperforms it on four of the five datasets. Note that SCD has been shown to be a very competitive method, our algorithm relying on these three simple modules is thus a very promising tool.

| Algorithm | Amazon  | DBLP     | Youtube  | LiveJ.   | Orkut  |
|-----------|---------|----------|----------|----------|--------|
| SCD       | 4.5 sec | 4 sec    | 23 sec   | 8.5 min  | 41 min |
| MOCDA     | 4.9 sec | 10.2 sec | 19.1 min | 83.1 min | 37.9 h |

TAB. 1: Running time comparison

## 4 Conclusion

We proposed a generic and modular algorithm to extract overlapping communities in large networks using a local-to-global approach. This algorithm, using default options, gives better results in most real-world graphs than a state-of-the-art algorithm, even though it is slower.

For future work, we first plan to parallelize the algorithm since the optimization procedure can be performed independently from several seed communities. Furthermore, we want to add more complex features, such as cliques of size more than 3 or other motifs that can be computed efficiently on large networks. We also plan to study in depth the impact of these parameters on the obtained results.

Learning which combinations of modules (quality function, optimization heuristic and cleaning procedure) perform best in an automatic way is also an interesting research perspective.

More important still, the modular design of MOCDA enables anybody to create and test its own quality function which will meet its needs.

## References

- [1] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.
- [2] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 2016.
- [3] Adrien Friggeri, Guillaume Chelius, and Eric Fleury. Triangles to capture social cohesion. In *SocialCom*, pages 258–265. IEEE, 2011.
- [4] Rushed Kanawati. Seed-centric approaches for community detection in complex networks. In *International Conference on Social Computing and Social Media*, pages 197–208. Springer, 2014.
- [5] Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 407(1):458–473, 2008.
- [6] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- [7] Aaron F McDaid, Derek Greene, and Neil Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint*, 2011.
- [8] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-LLuis Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *WWW*, pages 225–236. ACM, 2014.
- [9] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys (csur)*, 45(4):43, 2013.