An Overview of Algorithms for Community Detection in Networks

Author: Adam Słoczyński

Supervisor: Iwona Cieślik PhD

Bachelor's thesis, Computer Science - IT Analyst

JAGIELLONIAN UNIVERSITY

DEPARTMENT OF THEORETICAL COMPUTER SCIENCE

KRAKÓW, JUNE 2019

Abstract

Systems of entities connected by some form of a relationship are omnipresent in our surrounding world. Human circulatory system, the World Wide Web or the food chain are but a few examples of such systems, known as networks. Due to their widespreadness and importance, networks have been a focus of scientific studies for centuries, studies, which have only increased in frequency in recent years. By means of this research several properties of networks have been determined, one of them being the community structure. As it turns out many networks, particularly social ones, tend to consist of communities - groups of entities densely connected with one another, but having relatively few external connections.

In this work we present a simple classification of networks, outline their common properties and, focusing on one of them - the community structure, depict several algorithms for community detection and compare their efficiency as well as accuracy. Specifically, we focus on heuristic algorithms maximizing a certain parameter of a network partition known as modularity, high values of which are indicative of a "good" partition, that is one which closely resembles the desired split of a network into communities.

The algorithms analyzed are those by Blondel et al., Girvan-Newman, Clauset-Newman-Moore, and hierarchical clustering. They are tested using our own implementations and real-world as well as randomly generated datasets. The tests one-sidedly conclude the superiority of the algorithm by Blondel et al., both in the fields of accuracy and time efficiency.

Keywords – network, graph, community, modularity, algorithm

Contents

1 Introduction **1**
1.1 What is a network? . . . 1
1.2 Motivation . . . 1
1.3 Classification and properties of networks . . . 2
1.4 Community detection and modularity . . . 3

2 Algorithms **6**
2.1 Blondel et al. algorithm . . . 6
2.2 Girvan-Newman algorithm . . . 10
2.3 Clauset-Newman-Moore algorithm . . . 13
2.4 Hierarchical clustering . . . 17

3 Testing **20**
3.1 Objectives and methodology . . . 20
3.2 Real-world datasets . . . 24
3.2.1 Zachary's karate club . . . 24
3.2.2 Dolphin social network . . . 26
3.2.3 American College football . . . 26
3.2.4 Network science collaborations . . . 26
3.2.5 Amazon co-purchase network . . . 26
3.2.6 Email network . . . 27
3.2.7 Cumulative results . . . 27
3.3 Synthetic datasets . . . 29
3.3.1 Synthetic community model . . . 29
3.3.2 Social network model . . . 31
3.4 Tests in other publications . . . 34
3.5 Resolution limit . . . 35
3.6 Vertex ordering in Blondel et al. algorithm . . . 36

4 Conclusions **39**

References **41**

1 Introduction

1.1 What is a network?

A network, in simple words, is a group of entities connected by a relation. In literature the term often tends to be used interchangeably with a graph, which is a collection of items, called nodes (or vertices), and edges linking them. We will however differentiate between them by adapting the terminology from [5] and referring to abstract structures comprising nodes and edges as graphs, whereas the term network will denote a real world representation of such structure, in which nodes are entities and edges - relationships between them. Examples of networks include obvious ones, such as the Facebook social network or the World Wide Web but also the network of citations between scientific papers or the food chain (with a directed edge existing between two organisms if and only if one feeds on another).

1.2 Motivation

It was in 1736, when Leonard Euler resolved the famous problem of the seven bridges of Königsberg, laying foundations for future developments of graph (and network) theory. The science, which has flourished ever since, has experienced an especially rapid growth in recent years, particularly due to the development of information technology and the dawn of the Internet. However, the study of networks has not been purely mathematical. Defined as broadly as they are, they have been a focus of many disciplines including psychology, economics, sociology and biology.

Among network related problems, detecting communities, that is groups of nodes with dense internal and sparse external connections, is one of unparalleled importance, with clear interdisciplinary implications. Community detection can be used in criminology [9], targeted marketing [9] and to analyze neural function [7], among many others.

1.3 Classification and properties of networks

Mark Newman [16] classifies networks into the following groups:

- Social networks, in which nodes represent individuals or groups of people and edges stand for some form of connection between them, be it friendship, acquaintance or business relationship
- Information networks, in which edges represent the structure and relationship between information stored in nodes
- Technological networks - man-made networks representing distribution of resources
- Biological networks - naturally existing, graph-like biological systems

Largely on the ground of empirical analysis, several commonalities have been established between networks [16]. Some of those properties we outline below:

- The small world effect: the mean shortest path value between pairs of vertices in a network is small or, as some define more precisely, for fixed mean degree the mean shortest path value scales logarithmically with the network size.
- Large triangle density: triangle density (also clustering coefficient), defined as:

$$D = \frac{3 \times \text{number of triangles in the network}}{\text{number of paths of length 2}} \tag{1.1}$$

  is generally larger in networks than in random graphs. In social networks this fact is reflected by a known rule, that a friend of your friend is disproportionately likely to be your friend as well.
- Degree correlations: networks tend to be either assortative, which means that the mean degree of a neighbor of a node is larger for nodes which themselves have large degree, or disassortative, which indicates an opposite dependency. As Newman [16] states, assortativity is found in social networks, while disassortativity tends to characterize other network types.
- Power law degree distribution: many networks are known to be scale-free, which means that they consist of a few nodes with large degrees and a long tail of nodes

with small degree and their degrees follow a power law distribution, that is, for large values of k, the probability of a random node having a degree of exactly k can be expressed as: $P(k) \sim k^{-\gamma}$, where γ is a parameter typically in range $[2, 3]$

- Community structure: it is commonly believed that networks, particularly social ones, can be divided into dense subnetworks, known as communities, that are sparsely connected with other nodes.

It needs to be mentioned that not all of these properties apply to all network classes. An instance that stands out is the small triangle density in some information networks. To exemplify, let's observe that, since no publication can cite another that has not been written yet, citation networks, a type of information network, are acyclic and hence contain no triangles. Citation networks, however, possess a clear community structure, with communities corresponding to disciplines, to which the cited papers relate.

1.4 Community detection and modularity

As already mentioned, a community is a subset of a network vertices, with dense internal and sparse external connections. What exactly is meant by "dense" and "sparse" is imprecise. There have been many attempts to clarify the definition, which consequently led to a wide array of different algorithms for community detection, which is a problem of finding an unspecified number vertex subsets in a network, that best suit whatever definition of a community is applied. The definition of a community used throughout this paper will be one based on a certain parameter of a network partition known as modularity. Modularity coefficient of a network partition is defined as:

$$Q = \frac{1}{2m} \sum_{i,j} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j) \,, \tag{1.2}$$

where i, j denote vertices, A_{ij} is the appropriate entry of the adjacency matrix of the graph representing the network, m stands for the number of edges, k_i, k_j are degrees of corresponding nodes, c_i, c_j are communities of nodes i, j and δ is the Kronecker delta defined as:

$$\delta(x, y) = \begin{cases} 0 & \text{if } x \neq y \\ 1 & \text{if } x = y \end{cases} . \tag{1.3}$$

To better understand the formula, let us observe that, considering random attachment of edges to a graph with predefined vertex degrees, the expected value of a random variable X representing the total number of edges between nodes belonging to the same community can be expressed as:

$$\mathbb{E}(X) = \frac{1}{2}\sum_{i,j}\frac{k_i k_j}{2m}\delta(c_i, c_j)\,, \tag{1.4}$$

This becomes evident once we note that the probability of an edge getting attached to a pair of nodes i and j is:

$$P(a_{ij}) = \frac{k_i k_j}{2m} \tag{1.5}$$

and thus:

$$\mathbb{E}(X) = \frac{1}{2}\sum_{i,j}P(a_{ij})\delta(c_i, c_j)\,, = \frac{1}{2}\sum_{i,j}\frac{k_i k_j}{2m}\delta(c_i, c_j) \tag{1.6}$$

while the factor $\frac{1}{2}$ is there to nullify the effects of double counting for pairs (i, j) and (j, i). On the other hand, the actual number of edges with both ends in the same community is:

$$\frac{1}{2}\sum_{i,j}A_{ij}\delta(c_i, c_j) \tag{1.7}$$

and therefore the modularity coefficient is the difference between the actual number of edges with both ends in the same community in the network and the expected value of the same parameter in a random graph with degree sequence equal to the degree sequence of the network whose modularity is calculated. The value is normalized by the factor of $\frac{1}{m}$ so that it is always between -1 and 1. Defined as such, modularity reflects the density of connections inside communities with relation to the expected number of connections within them.

Equivalently to formula 3.5, modularity can be expressed as:

$$Q = \sum_r e_{rr} - a_r^2\,, \tag{1.8}$$

where:

$$e_{rs} = \frac{1}{2m}\sum_{ij}A_{ij}\delta(c_i, r)\delta(c_j, s)\,, \tag{1.9}$$

$$a_r = \frac{1}{2m}\sum_i k_i\delta(c_i, r) \tag{1.10}$$

and r, s are communities.

Thus, a community is an element of a network partition, which maximizes the modularity value and so the problem of community detection is equivalent to modularity maximization. It can be observed, that for this definition to work, each vertex has to be assigned to exactly one community, which implies that communities cannot overlap and no vertex can be left without a community.

Modularity maximization is a hard problem. The number of distinct partitions of a graph is equal to the Bell number of the size of the graph and is thus exponential. All the known algorithms capable of always finding a maximum modularity partition run in exponential time [19]. However, there exist numerous heuristic solutions to modularity maximization problem that give acceptable results. Some of these we present in the following chapter, while test results comparing their accuracy and efficiency as well as reflecting some known limitations are described in chapter 3.

2 Algorithms

2.1 Blondel et al. algorithm

Blondel et al. algorithm (also known as Louvain Modularity) was first proposed by Blondel, Guillaume, Lambiotte and Lefebvre in their article "Fast unfolding of communities in large networks" [1]. It is a heuristic algorithm for the modularity maximization problem. Designed to work on weighted networks, it can be easily generalized to unweighted ones, by assuming each edge in an unweighted network to be weighted with weight equal to one. The algorithm consists of two phases repeated interchangeably until an end condition is achieved.

Since the network processed by the algorithm is weighted, the utilized formula for modularity calculation differs slightly from the formula 3.5. Namely, instead of considering regular node degrees in the expression, we use weighted degrees, where weighted degree of a node is defined as a sum of weights of all edges incident to the node. We can also modify the equation 1.8 to make it applicable to weighted graphs. The formulae suited for partitions of weighted networks are as follows:

$$Q = \frac{1}{2m} \sum_{i,j} (w_{ij} - \frac{d_i d_j}{2m}) \delta(c_i, c_j) \tag{2.1}$$

where d_i, d_j are weighted degrees of corresponding nodes and w_{ij} is the weight of an edge between i and j if such edge exists, 0 otherwise. This is equivalent to:

$$Q = \sum_r e'_{rr} - {a'_r}^2 , \tag{2.2}$$

Where:

$$a'_r = \frac{\sum_i d_i \delta(c_i, r)}{\sum_i d_i} , \tag{2.3}$$

$$e'_{rs} = \frac{\sum_{i,j} w_{ij} \delta(r, c_i) \delta(s, c_j)}{\sum_i d_i} . \tag{2.4}$$

We start the first phase of the algorithm by assigning each node of the network to its own community. Then, we repeatedly iterate over all nodes and move each of them to a neighboring community, which is a community containing one or more neighbors of the vertex, in a way that maximizes the modularity gain over all possible assignments to neighboring communities. Ties are resolved arbitrarily and if no (positive) modularity gain can be obtained, the processed vertex remains in its old community. This procedure is repeated continuously for all nodes until no modularity improvement can be achieved by moving any single vertex to a neighboring community. The authors observe that the ordering in which the vertices are processed throughout this phase can considerably impact the running time of the algorithm but has no significant bearing on the result. However, no methods to improve the ordering over an arbitrary one have been proposed. We illustrate this dependency in section 3.6.

The second phase is the creation of a new graph G' from the original graph G. We do it by adding one node to G' per each community in G and assuming the nodes to represent their respective communities. As for the edges, for each node a in G', representing a community α, we attach to a a weighted self-loop with weight equal to sum of weights of all edges in G with both ends in community α, whereas for each pair of distinct nodes a and b in G', corresponding to communities α and β accordingly, we add an undirected, weighted edge between a and b with weight equal to sum of weights of all edges in G with one end in community α and the other in community β. The second phase ends with substituting G' for G.

We repeat the first and second phase until no modularity improvement is achieved throughout the whole first phase, which indicates that the algorithm has found the maximum modularity partition attainable by repeated application of the two phases. The result of the algorithm is said partition, which we can then use to calculate the corresponding modularity value.

To explain the reasoning behind the repeated application of two phases, let us observe that, while at a certain stage of the procedure there may be no singular vertex such that moving it to a different community would result in a modularity increase, it is entirely possible that merging whole communities, larger with every iteration, would improve the modularity of the partition. What repeating phases one and two does, is allowing us to

merge together larger and larger clusters of vertices during a single step.

It is important to remark that the algorithm never requires an explicit calculation of modularity. This is due to the fact that it only considers the potential change of modularity, instead of its actual value (see algorithm 1 line 18 below). Because of that, each time we consider a node to be moved we only need to calculate two elements of the sum 2.2, as all the others remain unchanged. Assuming we store and update values of parameter a'_r, for all communities, this requires a number of operations linear in the number of neighbours of the processed node and is thus significantly faster than calculating the whole formula.

While no rigorous complexity analysis of Blondel et al. algorithm has ever been conducted, the authors claim [1], based on empirical tests, that it runs in linear time on sufficiently sparse networks.

Below, we illustrate the algorithm with pseudocode.

Algorithm 1 Blondel et al.

▷ Let V_G - a set of vertices of G
▷ Let E_G - a set of edges of G
▷ Let W_G - a dictionary mapping edges in E_G to their weights; all edges in E_G mapped to 1 if G is unweighted
▷ Let $neighbors(v, G)$ - a set of all vertices u such that $(v, u) \in E_G$
procedure BLONDEL($G = (V_G, E_G, W_G)$)
    repeat
        $c \leftarrow$ PHASE1(G)
        $G \leftarrow$ PHASE2(G, c)
    until no change in c
    return c
end procedure

procedure PHASE1($G = (V_G, E_G, W_G)$)
    $c \leftarrow \{\{v\} \mid v \in V_G\}$ ▷ Let $c(u)$ - element of c such that $u \in c(u)$
    repeat
        for all $v \in V_G$ **do**
            for all $u \in neighbours(v, G) \setminus \{v\}$ **do**
                $\Delta \leftarrow$ modularity change when moving v from $c(v)$ to $c(u)$
                if $\Delta > 0$ **then**
                    $c(v) \leftarrow c(v) \setminus \{v\}$
                    $c(u) \leftarrow c(u) \cup \{v\}$
                end if
            end for
        end for
    until no change in c
    $c \leftarrow c \setminus \{\emptyset\}$
    return c
end procedure

procedure PHASE2($G = (V_G, E_G, W_G), c$)
    $G' \leftarrow (V_{G'}, E_{G'}, W_{G'})$
    $V_{G'} \leftarrow c$
    $E_{G'} \leftarrow \emptyset$
    $W_{G'} \leftarrow$ empty dictionary
    for all $(u, v) \in E_G$ **do**
        if $(c(u), c(v)) \in E_{G'}$ **then**
            $W_{G'}[(c(u), c(v))] \leftarrow W_{G'}[(c(u), c(v))] + W_G[(u, v)]$
        else
            $E_{G'} = E_{G'} \cup \{(c(u), c(v))\}$
            $W_{G'}[(c(u), c(v))] \leftarrow W_G[(u, v)]$
        end if
    end for
    return G'
end procedure

2.2 Girvan-Newman algorithm

The community detection algorithm by Girvan and Newman was introduced in their publication "Community structure in social and biological networks" [8]. It is centered around the idea of identifying edges that connect vertices in different communities. As communities are sparsely linked with one another, such edges often have the property of being contained in many shortest paths between distinct pairs of vertices. This notion can be formalized by the term of betweenness centrality of an edge, defined as:

$$C(u,v) = \sum_{s,t} \frac{\sigma_{st}(u,v)}{\sigma_{st}}, \tag{2.5}$$

where (u,v) is an edge, s and t are vertices, σ_{st} is the number of shortest paths between s and t and $\sigma_{st}(u,v)$ denotes how many of these paths contain the edge (u,v).

Based on the assumption that edges connecting communities have high values of betweenness centrality, the algorithm processes a network by progressively removing edges with the highest value of this parameter and assuming communities to correspond to connected components of the modified network. When to stop the procedure and consider the community partition to be final is not at all obvious. In their original paper [8], predating the concept of modularity, Girvan and Newman suggested running the algorithm until no edges remain and analyzing the structure of the network after each edge removal, to then decide which partition was the most appropriate one. In this form, the algorithm required the investigator to have some partial knowledge of what the community structure of the network should be, which typically meant knowing the expected number or sizes of communities. An example of such use of the algorithm could be grouping users of a social network by their nationality. Assuming we know how many nationalities are represented, we also know how many communities the network should be partitioned into, despite not knowing their exact structure. In such a case, we have the knowledge of when the algorithm should stop. When no such data was available, some selection scheme had to be applied to choose the best partition of the ones suggested by the algorithm. However, before modularity was introduced, different measures were used for that purpose and the authors do not suggest any in particular.

After the modularity coefficient was first defined in a further publication of Girvan and Newman [20], a modularity based selection scheme was proposed by Newman [17]. Newman suggests running the algorithm until the deletion of all edges but also computing modularity after each edge removal. Importantly, the network used for modularity calculation is the base, unmodified network and the community partition is the one defined by connected components of the network after the most recent edge deletion. Finally, once the procedure has concluded, we select the partition corresponding to the highest modularity value.

The calculation of betweenness centrality for all edges can be done using a simple modification of Brandes algorithm [2] in $O(mn)$ time, where m is the number of edges and n is the number of nodes. The algorithm, based on repeated breadth-first search traversals, is designed to calculate betweenness centality of vertices but can be easily adapted to calculate edge centrality as well. The modified Brandes algorithm works as follows:

1. Betweenness centrality $C(u, v)$ of every edge is set to 0.
2. A vertex s is selected as source, distance d_{su} from s to u is set to ∞ for all vertices $u \neq s$ in the graph, d_{ss} is set to 0 and σ_{su} - the number of shortest paths between s and u is set to 0 for all $u \neq s$, σ_{ss} is set to 1. Lastly, $\delta_s(u)$ is set to 0 for all u.
3. A breadth-first search traversal takes place: vertex s is put in queue and the following steps are repeated until the queue is empty:
   3.1. v is taken off the top of the queue.
   3.2. σ_{sv} is increased by σ_{su} for all $u \in neighbors(v)$ such that $d_{su} = d_{sv} - 1$.
   3.3. A vertex u is added to the queue for all $u \in neighbors(v)$ such that $d_{su} = \infty$.
   3.4. d_{su} is set to $d_{sv} + 1$ for all $u \in neighbors(v)$ such that $d_{su} = \infty$.
4. The following step is repeated for all vertices v in descending order of d_{sv}:
   4.1. For all $u \in neighbors(v)$ such that $d_{su} = d_{sv} - 1$: $\delta_s(u)$ and $C(v, u)$ are both increased by $\frac{\sigma_{su}}{\sigma_{sv}}(1 + \delta_s(v))$.
5. Steps 2 - 4 are repeated for all vertices s in the graph.
6. C_{uv} is divided by two for all edges (u, v) to nullify the effect of double counting.

We can observe that step 3 serves to calculate the number of shortest paths from s to u for all u and a fixed s. On the other hand, in step 4 we calculate values of δ and C, where:

$$\delta_s(u) = \sum_{s \neq v \neq u} \frac{\sigma_{sv}(u)}{\sigma_{sv}}\,, \tag{2.6}$$

where $\sigma_{sv}(u)$ is the number of shortest paths between s and v containing vertex u. We can observe that $\delta_s(u)$ follows a certain recursive relation:

$$\delta_s(u) = \sum_{w:u \in P_s(w)} \frac{\sigma_{su}}{\sigma_{sw}}(1 + \delta_s(w))\,, \tag{2.7}$$

where $P_s(w)$ is the set of all $u \in neighbors(w)$ such that $d_{su} = d_{sw} - 1$. The detailed reasoning behind the equation 2.7 can be found in [2]. The formula and the reasoning behind it can be generalized to $C_s(u, w)$ defined as:

$$C_s(u, w) = \sum_t \frac{\sigma_{st}(u, w)}{\sigma_{st}}\,, \tag{2.8}$$

which also follows a similar relation:

$$C_s(u, w) = \frac{\sigma_{su}}{\sigma_{sw}}(1 + \delta_s(w))\,, \tag{2.9}$$

for $w : u \in P_s(w)$. Finally:

$$C(u, w) = \sum_s \frac{C_s(u, w)}{2}\,, \tag{2.10}$$

which produces the desired betweenness centrality.

During Girvan-Newman algorithm, each edge removal may impact betweenness centrality of other edges in the network but only those in connected components of the two vertices incident to the deleted edge. Thus, after each deletion, betweenness centrality needs to be recomputed using Brandes algorithm on these components and so the worst-case time complexity of the algorithm is $O(m^2n)$ or $O(n^3)$ on typical, sparse networks. In most real world scenarios, however, the algorithm performs better than the worst-case. As networks with distinct community structure have sparse connections between communities, the algorithm splits them into separate components after just a few iterations of edge removal and so the betweenness recalculation can be significantly faster than $O(mn)$ in further

passes of the procedure.

Algorithm 2 Girvan-Newman

```
1: ▷ Let V_G - a set of vertices of G
2: ▷ Let E_G - a set of edges of G
3: procedure GIRVAN-NEWMAN(G)
4:     G' ← G
5:     c_max ← {V_G}
6:     m_max ← modularity(G', c_max)
7:     C ← betweenness centrality of all edges in G'
8:     while E_G' ≠ ∅ do
9:         (u, v) ← edge in E_G' for which C is maximal
10:        E_G' ← E_G' \ {(u, v)}
11:        update C for all edges in connected components of u and v
12:        c ← a set of connected components of G'
13:        if modularity(G, c) > m_max then
14:            m_max ← modularity(G, c)
15:            c_max ← c
16:        end if
17:    end while
18:    return c_max
19: end procedure
```

2.3 Clauset-Newman-Moore algorithm

The algorithm, first suggested by Newman [15] and later improved by Clauset, Newman and Moore [4], is a straightforward, greedy modularity maximization method. We begin the procedure by assigning each node to its own singleton community. We then merge two communities, the union of which results in the highest (positive) modularity gain. A step, which we later repeat until there is no such pair of communities that joining them would result in a modularity increase. At this point, the resulting partition is returned.

Discussing the algorithm, we use a term of community adjacency. A vertex u is adjacent to a community c if $u \notin c$ and there exists a vertex $v \in c$ such that u is adjacent to v. Two communities c and c' are adjacent if $c \neq c'$ and there exists a pair of adjacent vertices u and v, such that $u \in c$ and $v \in c'$

A naive implementation on the algorithm runs in $O(mn)$ time and involves storing and updating potential modularity gains for all pairs of communities connected by at least

one edge so that they do not need to be recalculated at each step of the method. We can, however, improve this complexity by utilizing an alternative approach relying heavily on the sparsity of typical networks and their distinct community structure. When processing a network, for all pairs of communities i and j connected by at least one edge, we keep track of values of modularity change Δ_{ij}, resulting from merging the respective pairs of communities. We also store values of a_i, which denote a fraction of edge ends attached to vertices in community i among all edge ends in the network and can therefore be expressed as:

$$a_i = \frac{1}{2m} \sum_{v \in i} k_v \,, \tag{2.11}$$

where k_v denotes the degree of vertex v. To maintain and update these values, we utilize the following data structures:

- A number of max heaps H_i, one per each community, each of them storing values Δ_{ij}, where i is the community corresponding to the heap and j goes over all communities adjacent to i. Each max heap supports removal of any item by reference, in $O(log(n))$ time, alongside all the other operations typically supported by a max heap.
- A single max heap H_{max} for storing the top elements of all the max heaps H_i, mentioned above. The max heap should also support direct deletion of referenced elements.
- A number of hash maps M_i, one per each community, mapping pairs of $\{i, j\}$ to references to their respective Δ_{ij} values stored in heaps. Each hash map should only contain values for pairs where i is the community corresponding to it and j goes over all communities adjacent to i.
- An array a for storing values of a_i.

Our implementation differs slightly from that outlined in [4], which uses balanced binary search trees instead of hash maps. The change worsens the worst-case complexity of the algorithm but it does not impact the average-case complexity.

Initially, each vertex is placed in its own community and the appropriate structures are initialized. The starting values of Δ_{ij} and a_i are as follows:

$$\Delta_{ij} = \frac{1}{2m} - \frac{k_i k_j}{4m^2} \,, \tag{2.12}$$

and

$$a_i = \frac{k_i}{2m}\,, \tag{2.13}$$

where m is the number of edges in the network and k_i is the degree of vertex i. The values are calculated and later updated based on the formula 1.8 expressing the modularity coefficient of a network partition. To decide, which communities to merge, we look at the top element of H_{max}, remove it from the heap and join the corresponding communities i and j into a new community i. We create a new max heap H_i' and populate it with elements Δ_{ik}', where k goes over all communities adjacent to the union of communities i and j, initialized as follows:

$$\Delta_{ik}' = \begin{cases} \Delta_{ik} + \Delta_{jk}, & \text{if both } i \text{ and } j \text{ are adjacent to } k \\ \Delta_{ik} - 2a_j a_k, & \text{if only } i \text{ is adjacent to } k \\ \Delta_{jk} - 2a_i a_k, & \text{if only } j \text{ is adjacent to } k \end{cases} . \tag{2.14}$$

We add the top element of the newly created H_i' to H_{max}. We then substitute H_i' for H_i and update M_i to map pairs $\{i, j\}$ to references to values in new H_i. We also remove entries Δ_{ki} and Δ_{kj} from H_k for all values of k and in their place insert Δ_{ki}', where $\Delta_{ki}' = \Delta_{ik}'$ is defined as above. We can quickly access the elements to remove with hash maps M_k and delete them using the special property of max heaps H_k. Modifying heaps H_k may change their top elements, in which case we need to update H_{max}, which we achieve by removing the previous top element of H_i from H_{max} and inserting to it the new top element. Finally, the merge step ends with setting:

$$a_i = a_i + a_j \tag{2.15}$$

and removing a_j from the array as well as deleting H_j and M_j. We repeat the whole sequence of operations above, until the top element of H_{max} is nonpositive or H_{max} is empty. One can observe, that once the top element of H_{max} is nonpositive, and thus all elements Δ_{ij} in all heaps are less or equal to 0, the Δ_{ij} values will never increase, as evidenced by equation 2.14. Therefore, we can end the procedure at this stage.

We can elegantly represent the progressive community merges in form of a dendrogram. A dendrogram is a diagram representing a tree, the leaves of which, in case of this

algorithm, correspond to singleton communities and each non-leaf node represents a union of its children, which were merged during the procedure. Since the algorithm usually concludes before all the vertices are merged into one large community, instead of as a single dendrogram, we have to represent it as a "dendrogram forest". Within it, each singular dendrogram corresponds to one community from the final community decomposition returned by the algorithm. Such representation allows us to reproduce the steps of the procedure and will also be useful in analyzing its computational complexity.

At each step of the algorithm, when merging communities of i and j adjacent to n_i and n_j vertices respectively, we create a new heap of size at most $n_i + n_j$ which takes $O(n_i + n_j)$ time. There are also at most $n_i + n_j$ updates to H_{max} and H_k for different values of k, in total taking $O((n_i + n_j)\log(n))$ time. Looking at the dendrogram forest of the procedure, let us denote by the term "forest level" l, the set of all communities that are at the tree level l in trees corresponding to their respective dendrograms. Let us observe that the communities at the same forest level are all pairwise disjoint and thus the sum of degrees of vertices contained in them is at most $2m$. This means that the merges at each level of the dendrogram forest require at most $2m$ updates to heaps, taking $O(m\log(n))$ time per level, and so the total running time of the algorithm is $O(dm\log(n))$, where d is the height of the highest tree in the dendrogram forest. Importantly, it is possible that $d \sim n$, in which case the algorithm complexity is $O(mn\log(n))$ and is actually worse than that of the naive approach. However Clauset at al. [4] claim that in cases of most networks $d \sim \log(n)$ and $m \sim n$, which indicates it to be a significantly faster, $O(n\log^2(n))$ method.

Unlike the implementation of Clauset et al., ours also requires at most $O(n_i + n_j)$ accesses and insertions to a hash table of size $O(n_i + n_j)$ after each merge, which is worst-case $O((n_i + n_j)^2)$ and average case $O(n_i + n_j)$ operations at each step. This results in an additional factor of $O(mnd)$ operations for the worst-case and $O(md)$ for the average-case scenario, changing the worst-case complexity of our approach to $O(mn(d + \log(n)))$. Meanwhile the use of balanced binary search trees results in $O((n_i + n_j)\log(n))$ additional operations per step of the procedure, which is $O(md\log(n))$ in both average and worst case and thus does not affect the complexity of the algorithm.

Algorithm 3 Clauset-Newman-Moore

▷ Let V_G - a set of vertices of G
▷ Let E_G - a set of edges of G
procedure CLAUSET-NEWMAN-MOORE(G)
    $c \leftarrow \{c_i : c_i = \{v_i\} \mid v_i \in V_G\}$
    $H \leftarrow$ array of max heaps H_i, each containing Δ_{ij} for all $j \in neighbors(i)$
    $H_{max} \leftarrow$ max heap of top elements of H_i for all i
    $M \leftarrow$ array of hash maps or BSTs M_i mapping i, j to references to Δ_{ij}
    $a \leftarrow \{a_i \mid a_i = \frac{k_i}{2m}\}$
    while H_{max} is not empty and $top(H_{max}) > 0$ **do**
        $\Delta_{ij} \leftarrow top(H_{max})$
        $pop(H_{max})$
        $remove(H_{max}, top(H_j))$
        $c_i \leftarrow c_i \cup c_j$
        $c \leftarrow c \setminus \{c_j\}$
        $\Delta' \leftarrow$ updated Δ_{kl} for $k \in \{i, j\}$ and l in set of communities adjacent to $c_i \cup c_j$
        $H_i \leftarrow$ max heap populated with all elements of Δ'
        $M_i \leftarrow$ hash map or BST updated with elements of Δ'
        $push(H_{max}, top(H_i))$
        for all $\Delta'_{kl} \in \Delta'$ **do**
            $\Delta_{top} \leftarrow top(H_l)$
            $remove(H_l, \Delta_{lk})$ ▷ Δ_{lk} accessed through M_l
            $push(H_l, \Delta_{li} \leftarrow \Delta'_{lk})$
            if $\Delta_{top} \neq top(H_l)$ **then**
                $remove(H_{max}, \Delta_{top})$
                $push(H_{max}, top(H_l))$
            end if
        end for
        $a_i \leftarrow a_i + a_j$
    end while
    return c
end procedure

2.4 Hierarchical clustering

Hierarchical clustering is one of the oldest community detection algorithms that long predates the concept of modularity. The idea behind it is to compute a selected similarity function for all pairs of vertices of network G. Then, starting with an edgeless network G' with nodes corresponding to nodes in G, we add edges to G' in order of decreasing similarity between pairs of vertices in G. We stop the procedure at some point and analyze the structure of G' to decide the community partition of G.

The most common similarity measures between pairs of vertices are the number of vertex (or edge) independent paths joining them and the Euclidean distance between their corresponding rows in network's adjacency matrix. Vertex (edge) independent paths are paths that share no vertex (edge) with each other. On the other hand, Euclidean distance between vertices i and j can be expressed as:

$$d_{ij} = \sqrt{\sum_{k \neq i,j} (A_{ik} - A_{jk})^2}\,, \tag{2.16}$$

where A is the adjacency matrix. Importantly, Euclidean distance is actually a dissimilarity measure and thus, in hierarchical clustering algorithm, edges should be added in increasing, not decreasing, order of Euclidean distance.

There are also several strategies of extracting community partition from the network generated by the procedure. One of them, called simple linkage, identifies the communities by connected components of the network. A different method, complete linkage, considers communities to correspond to maximal cliques, where a maximal clique is a subset of vertices all adjacent to one another that is not contained in any other such subset. While, according to Newman [17], this method produces better results, it is also computationally hard. A graph may have exponentially many maximal cliques and the problem of enumerating them is known to be NP-complete. Also, a vertex may belong to more than one maximal clique and so the community assignment produced is not unique. This, while beneficial in many cases, in others requires an additional heuristic to guarantee the uniqueness of the attribution.

As for the end condition, the algorithm does not provide any guidelines as to which of the generated partitions is the best one. Similarly to Girvan-Newman algorithm, we can run it until a complete graph is reached and extract the community structure from it after each edge insertion. Then, using some additional knowledge about the expected partitioning: the number of communities, their sizes or any other parameter, we can choose the most fitting of the results. Alternatively, we can run it until a complete graph is attained while taking note of the community decomposition after each edge addition and then select the one which results in the highest modularity when applied to the original graph.

In our implementation we use Euclidean distance as the similarity function, simple linkage

as community extraction method and the modularity based approach to determining the best community decomposition. The pseudocode for this version of the algorithm is presented below:

Algorithm 4 Hierarchical clustering

```
1: ▷ Let V_G - a set of vertices of G
2: ▷ Let E_G - a set of edges of G
3: procedure CLUSTERING(G)
4:     d ← {d_uv : d_uv = euclidean_distance(u, v) | u, v ∈ V_G}
5:     V_G' ← V_G
6:     E_G' ← ∅
7:     G' ← (V_G', E_G')
8:     c_max ← {{v} | v ∈ V_G}
9:     m_max ← modularity(G, c_max)
10:    while d ≠ ∅ do
11:        (u, v) ← pair of vertices in V_G such that d_uv is minimal in d
12:        E_G' ← E_G' ∪ {(u, v)}
13:        d ← d \ {d_uv}
14:        c ← a set of connected components of G'
15:        if modularity(G, c) > m_max then
16:            m_max ← modularity(G, c)
17:            c_max ← c
18:        end if
19:    end while
20:    return c_max
21: end procedure
```

The complexity of the algorithm is dominated by the similarity calculation which requires $O(n^3)$ operations when Euclidean distance is used as the measure of similarity.

3 Testing

3.1 Objectives and methodology

For testing, we use our own implementations of the algorithms, all written in Python 3.6. The tests are run on a computer with AMD Ryzen 5 1600 Six-Core Processor running at 3.2GHz using 16 GB of RAM, running Ubuntu 18.04.

When testing the algorithms, we focus on three main areas:

- How high the modularity values for the partitions returned are.
- How well the partitions returned correspond to the established, ground truth community structure of the networks, in cases where such ground truth data is available.
- The execution time of the algorithms.

In order to comparatively analyze the performance of the algorithms with regard to aforementioned areas, we run them on both documented, real-world networks and synthetic, randomized graphs, generated in accordance with two different models.

The real-world networks we use in our testing are taken from a website of Mark Newman [14] as well as from Stanford Large Network Dataset Collection [11]. The datasets analyzed include well studied Zachary's karate club network, American College football network and the network of coauthorships between network theory scientists, among others. A few of the networks used for testing have a known, ground truth community structure.

The randomized graphs used are generated based on two models: a social network model suggested by Toivonen et al. [21] and a commonly used model we shall call a synthetic community model, which constructs networks based on predefined community structure among other parameters. The model is mentioned in [1], [8] and [10].

The social network model progressively constructs a network utilizing a combination of random and preferential attachment and produces networks possessing most of the known social network properties such as high triangle density, power law degree distribution, average shortest path length scaling logarithmically with network size and,

most importantly, a distinct community structure. The model utilizes the following steps to produce a graph:

1. A base network with an arbitrary number of vertices and connections is selected. In our tests we use a simple path of 5 vertices, but different seed networks may be used.
2. A vertex v is added to the network.
3. An average of $m_p \geq 1$ randomly selected vertices other than v are connected to v as primary contacts.
4. An average of $m_s \geq 0$ randomly selected neighbors of each primary contact other than v are connected to v.
5. Steps 2 - 4 are repeated until the network has reached the required size.

The numbers m_p and m_s are both expected values of probability distributions that parametrize the model. In some cases, it may not be possible for step 4 to execute fully. This is due to what the authors call the saturation effect. When step 4 is executed, a certain number p of neighbors of a primary contact u are to be assigned to vertex v as secondary contacts. It may, however occur that the number of neighbors of u that have not yet been connected to v is p', where p' is strictly less than p. In such a case we consider vertex u to be saturated (with regards to vertex v) and connect only the p' neighbors of u to v, disregarding the remaining $p - p'$ vertices that were also supposed to be connected to v. This happens most commonly when the degree of u is low and when neighbors of u already become connected to v either as primary contacts in step 3 or as secondary contacts in step 4.

The synthetic community model aims at producing networks with distinct community structure, the details of which can be determined by model parameters. In this model we start with a set of vertices of the graph and the desired community partition C. Then, we add an edge between each pair of nodes with probability p_{in} if the nodes are in the same community and with probability p_{out} otherwise. To maintain a proper community structure, the value of p_{in} must be significantly higher than p_{out}.

With appropriate values of p_{in} and p_{out} the maximum modularity partition of the network

will, with high probability, correspond to the partition C selected as the parameter of the model. Thus, we can use C as the ground truth community structure of the network and compare community detection algorithms on the basis of how well the partitions returned by them correspond to C. While it is possible for C to not be the maximum modularity partition, its modularity value establishes a lower bound for maximum modularity over every partition of the network, which is also a useful baseline for algorithm testing.

When testing algorithms in terms of modularity value of the partitions returned, we analyze the results both comparatively between the algorithms and in some cases, against known lower bounds for modularity, when such approximations are known. We run those tests on both real-world as well as randomized datasets.

Comparing the algorithms' results against the ground truth community structures of networks, we utilize Rand index (RI), which is a measure of similarity between datasets. The use of Rand index to evaluate similarity between two partitions is referenced by Chen et al. [3]. Rand index between two partitions P and Q of set S into communities defines the fraction of pairs of elements of S that are in the same communities in both P and Q or in different communities in both P and Q, among all the pairs of elements of S. Thus, it can be expressed as:

$$R(P,Q) = \frac{1}{n(n-1)} \sum_{i,j} \delta(\delta(p_i, p_j), \delta(q_i, q_j)) \,, \tag{3.1}$$

where i, j are elements of S, $n = |S|$, p_i, p_j are communities of i and j in P and q_i, q_j are communities of i and j in Q. It can be noted that higher values of Rand index reflect greater similarity between two partitions. We will use Rand index to compare the outputs of algorithms against the known community structure of some networks, both real-world ones and those generated in accordance with the synthetic community model.

Finally, we analyze the execution time of algorithms by running them on real-world networks as well as randomized graphs of different sizes.

In section 3.5 we also exemplify a well documented flaw of the modularity based community model, and consequently a flaw of community detection algorithms based on modularity optimization.This flaw known as resolution limit was first identified by Fortunato and Barthelemy [6], who observe that modularity maximization models fail to

detect communities smaller than a certain scale. Namely, communities with the number of inner edges fewer than $\sqrt{\frac{m}{2}}$, where m is the number of edges in the network, cannot be detected by modularity maximization methods, even if they are complete graphs connected to the rest of the network by only a single edge. We show that the algorithms tested by us fail to find the obvious community decompositions when communities are sufficiently small.

Lastly, in section 3.6 we analyze the statement by Blondel et al., that the execution time of their algorithm is heavily dependent on the ordering in which the vertices are processed throughout the first phase of the procedure and that the ordering does not significantly affect the modularity of partitions detected. To identify the correlations mentioned, we utilize standard deviation as well as a measure known as coefficient of variation (also relative standard deviation), which is the ratio of standard deviation of a parameter to its mean. We run the algorithm several times on the same random network generated in accordance with the social network model, each time randomizing the ordering of vertices as they are processed and apply coefficient of variation to the execution time and standard deviation to the modularity value of the partitions returned by the algorithm. We also test a heuristic of ordering the vertices by degree and compare the results of the procedure with such vertex ordering to the mean, maximum and minimum values of execution time resulting from running the algorithm with randomized vertex ordering.

3.2 Real-world datasets

3.2.1 Zachary's karate club

The first network whose community structure we analyze is a network of a university karate club studied by Zachary [24]. The 34 nodes of the network correspond to members of the club, while the 78 edges each represent an interaction between two members that took place outside of the club. At some stage, while the network was studied, a conflict between members labeled as 1 and 34 led to a division of the group into two separate clubs. We illustrate network and its division below:

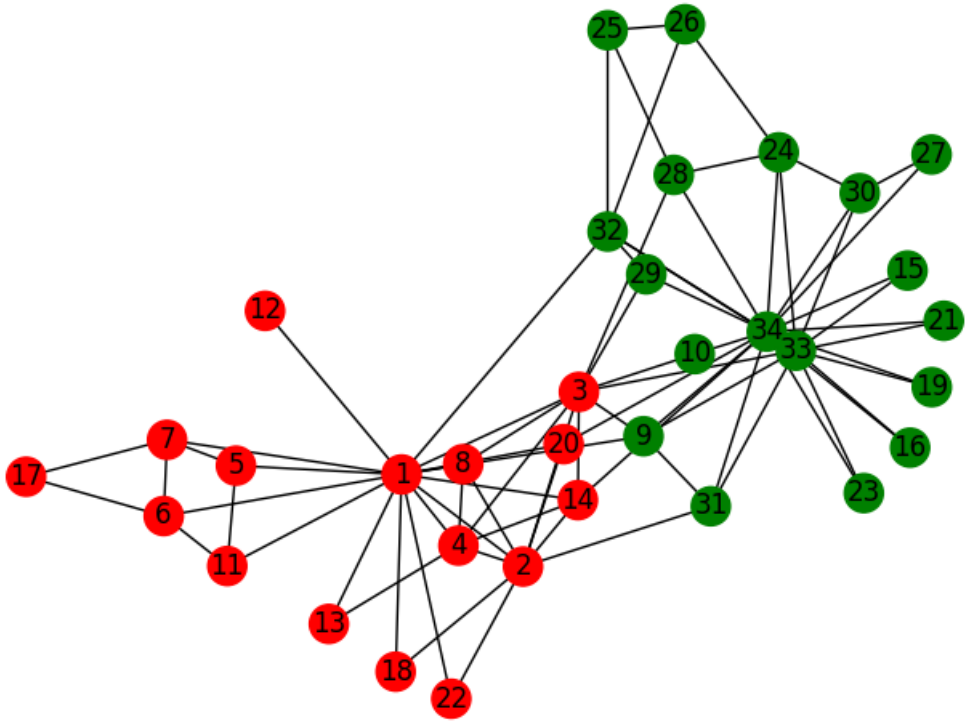
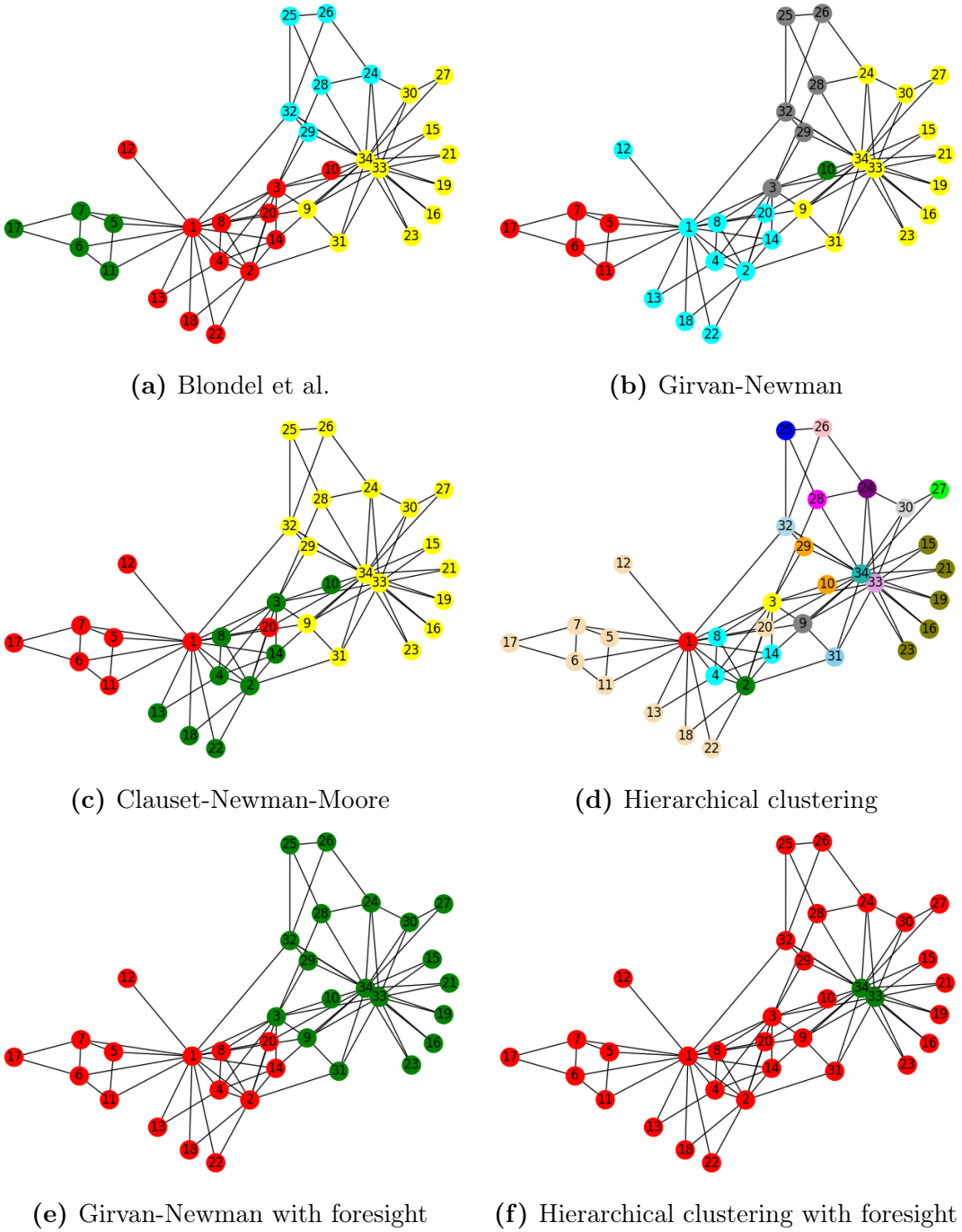


Figure 3.1: Actual community decomposition

When run on this network, the algorithms described in section 2 returned the following partitions:



(a) Blondel et al.

(b) Girvan-Newman

(c) Clauset-Newman-Moore

(d) Hierarchical clustering

(e) Girvan-Newman with foresight

(f) Hierarchical clustering with foresight

Figure 3.2: Community decomposition by various algorithms

The algorithms Girvan-Newman with foresight and hierarchical clustering with foresight are the implementations of respective algorithms, that instead of selecting the highest modularity partition, choose the partition that consists of exactly two communities, which is the number of communities in the actual partition. We can note that the community

decomposition returned by Girvan-Newman algorithm with foresight is almost identical to the actual decomposition, barring one misassigned vertex.

3.2.2 Dolphin social network

The network of bottlenose dolphins was assembled and studied by Lusseau et al. [13]. The particular group of dolphins exhibited exceptionally strong and long lasting connections which are reflected by edges of the network. There are 62 nodes and 159 edges in total and the ground truth community structure is not available.

3.2.3 American College football

The network, assembled by Girvan and Newman [8], reflects the schedule of American College football league in year 2000. Nodes correspond to teams while an edge exists between two vertices if the teams they represent played a game together during the season. Importantly, teams are separated into conferences based on geographical proximity and games between teams in the same conference are more frequent that games between those from different divisions. Thus, a ground truth community partition corresponds to the conference structure of the league. There are 115 nodes, 613 edges and 12 ground truth communities in the network.

3.2.4 Network science collaborations

Compiled by Mark Newman [18], this network of 1589 nodes and 2742 edges reflects coauthorships between scientists working in the area of network studies. The ground truth community structure is not provided.

3.2.5 Amazon co-purchase network

In this network, studied by Leskovec and Yang [22], vertices denote products sold on Amazon website, while an edge between two nodes exists if the two products were frequently purchased together. The network contains 334863 nodes, 925872 edges and 75149 ground

truth communities corresponding to product categories provided by Amazon. While ground truth communities are available for this network, they are not disjoint, as is the case with Amazon product categories and thus we shall not compare them with outputs of the algorithms tested, as they are all designed to find disjoint community decompositions.

3.2.6 Email network

Analyzed by Leskovec et al. [23], this dataset represents anonymized data about email correspondence between members of a certain European research institution. Each directed edge in the network corresponds to an email sent between a pair of members represented by nodes. We have removed self-loops and edge direction from the graph to make it suitable for testing. The modified graph contains 1005 nodes, 16064 edges and 42 ground truth communities, each corresponding to a department within the research institution, with a node contained in a community if the member represented by it belonged to the respective department.

3.2.7 Cumulative results

In this section, we present the results of tests run on datasets described above.

	Blo	CNM	GN	Hc	**Ground truth**
Karate	0.42	0.38	0.40	0.02	**0.37**
Dolphins	0.52	0.5	0.52	0.07	**N/A**
Football	0.6	0.57	0.6	0.59	**0.55**
Network science	0.96	0.96	0.58	0.96	**N/A**
Amazon	0.93	N/A[a]	N/A[b]	N/A[c]	**N/A**
Email	0.4	0.35	0.09	0.02	**0.29**

Table 3.1: An overview of modularity values returned by Blondel et al. (Blo), Girvan-Newman (GN), Clauset-Newman-Moore (CNM) and Hierarchical clustering (Hc) algorithms, compared against the modularity values of the ground truth community partitions. Values (a) and (c) could not be computed due to excessive memory requirements mentioned further in the chapter. Value (b) could not be computed within 6 hours of running the algorithm.

	Blo	CNM	GN	Hc
Karate	0s	0s	0s	0s
Dolphins	0s	0s	0s	0s
Football	0s	0s	0s	0s
Network science	0s	0s	22s	168s
Amazon	40s	N/A[d]	>6h	N/A[e]
Email	0s	4s	5.97h	44s

Table 3.2: Time it took each algorithm to execute on respective datasets. Values (d) and (e) could not be computed due to excessive memory requirements.

	Blo	CNM	GN	Hc
Karate	0.73	0.84	0.62	0.74
Football	0.97	0.87	0.96	0.96
Email	0.84	0.75	0.92	0.4

Table 3.3: Rand index of partitions returned by the algorithms, calculated against the ground truth community partitions for networks, which have a known ground truth structure.

As we can see, on real world datasets Blondel et al. algorithm achieved the best results both in terms of modularity value and execution time. However, there is no algorithm that performed significantly better than the other ones in terms of maximizing Rand index. Interestingly, the highest modularity partition was also the highest Rand index partition in only one of the three cases when similarity was measured. Also, the modularity values of ground truth community partitions were always lower than those found by Blondel et al. algorithm, which, however, should not be too worrying, as no model is expected to correspond perfectly to the real-world data. We can also note the execution time of Girvan-Newman algorithm, particularly on the "network science collaboration" network. Because of a very distinct community structure of the network (modularity value of at least 0.96), the algorithm executed much faster than hierarchical clustering algorithm, which has a better time complexity. This can be explained by the observation made by the authors (and mentioned already in section 2.2), that Girvan-Newman algorithm performs much faster on sparse networks with clearly identifiable communities, as it quickly splits such networks into smaller ones, which limits the required computations. The algorithm's execution time on "Email network" dataset is perhaps a better example of it's performance on denser networks with less profound community structure. We can also note that hierarchical clustering and Clauset-Newman-Moore algorithm could not

finish computation on "Amazon co-purchase network" dataset due to memory constraints. While hierarchical clustering uses $O(n^2)$ memory, which explains its failure to perform, all the other algorithms require $O(m+n)$ memory. Apparently the higher constant was the reason why Clauset-Newman-Moore algorithm could not execute fully, unlike the algorithms by Blondel et al. and Girvan-Newman.

3.3 Synthetic datasets

3.3.1 Synthetic community model

It is straightforward to verify that, when generating a network in accordance with synthetic community model, the expected number of edges can be expressed as:

$$\mathbb{E}(X) = c\binom{k}{2}(p_{in} + cp_{out}), \tag{3.2}$$

where c is the number of communities in the network, k is the size of each community (for simplicity we consider all communities to be equally sized, as do Blondel et al. [1], Girvan-Newman [8] and Lancichinetti et al. [10]), p_{in} is the probability of an edge being added between two vertices in the same community and p_{out} is the probability of adding an edge between two nodes in different communities. Thus, to maintain a constant expected value of the number of edges when modifying the values of p_{in} and p_{out}, for each increase in p_{in} by x we have to decrease p_{out} by $\frac{x}{c}$. Below, we illustrate various data as functions of p_{in} (and equivalently p_{out}), for constant values of k, c and $\mathbb{E}(X)$. The exact parameters of the model are $c = 10$ and $k = 12$, while p_{in} ranges over values 0.5 and 1 and p_{out} ranges over $\frac{1}{12}$ and $\frac{2}{15}$. Moreover, $p_{in} = \frac{22}{12} - 10p_{out}$, so that $\mathbb{E}(X)$ is constant and equal to 1210. The number of nodes is also constant and equal to ck, which is 120.

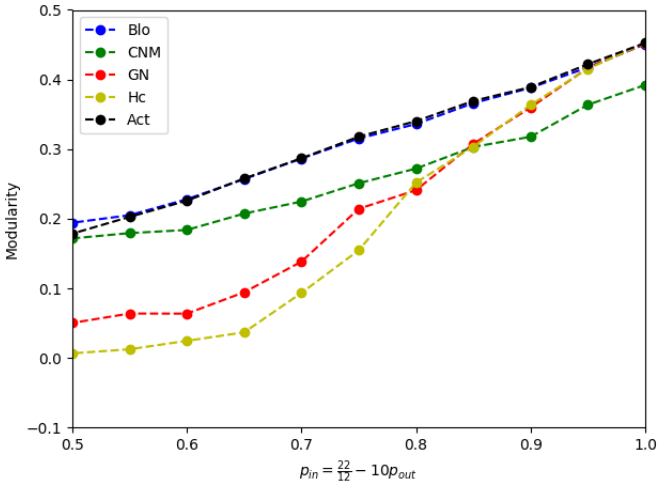


Figure 3.3: Modularity of partitions of the network as detected by Blondel et al. (Blo), Girvan-Newman (GN), Clauset-Newman-Moore (CNM) and Hierarchical clustering (Hc) algorithms, compared against modularity of the ground truth partition (Act). Each datapoint reflects an average over 10 such values calculated over 10 distinct randomized networks.

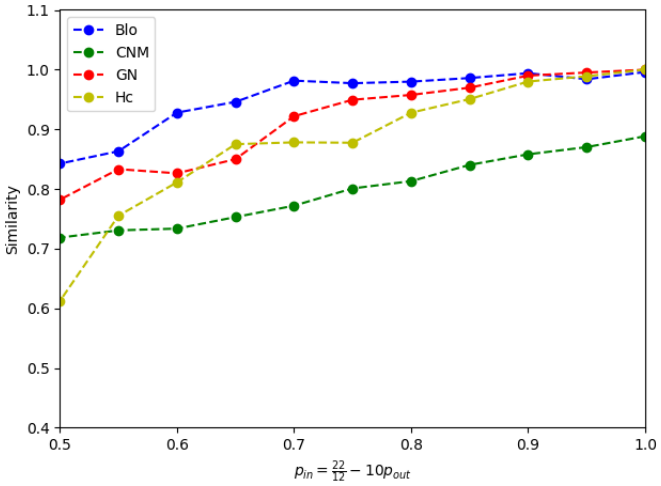


Figure 3.4: Rand index of partitions of the network as detected by the algorithms discussed, calculated against the ground truth partition. Each datapoint reflects an average over 10 such values calculated over 10 distinct randomized networks.

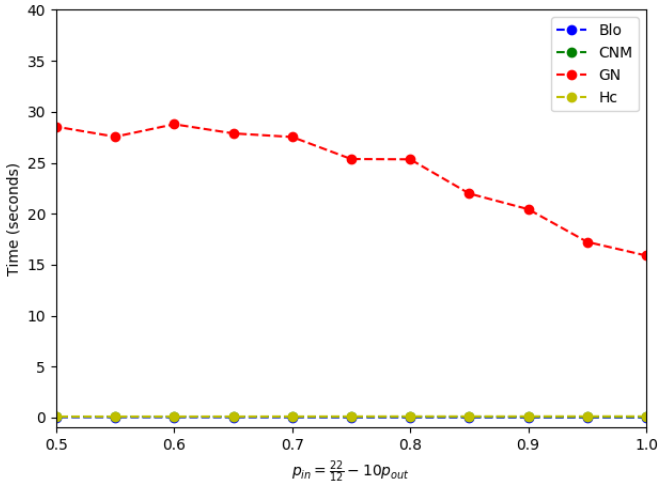


Figure 3.5: Time it took each of the algorithms discussed to produce the output partition. Each datapoint reflects an average over 10 such values calculated over 10 distinct randomized networks.

In terms of modularity maximization Blondel et al. algorithm once again achieves the best results and is the only algorithm that found partitions of higher modularity value than the ground truth partitions suggested by the model. Clauset-Newman-Moore algorithm also returns community decompositions of high modularity values but, compared to the algorithms of Girvan-Newman and hierarchical clustering, seems to do quite poorly on networks with more prominent community structure (higher values of p_{in}).

When it comes to Rand index, the algorithm by Blondel et al. also performs the best, with Clauset-Newman-Moore achieving the worst results for all but one data point.

As far as computation time is concerned, it is interesting to observe how the execution time of Girvan-Newman algorithm decreases as community structure becomes more distinct. This corresponds to the authors' observation mentioned in section 2.2 and also exemplified in section 3.2.7.

3.3.2 Social network model

Using the social network model, we test the execution time of the algorithms and modularity of partitions they detect. We generate the networks using the following probability distributions:

$$p(X = x) = \frac{1}{3}, \forall x \in \{1, 2, 3\}, \tag{3.3}$$

$$p(Y = y) = \begin{cases} 0.95 & \text{if } y = 1 \\ 0.05 & \text{if } y = 2 \end{cases}, \tag{3.4}$$

and a simple path of 5 vertices as a seed network, where X in the formula is a random variable reflecting the number of primary contacts and Y is a random variable defining the number of secondary contacts. We end the network generation when it reaches a specified number of vertices. The probability distributions are the same as suggested by the authors of the model in [21]. Importantly, the probability distributions put a somewhat unrealistic constraint on the size of the largest clique in the network being at most 6. While the authors also make that observation, they do not comment on it. They do, however, prove in their paper that with those probability distributions, the networks generated in accordance with the model have all the desirable properties mentioned in section 1.3.

In a perfect scenario, when each vertex is assigned as many neighbors as the probability distribution suggests, the expected value of a random variable Z denoting the number of edges in a network of n vertices generated as above can be thus defined as:

$$\mathbb{E}(Z) = 4 + n(\mathbb{E}(X) + \mathbb{E}(X)\mathbb{E}(Y)) = 4 + 4.1n\,. \tag{3.5}$$

However, this disregards the saturation effect mentioned in section 3.1. The mean of the actual number of edges is in fact $\sim 3.5n$ across all test networks below.

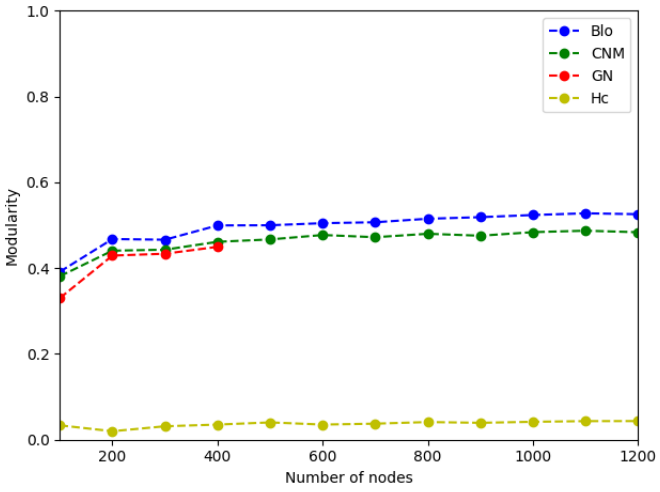


Figure 3.6: Modularity of partitions found by the algorithms.

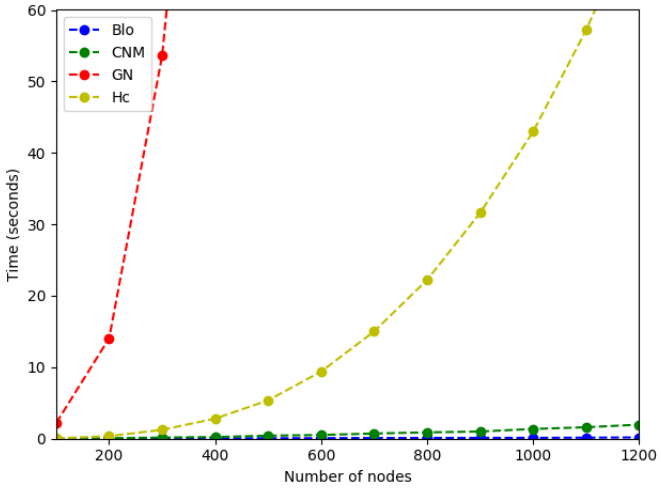


Figure 3.7: Execution time (in seconds) of the algorithms (small networks).

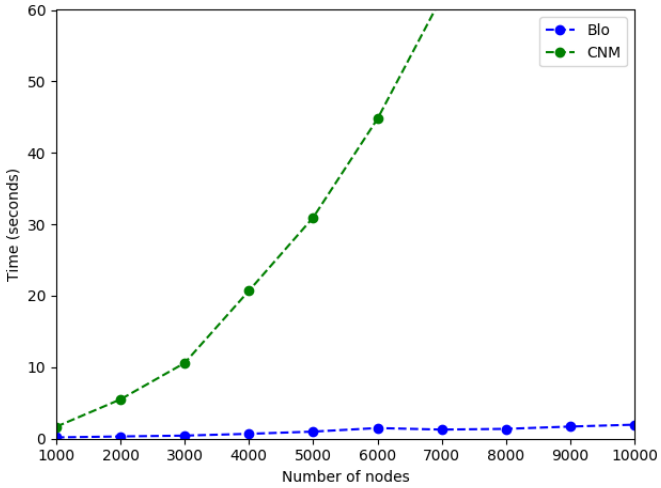


Figure 3.8: Execution time (in seconds) of the algorithms (medium networks).

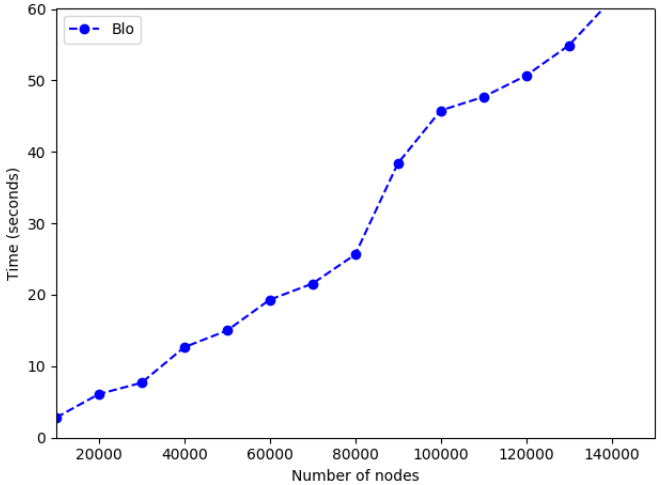


Figure 3.9: Execution time (in seconds) of the algorithms (large networks).

The test results support the observations already made in previous sections. Blondel et al. algorithm is by far the fastest and the best in terms of modularity optimization, while Clauset-Newman-Moore is second best in terms of both the parameters.

3.4 Tests in other publications

There have been a few attempts to comparatively analyze different community detection algorithms. However, only some of them used similar methodology and algorithm selection as us, and only those can be reasonably compared with our results.

Blondel et al. [1] compare performance of their algorithm with that by Clauset-Newman-Moore on a few real-world datasets, including "Zachary's karate club" network. The test on this dataset produce the exact same modularity results as our tests for both the algorithms. The superiority of Blondel et al. algorithm is also observed, both in terms of execution times and modularity values of partitions found on all the datasets, which is in line with our observations. Blondel et al. also note the susceptibility of their algorithm to resolution limit, which we register as well, in section 3.5.

Lancichinetti et al. [10] compare performance of various community detection algorithms including those by Blondel et al., Clauset-Newman-Moore and Girvan-Newman. They run the algorithms on networks generated in accordance with synthetic community model (which they refer to as "Girvan-Newman model"), with 4 communities, each of size 32 totaling 128 nodes, and varying values of p_{in} and p_{out}, such that the expected number of edges is always equal to 1024. Unlike ours, their results list Girvan-Newman algorithm as the worst of the three in terms of the similarity between the ground truth partitions and partitions found by the algorithm. They also note that Clauset-Newman-Moore algorithm does even better than Blondel et al. algorithm, when community structure of a network is barely observable ($p_{in} \sim p_{out}$). However, as a measure of similarity they use normalized mutual information instead of Rand index, which, alongside different model parameters may explain the discrepancies between our results and theirs.

3.5 Resolution limit

To exemplify the existence of resolution limit and the algorithms' susceptibility to it, we test them on a network of 150 degrees and 330 edges consisting of a ring of 30 cliques of 5 vertices each, connected by singular edges. Such network, proposed by Fortunato and Barthélemy [6] possesses an intuitively obvious community structure with each clique corresponding to a single community. However, as we show below, this community decomposition is not at all reflected by the outputs of modularity maximization algorithms.

	Blo	CNM	GN	Hc	**Ground truth**
Modularity	0.89	0.89	0.89	0.88	**0.88**
Rand index	0.97	0.97	0.97	1	**1**
Number of communities	15	17	16	29	**30**
Average community size	10	8.82	9.38	5.17	**5**

Table 3.4: Various parameters output by the algorithms tested on the example network. Rand index is calculated against ground truth community structure.

We can observe that the number of communities in partitions detected by three out of four algorithms differs significantly from the actual decomposition. Due to resolution limit, the communities found by these algorithms were unions of actual communities. In case of Blondel et al. algorithm, for instance, each community returned corresponded to a union of two cliques of size five. It is, however, important that the modularity values for partitions found by the algorithms were actually higher than that of the ground truth decomposition, which clearly identifies the problem to be with modeling community decomposition by modularity and not with the algorithms themselves.

Interestingly, hierarchical clustering produced a network partition nearly identical to the ground truth one, except for two cliques of size five merged into a single community. This good result can be explained by the fact that hierarchical clustering is not primarily a modularity maximization method but only utilizes modularity to determine the best partition from a number of available ones, which were generated otherwise.

3.6 Vertex ordering in Blondel et al. algorithm

The graphs used for testing the dependency between vertex ordering in Blondel algorithm and its execution time and modularity of partitions returned are generated in accordance with social network model using the same probability distributions and seed network as defined in section 3.3.2. The variant of standard deviation we use in our tests is the sample standard deviation (SD) defined as:

$$s = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N - 1}}, \tag{3.6}$$

while the coefficient of variation (CV) is expressed as:

$$c = \frac{s}{\bar{x}}, \tag{3.7}$$

where x_i are observed values, N is the number of samples and $\bar{x}$ is the mean over all x_i.

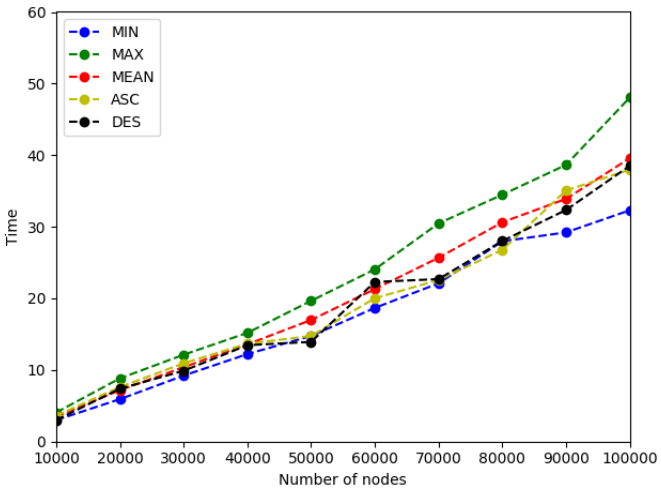


Figure 3.10: Minimum (MIN), maximum (MAX) and mean (MEAN) execution times (in seconds) of Blondel et al. algoirthm over 20 trials with randomized vertex ordering. Execution times with vertices ordered by ascending (ASC) and descending (DES) degrees added for comparison.

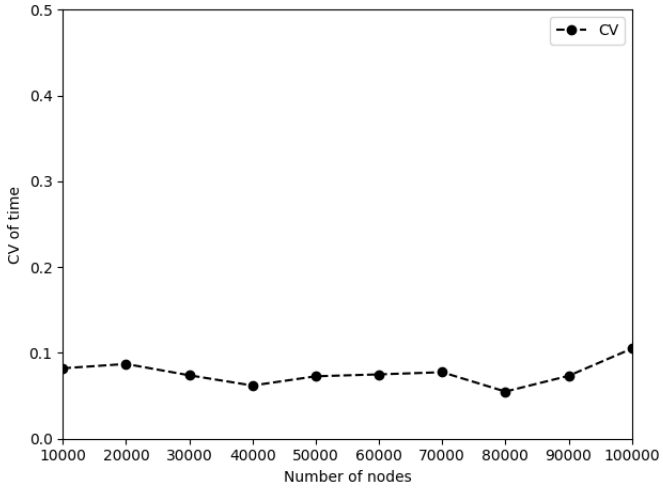


Figure 3.11: Coefficient of variation of execution time of Blondel et al. algorithm over 20 trials with randomized vertex ordering.

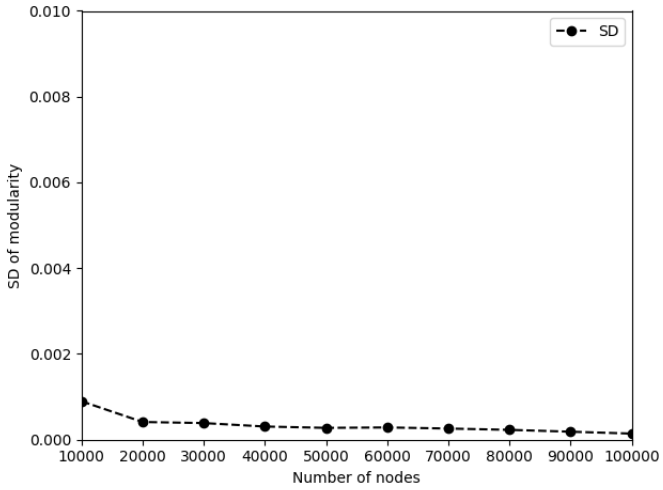


Figure 3.12: Standard deviation of modularity of community decompositions returned by Blondel et al. algorithm over 20 trials with randomized vertex ordering.

Our observations based on the tests above support those made by Blondel et al. Firstly, the standard deviation of modularity with regard to different randomized vertex orderings is negligible, standing at approximately 0.005 and varying slightly depending on sizes of the networks. The coefficient of variation of execution time being less than 0.1 indicates that standard deviation is less than 10% of the mean value of execution time. This might not be a lot, however, looking at figure 3.10 we can observe that the maximum execution time can be as high as 150% of minimum execution time, which indicates that with an optimal vertex ordering we could execute Blondel et al. algorithm in no more than $\frac{2}{3}$ of the pessimistic execution time. Considering this fact, it is perhaps worth it to study

the vertex orderings further. We also observe that the simple heuristics proposed by us, of ordering the vertices by ascending or descending degree both result, in most cases, in slightly faster computation times than the mean. The difference however is not too meaningful but perhaps the idea is worth further investigation.

4 Conclusions

In this paper we discuss common properties of networks and highlight the validity and importance of community detection. We introduce the modularity based definition of a community and describe in detail four heuristic community detection algorithms based on modularity optimization. Finally, we test the algorithms on both randomly generated and real-world datasets. As a result of those tests we conclude that, out of the algorithms analyzed, the one by Blondel et al. seems to have the most applications, being by far the fastest and performing the best in terms of modularity values of the partitions produced. It can be used with great effect to generate community decompositions of networks with up to a million edges in less than a minute. We also observe that the performance of hierarchical clustering algorithm is rather underwhelming on most networks, which is exemplified most profoundly in figure 3.6.

We also showcase several properties of community detection algorithms: Girvan-Newman algorithm's execution time dependency on the distinctiveness of network's community structure and the moderately high variance of Blondel et al. algorithm's execution time with regards to the ordering of network's vertices.

Lastly, we exemplify some limitations of the modularity based community model. Most importantly, we illustrate the existence of so called resolution limit of modularity and present instances when it leads to detection of communities which in no way correspond to the intuitively correct community structure. A lesser problem is the fact that the modularity value of ground truth community structure is often significantly lower than the values found by modularity optimization algorithms. While this undermines the validity of defining community partition as the maximum modularity partition, it is also expected for a model to not be a perfect representation of reality. However, this discrepancy suggests that there might be a better way of modeling communities than with the use of modularity.

There are many directions in which community detection could be developed. One could attempt to create a benefit function alternative to modularity, which would be more resilient to resolution limit or one which would perhaps lead to a more fitting model of a community. In fact, some such functions have already been proposed, for instance,

modularity density suggested by Li et al. [12] and defined as:

$$D = \sum_{c \in C} \frac{2|E_c^{in}| - |E_c^{out}|}{|c|}, \tag{4.1}$$

where C is the set of all communities in the network, E_c^{in} is the set of edges with both ends in nodes in community c and E_c^{out} is the set of edges with exactly one end in a node in community c. Modularity density maximization heuristics have also been suggested, for example by Chen et al. [3]. As tested in their paper, the algorithm by Chen et al. achieves, in most cases, higher values of Rand index than Clauset-Newman-Moore algorithm, however it is not mentioned how it compares to, for instance, Blondel et al. algorithm. It does also seem to be insusceptible to resolution limit, finding the intuitively correct community decomposition of the clique network mentioned in section 3.5.

One could also look at creating a new, possibly faster algorithm for modularity optimization or perhaps improve an existing one. In that regard, the algorithm by Blondel et al. is particularly appealing, as not only is it the best performing of the algorithms we have tested but, as we have shown, it could also be quite significantly improved with a vertex ordering heuristic.

References

[1] 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):10008.

[2] Brandes, U. (2001). A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177.

[3] Chen, M., Kuzmin, K., and Szymanski, B. K. (2014). Community detection via maximization of modularity and its variants. *IEEE Transactions on Computational Social Systems*, 1(1):46–65.

[4] Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111.

[5] Estrada, E. (2013). Graph and Network Theory in Physics. *arXiv e-prints*, page arXiv:1302.4378.

[6] Fortunato, S. and Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the National Academy of Science*, 104(1):36–41.

[7] Garcia, J. O., Ashourvan, A., Muldoon, S., Vettel, J. M., and Bassett, D. S. (2018). Applications of community detection techniques to brain graphs: Algorithmic considerations and implications for neural function. *Proceedings of the IEEE*, 106(5):846–867.

[8] Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826.

[9] Karataş, A. and Şahin, S. (2018). Application areas of community detection: A review. pages 65–70.

[10] Lancichinetti, A. and Fortunato, S. (2009). Community detection algorithms: A comparative analysis. *Phys. Rev. E*, 80(5):056117.

[11] Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[12] Li, Z., Zhang, S., Wang, R.-S., Zhang, X.-S., and Chen, L. (2008). Quantitative function for community detection. *Phys. Rev. E*, 77:036109.

[13] Lusseau, D., Schneider, K., Boisseau, O., Haase, P., Slooten, E., and Dawson, S. (2003). The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations - can geographic isolation explain this unique trait? *Behavioral Ecology and Sociobiology*, 54:396–405.

[14] Newman, M. (2013). Network data. http://www-personal.umich.edu/~mejn/netdata/. Accessed: 2019-06-10.

[15] Newman, M. E. (2004). Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6):066133.

[16] Newman, M. E. J. (2003). The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256.

[17] Newman, M. E. J. (2004). Detecting community structure in networks. *The European Physical Journal B*, 38(2):321–330.

[18] Newman, M. E. J. (2006). Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104.

[19] Newman, M. E. J. (2010). *Networks: an introduction*. Oxford University Press, Oxford; New York.

[20] Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113.

[21] Toivonen, R., Onnela, J.-P., Saramäki, J., Hyvönen, J., and Kaski, K. (2006). A model for social networks. *Physica A Statistical Mechanics and its Applications*, 371(2):851–860.

[22] Yang, J. and Leskovec, J. (2012). Defining and evaluating network communities based on ground-truth. In *2012 IEEE 12th International Conference on Data Mining*, pages 745–754.

[23] Yin, H., Benson, A. R., Leskovec, J., and Gleich, D. F. (2017). Local higher-order graph clustering. *KDD : proceedings. International Conference on Knowledge Discovery and Data Mining*, 2017:555–564.

[24] Zachary, W. (1976). An information flow model for conflict and fission in small groups1. *Journal of anthropological research*, 33.