Efficient Anti-community Detection in Complex Networks

Sebastian Lackner
Heidelberg University
Heidelberg, Germany
lackner@informatik.uni-heidelberg.de

Andreas Spitz
Heidelberg University
Heidelberg, Germany
spitz@informatik.uni-heidelberg.de

Matthias Weidemüller
Heidelberg University
Heidelberg, Germany
weidemueller@uni-heidelberg.de

Michael Gertz
Heidelberg University
Heidelberg, Germany
gertz@informatik.uni-heidelberg.de

ABSTRACT

Modeling the relations between the components of complex systems as networks of vertices and edges is a commonly used method in many scientific disciplines that serves to obtain a deeper understanding of the systems themselves. In particular, the detection of densely connected communities in these networks is frequently used to identify functionally related components, such as social circles in networks of personal relations or interactions between agents in biological networks. Traditionally, communities are considered to have a high density of internal connections, combined with a low density of external edges between different communities. However, not all naturally occurring communities in complex networks are characterized by this notion of structural equivalence, such as groups of energy states with shared quantum numbers in networks of spectral line transitions. In this paper, we focus on this inverse task of detecting *anti-communities* that are characterized by an exceptionally low density of internal connections and a high density of external connections. While anti-communities have been discussed in the literature for anecdotal applications or as a modification of traditional community detection, no rigorous investigation of algorithms for the problem has been presented. To this end, we introduce and discuss a broad range of possible approaches and evaluate them with regard to efficiency and effectiveness on a range of real-world and synthetic networks. Furthermore, we show that the presence of a community and anti-community structure are not mutually exclusive, and that even networks with a strong traditional community structure may also contain anti-communities.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; • **Applied computing →** *Physics*;

KEYWORDS

community detection; graph algorithms; hierarchical clustering

SSDBM '18, July 9–11, 2018, Bozen-Bolzano, Italy

ACM Reference Format:
Sebastian Lackner, Andreas Spitz, Matthias Weidemüller, and Michael Gertz. 2018. Efficient Anti-community Detection in Complex Networks. In *SSDBM '18: 30th International Conference on Scientific and Statistical Database Management, July 9–11, 2018, Bozen-Bolzano, Italy*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3221269.3221289

1 INTRODUCTION

The expression *birds of a feather flock together* aptly describes the intuition behind clustering and community detection approaches in complex networks of interconnected vertices, which strive to group the vertices of a network based on their shared characteristics. In the case of social networks, such characteristics might be shared interests, hobbies, or political convictions, and can be modeled as vertex attributes. In many application scenarios, however, such vertex labels are not available, meaning that the grouping of vertices has to rely solely on the connectivity structure of the network. This concept, which is known as *structural equivalence*, describes the observation that similar vertices have similar neighbors, and thus indicates that the connectivity structure within observed communities or clusters tends to be very dense. On an intuitive level, such an interpretation of communities as clique-like substructures of the network is meaningful in many common types of networks that are as diverse as (online) social networks, citation and authorship networks, or biological networks. It is thus not surprising that *community detection* in large complex networks has received a lot of attention from researchers in seemingly unrelated scientific fields in the past. The historically most established approach is the clustering of vertices into communities based on the so-called *modularity* score [10, 20], which maximizes the number of *internal* connections between vertices within a group, while simultaneously minimizing the number of connections between vertices of different groups.

In contrast to these networks with predominating traditional community structures, some networks contain communities with diametrically opposed properties. As a trivial, bipartite example, consider a network of customer reviews for products, which contains no connections between products or between reviewers, and can be partitioned into two sets of vertices that correspond to the reviewers and products. More complex examples include adjacency networks of words in natural language processing that only rarely include connections between words with identical part-of-speech tags, or networks of spectral line transitions in spectroscopy that seldomly contain transitions between states suppressed by selection rules (e.g., states with the same azimuthal quantum number for

electric dipole transitions). For such networks, reasonable vertex partitions have a low number of internal edges but a high number of external edges, and thus minimize (instead of maximize) the modularity score. In the following, we focus on these *anti-communities*.

In contrast to the well-researched topic of traditional community detection, research into anti-community detection has so far been limited to single algorithms with limited scalability and to specific applications such as protein-protein interactions [7] or conflicts in traditional Chinese medicine [35]. Intuitively, the task of finding anti-community structures in a network is closely related to the task of finding communities in the graph complement (i.e., the network that is obtained by removing all existing edges and including all missing edges). For the problem of graph partitioning, it can also be shown mathematically that both tasks are equivalent [34], which means that algorithms for community detection can in principle be applied to the detection of anti-community structures. For an illustration of this transformation, see the example in Figure 1. However, while such a transformation is possible in theory, it is ill advised in practice since the number of edges is typically a factor in the runtime complexity of community detection algorithms. For sparse input networks with n vertices and $O(n)$ edges, the graph complement is dense and has $O(n^2)$ edges. While an increased memory overhead can likely be avoided by compression or by computing the graph complement dynamically, the increased number of edges still raises the runtime complexity of algorithms that rely on the sparsity of the network to achieve their efficiency, such as the well-established agglomerative hierarchical clustering [18]. Since most community detection algorithms already have a loglinear or quadratic runtime, an additional factor of n would make them prohibitively expensive for use on large networks. Thus, even though numerous algorithms exist for community detection, the majority cannot be used to solve the inverse problem efficiently.

In this paper, we give an overview of existing methods for anti-community detection and propose a number of novel methods, which we evaluate on real-world networks and synthetic data sets for runtime efficiency and anti-community detection effectiveness. In summary, our contributions in this work are:

- **Anti-community detection methods.** We propose greedy modularity minimization (GrM) and anti-modularity maximization (GrAM) methods, as well as two vertex similarity based methods (VSA, VSD) for anti-community detection.[1]
- **Anti-community random network models.** We adapt the *Erdős-Rényi* and *Barabási-Albert* random graph models to facilitate the generation of synthetic graphs with variable community or anti-community structure.
- **Novel performance measures.** We adapt the *adjusted Rand index* and the *normalized mutual information* for the evaluation of anti-community detection in networks with multiple connected components.
- **Comparative evaluation.** We evaluate the performance and runtime of our proposed algorithms and multiple algorithms from the literature on synthetic and real data.
- **Applications.** We investigate the anti-community structure of several real-world networks, including networks of spectral line transitions from the field of Physics.

[1] Our source code is available at https://github.com/slackner/anti-community.

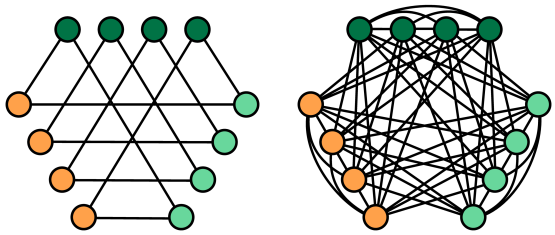

Figure 1: Example graph with 12 edges and three anti-communities (denoted by colors) and the graph complement with the corresponding three communities and 54 edges.

The remainder of this paper is structured as follows. In Section 2, we discuss community and anti-community detection methods in the literature. In Section 3, we introduce our proposed algorithms for detecting anti-communities, which we evaluate together with the baseline algorithms in Section 4. Finally, in Section 5, we explore the anti-community structures of real networks.

2 RELATED WORK

Although the concept of traditional communities as groups of vertices with a high density of edges within each group and a small density of edges between different groups is intuitive, there is no unique mathematical definition of community in the literature [8]. Thus, the output of community detection methods depends on the used concept or algorithm. In the past, various definitions and methods have been proposed, of which we discuss the most related approaches in the following before giving an overview over the few existing approaches to anti-community detection.

Community detection. Community detection is closely related to *graph partitioning*, which refers to the problem of dividing a graph into a specific number of disjoint groups of predefined sizes, such that the *cut size* (i.e., the number of edges between groups) is minimized [2, 12, 29]. The problem is especially relevant for parallel computing and for designing circuit layouts [12]. In principle, graph partitioning methods can be used for the purpose of community detection but usually require that the number of groups is known in advance, which is rarely the case for arbitrary complex networks.

On the other hand, methods that are based on the *modularity* measure do not suffer from this problem. Originally introduced by Newman and Girvan [20], the modularity has rapidly become one of the most popular approaches to community detection since it allows the evaluation of the quality of a partitioning in a specific network. The first algorithm for finding community structures based on maximizing modularity was a greedy method suggested by Newman [18]. Other methods rely on simulated annealing [11], genetic algorithms [14, 25, 28, 30], or spectral optimization [19].

Using modularity as a quality function entails that the algorithm is optimizing for dense substructures in the network. In contrast, other methods are more flexible and work for arbitrary connectivity patterns. This applies, for example, to algorithms that are based on the concept of *structural equivalence* [16], which postulates that two vertices are structurally equivalent if they share common neighbors. This idea can also be used to define similarity measures for vertices [3], or to group vertices that have similar link patterns [15]. Similarly, *stochastic block model* methods try to fit a generative

model to the network and also do not make assumptions about the underlying structure [21, 23, 24]. Finally, community detection is also closely related to clustering if vertices can be attributed with (higher-dimensional) features that are used as input. For example, hierarchical clustering algorithms can be applied to a matrix of vertex similarities to extract the corresponding groups [8].

Anti-community detection. The term *anti-community* was first used by Newman to describe the presence of approximately bipartite or multipartite structures in networks [19]. The same paper also introduces the idea of obtaining a (bipartite) clustering by minimizing modularity using spectral optimization. Intuitively, anti-community detection in a network is closely related to community detection in the graph complement and mathematically equivalent for the case of graph partitioning [34].

More recently, Chen et al. proposed a new *anti-modularity* measure for the purpose of evaluating anti-community structures and presented a fast label propagation algorithm. For evaluation purposes, they use various bipartite networks. Similarly, Zhu et al. presented the two heuristic methods REON and REONI, which effectively try to minimize modularity. Their algorithms are evaluated on both bipartite and non-bipartite networks, including a network of herbs containing information about the incompatibility of traditional Chinese medicine. Fasino et al. proposed a spectral method for simultaneous community and anti-community detection based on modularity minimization or maximization [7].

In addition to algorithms that are explicitly designed for anti-community detection, several algorithms for traditional community detection can be adapted to this purpose. These include methods based on vertex similarity and similar link patterns, as well as stochastic block models, which only make generic assumptions about the structure present in the graph. In fact, by first computing the graph complement, any community detection algorithm can be used for this purpose and can thus be seen as a baseline.

Overall, there are only few and specialized algorithms for anti-community detection in the literature. Computing the graph complement is infeasible for large sparse graphs because the complement is dense, which leads to increased storage and runtime requirements. While the label propagation algorithm is indeed very fast, it only performs well for networks with a well defined structure, as we show in the performance evaluation. Even for vertex similarity methods, it has not yet been investigated which similarity measures are suitable for detecting anti-communities. In the following section, we address these problems and present specialized algorithms for anti-community detection.

3 MODELS AND ALGORITHMS

In this section we discuss the proposed algorithms and the underlying models and measures. We begin with the *modularity* and *anti-modularity* measures that provide a basis for our greedy algorithms, before we discuss the various baseline algorithms. Finally, in Section 3.3 and Section 3.4, we introduce our novel algorithms.

Graph notation. In the following, we formalize networks as graphs $\mathcal{G} = (V, E)$ that connect a set of vertices V by a set of edges $E \subseteq V \times V$. With $\mathbf{A} = [a_{ij}]$, we denote the adjacency matrix of such a graph. We say that $\mathcal{G}$ has $m = |E|$ edges and $n = |V|$ vertices. With $d_i := \sum_{j=1}^{n} a_{ij}$, we denote the degree of vertex v_i.

3.1 Modularity and Anti-modularity

In order to distinguish between good and bad partitions of vertices, it is useful to define *quality functions* that assign real numbers to each partition of a graph. Since most real networks contain no unique true community structure, such quality functions can be used to compare the results of different algorithms. Furthermore, several algorithms for community detection directly make use of quality measures, which are used either as a stopping criterion or for iterative improvements until a local maximum is found.

Modularity. The most popular quality function for partitions is the *modularity* [20], which is based on the assumption that a random graph does not have any community structure. If the density of connections in a subgraph is higher than expected by pure chance, a community structure is present. Formally, the modularity of an undirected and unweighted graph $\mathcal{G} = (V, E)$ is defined as

$$M := \frac{1}{2m} \sum_{ij} \left[a_{ij} - \frac{d_i d_j}{2m} \right] \delta(g_i, g_j) \tag{1}$$

where $g = [g_i]$ denotes the group assignments of vertices such that the δ-function evaluates to one if $g_i = g_j$ and zero otherwise. The term $d_i d_j / 2m$ serves as a *null model* and contains the expected number of edges connecting v_i and v_j in a random graph with the same degree distribution. Note that the modularity (as well the algorithms that use it) can be extended to weighted graphs.

Anti-modularity. Similar to the modularity metric, efforts have been made to develop quality functions for the evaluation of anti-community structures. Chen et al. introduce an *anti-modularity* measure based on the observation that vertices in the same anti-community group are often connected by paths of length two [4].

$$M_A := \frac{1}{n} \sum_{ij} \left[\sum_{k=1}^{n} a_{ik} a_{kj} - \frac{d_i d_j}{n} \right] \delta(g_i, g_j). \tag{2}$$

The δ-function serves the same purpose as above. Up to a constant factor, the terms in the brackets are identical to the elements of the covariance matrix of column vectors of the adjacency matrix $\mathbf{A}$.

3.2 Baseline Algorithms

Due to the symmetry between communities and anti-communities, algorithms for community detection can also be used for anti-community detection by first computing the graph complement. Since numerous methods for traditional community detection exist, it is infeasible to cover all possible baseline algorithms. Thus, we focus on the most promising and well-known approaches.

Graph complement + Modularity (GCM). The *graph complement* $\hat{\mathcal{G}} = (V, \hat{E})$ of a graph $\mathcal{G} = (V, E)$ can be obtained by adding all edges that are not present in the original graph, and by removing those that are present. Mathematically, the set of edges of $\hat{\mathcal{G}}$ is given by $\hat{E} := (V \times V) \setminus E$. Using the graph complement as input, we apply the *agglomerative hierarchical clustering* method proposed by Newman [18] to detect communities in the complement that correspond to anti-communities in the original graph.

Label propagation algorithm (LP). As a second baseline, we use the *label propagation algorithm for anti-community detection* proposed by Chen et al. [4]. The algorithm starts by assigning each vertex to a separate anti-community, which is stored in a label vector $g = [g_i]$. As long as there are non-adjacent vertices v_i and

v_j, and none of the neighbors of v_i shares a label with v_j, the label vector is updated by assigning $g_i \leftarrow g_j$. The process is repeated until the label vector converges to the final result. We implemented the method based on the description in their paper.

Stochastic Block Model (SBM). The SBM method by Newman and Reinert allows to estimate the number of communities in graphs, but can also be used for anti-community detection [21]. The approach is based on the assumption that a graph can be approximated by a *block model* in which each vertex is assigned to one of k groups and edges between groups are distributed randomly. The probability of edges between different groups is given as a probability matrix $\omega = [\omega_{ij}]$ of size $k \times k$. The algorithm fits the block model to the data, but does not enforce a specific type of structure. It initializes with a random configuration (k, g) and computes the likelihood $P(k, g|\mathbf{A})$. A *Monte Carlo* process is then used to update the configuration. In each step, either the parameter k or the group assignment g is changed. When the configuration has converged, sampling g can be used to obtain the anti-communities.

Nested Stochastic Block Model (NSBM). The SBM suffers from underfitting, i.e., very small structures can be misinterpreted as randomness. Modularity-based measures are affected by a similar problem [9]. Peixoto proposed to solve this by using hierarchical block models [23]. Here, the idea is to interpret the entries of the probability matrix $\omega = [\omega_{ij}]$ as edge counts in a multigraph, which itself can be described by another block model. Recursive application of this process leads to a nested model with multiple levels. To find the optimal configuration for a given graph, the algorithm performs a local move at each step, e.g., choosing a new partition for a specific level, inserting new levels, or deleting levels. The best configuration can be obtained by sampling or maximizing the posterior probability [24]. In our evaluation we use sampling.

3.3 Greedy Algorithms

Using modularity or anti-modularity, anti-community detection can be reduced to an optimization problem. Finding the best partition is usually **NP**-hard [8], meaning that the time requirement for computing the exact solution likely grows at least exponentially with the input size. As a result, it is common practice to use heuristic approximation algorithms. Greedy algorithms are a special class of approximation algorithms that make a locally optimal choice in each step instead of searching for the global optimum.

Greedy algorithm with modularity (GrM). As pointed out by Newman, a large negative modularity can indicate the presence of a bipartite (or multipartite) structure [19]. This means that the anti-community structure of a graph $\mathcal{G}$ can be obtained by computing the partition that minimizes the modularity $M(g)$ among all possible partitions of V. Since computing the global optimum is infeasible for large graphs, we propose a greedy method that is similar to agglomerative hierarchical clustering [18]. Variations of this method are popular due to the low computational complexity. In order to find anti-communities, the modularity measure has to be minimized instead of maximized. Although this change seems trivial, it requires significant modifications to ensure the algorithm's efficiency. We discuss the algorithm's construction in the following. For a pseudo code description, see Algorithm 1 and Function FindMergeM.

The original algorithm starts with n clusters, each containing a single vertex. In each step, the algorithm loops through all edges $(v_i, v_j) \in E$ of the graph and computes the new modularity after merging the clusters containing v_i and v_j. This heuristic is motivated by the observation that merging two clusters that are not connected by an edge always decreases the modularity, so it is easier (but not always equivalent) to consider only m instead of $\binom{n}{2}$ possible merges. The merge that results in the largest modularity increase (or smallest modularity decrease) is then performed. If the value of the current partition is higher than the previously found best solution, the corresponding label vector is stored. The process is repeated until all edges are contained within a cluster. For a connected graph, the algorithm stops after $n - 1$ merges, and there is only a single cluster left. The algorithm then returns the partition with the largest modularity value found. The complexity of each step is $O(m + n)$, and the total time complexity is $O(n(m + n))$.

The concept of a greedy algorithm for modularity minimization is similar. As a starting point, n clusters containing a single vertex are initialized. However, we are now interested in merging clusters that are *not* connected by an edge to decrease the modularity. A naive enumeration would increase the total complexity to $O(n^3)$. Instead, we propose an algorithm that checks *all* $\binom{n}{2}$ possible merges, while keeping the complexity of the original method. For each merge $(v_i, v_j) \in V^2$, we have to compute the modularity change by merging the clusters containing v_i and v_j. The merge that leads to the largest modularity decrease is performed afterwards. The algorithm stops after at most $n - 1$ steps and returns the partition with the smallest modularity value found during all iterations.

To compute the modularity changes $\delta\mathbf{M} = [\delta m_{ij}]$ efficiently for each merge, we keep track of a state that is updated in each step. In the initial configuration, the modularity is given by

$$M = \frac{1}{2m}\left[\mathrm{Tr}(\mathbf{A}) - \sum_{i=1}^{n} \frac{d_i^2}{2m}\right]. \tag{3}$$

Algorithm 1: GrM($\mathbf{A}$): Greedy algorithm for *modularity* min.

Input: Adjacency matrix $\mathbf{A} = [a_{ij}]$ of size $n \times n$
Output: Vector of vertex labels g, modularity M

```
g ← empty vector of size n;
d ← empty vector of size n;
for i ← 1 to n do  g_i ← i;                          ▷ assign initial labels
for i ← 1 to n do  d_i = Σ_{j=1}^{n} a_ij;           ▷ compute vertex degrees
M ← 1/(2m) Σ_{i=1}^{n} (a_ii − d_i^2/2m);            ▷ compute modularity
s ← ARGSORT(d);                                      ▷ prepare index for sorted access
while i, j ← FindMergeM(A, d, s) do
    M ← M + 1/m (a_ij − d_i d_j/2m);                 ▷ update modularity
    for k ← 1 to n do  a_ik ← a_ik + a_jk;           ▷ add rows
    for k ← 1 to n do  a_ki ← a_ki + a_kj;           ▷ add columns
    d_i ← d_i + d_j;                                 ▷ update vertex degrees
    for k where g_k = j do  g_k ← i;                 ▷ update labels
    delete row/column j from A and d;
    reorder s to reflect updated vertex degrees;
return best partition of all iterations;
```

The change for merging clusters i and j is

$$\delta m_{ij} \coloneqq \frac{1}{2m}\left(a_{ij} + a_{ji} - \frac{d_i d_j}{m}\right) = \frac{1}{m}\left(a_{ij} - \frac{d_i d_j}{2m}\right), \tag{4}$$

which can be simplified for vertices not connected by an edge to

$$\delta m^*_{ij} \coloneqq -\frac{d_i d_j}{2m^2}. \tag{5}$$

When a suitable merge is found, both the matrix $\mathbf{A}$ and the vector of degrees d can be updated by adding the corresponding rows and columns to reflect the changed connectivity.

$$\begin{aligned} \forall k : a^{(\text{tmp})}_{ik} &\leftarrow a_{ik} + a_{jk} \\ \forall k : a^{(\text{new})}_{ki} &\leftarrow a^{(\text{tmp})}_{ki} + a^{(\text{tmp})}_{kj} \\ d^{(\text{new})}_i &\leftarrow d_i + d_j \end{aligned} \tag{6}$$

After the first update step, $\mathbf{A}$ may also contain self loops or values greater than one. The entries that correspond to row/column j are no longer needed and can be removed. Both the initial computation and the updates can be performed in $O(n)$.

The most significant difference to the original method is the search for the best modularity change δm_{ij}. The original algorithm loops over all edges $(v_i, v_j) \in E$, which requires $O(m)$ time. In this case, however, we are primarily interested in suitable merges $(v_i, v_j) \notin E$, i.e., groups that are not connected by any edges. The search for a suitable merge is split in two steps. First, we search for the best merge with $(v_i, v_j) \notin E$. Second, we iterate over all edges $(v_i, v_j) \in E$ in $O(m)$ and check if we can find an even better merge. The second step is trivial, so we focus on the first.

In order to solve this problem efficiently, we maintain a lookup table s that provides an index for accessing elements of the vertex degree vector d in sorted order, i.e., $d_{(s_1)} \geq \cdots \geq d_{(s_i)} \geq \cdots \geq d_{(s_n)}$. The final goal is to enumerate products $d_i d_j$ in sorted order, because the same term also appears in the formula for modularity changes δm^*_{ij} in Equation 5. Up to a constant factor, the sorted entries are:

$$\begin{bmatrix} d_{(s_1)}d_{(s_1)} & d_{(s_1)}d_{(s_2)} & \cdots & d_{(s_1)}d_{(s_{n-1})} & d_{(s_1)}d_{(s_n)} \\ d_{(s_2)}d_{(s_1)} & d_{(s_2)}d_{(s_2)} & \cdots & d_{(s_2)}d_{(s_{n-1})} & d_{(s_2)}d_{(s_n)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{(s_{n-1})}d_{(s_1)} & d_{(s_{n-1})}d_{(s_2)} & \cdots & d_{(s_{n-1})}d_{(s_{n-1})} & d_{(s_{n-1})}d_{(s_n)} \\ d_{(s_n)}d_{(s_1)} & d_{(s_n)}d_{(s_2)} & \cdots & d_{(s_n)}d_{(s_{n-1})} & d_{(s_n)}d_{(s_n)} \end{bmatrix}$$

Since this matrix is symmetric, it is sufficient to consider only the lower triangle (diagonal entries are ignored since a community cannot be merged with itself). The relevant entries are highlighted.

While entries on the diagonal can be sorted trivially, this does not apply to off-diagonal elements. However, it is easy to see that marked entries greater than a specific diagonal element $L = d_{(s_i)}d_{(s_i)}$ are all located in columns $< i$. This allows an efficient enumeration by looking at submatrices of increasing size. To track which entries have already been checked for each column, a status array $u = [u_i]$ is used. It contains the next element to be checked for each column and is initialized to $u_i \coloneqq i + 1$. For each diagonal element $L = d_{(s_i)}d_{(s_i)}$, the algorithm enumerates entries $d_{(s_j)}d_{(s_k)} \geq L$ with $j < i$ and $k \geq u_j$. When an element is found that is smaller than L, we can stop iterating the current column and proceed with the next one. Merges can only be done when two clusters are not already connected, so for each entry it is necessary to check if

Function FindMergeM($\mathbf{A}$, d, s)

Input: Adjacency mat. $\mathbf{A} = [a_{ij}]$, vertex deg. d, sorted index s
Output: Best merge $b = (i, j)$ or *None*

```
1  u ← empty vector of size n;
2  b ← None;
3  for i ← 1 to n do u_i ← i + 1;                    ▷ assign initial state
4  for i ← 1 to n − 1 do
5      L ← d_(s_{i+1}) d_(s_{i+1});                  ▷ compute lower bound
6      for j ← 1 to i do                             ▷ for each column
7          while u_j ≤ n do                          ▷ for each row
8              (k, l) ← (s_j, s_(u_j));              ▷ get indices of next entry
9              if d_k d_l < L then break;
10             if b ≠ None ∧ d_k d_l = L then break;
11             if a_kl = 0 then                      ▷ merge was found
12                 b ← (k, l);
13                 L ← d_k d_l;
14                 break;
15         u_j ← u_j + 1;                            ▷ proceed with next row
16     if b ≠ None then break;
17 δm ← −L / (2m²);
18 foreach (v_i, v_j) ∈ E do
19     v ← (1/m) (a_ij − d_i d_j / 2m);              ▷ compute modularity change
20     if v < δm then                                ▷ better merge was found
21         b ← (i, j);
22         δm ← v;
```

$a_{(s_i)(s_j)} = 0$. When the first non-existing edge is found, we may abort after making sure that no better solution exists. If the first element in a column is less than or equal to the previously found best solution, we simply skip it. Furthermore, we can definitely terminate after processing any element that is greater than or equal to the current lower bound L. Effectively, this step always searches for two disconnected groups with the highest vertex degree product.

For a graph with m edges, the first non-existing edge is definitely found by iteration $k_{\max}$ (or earlier), with $k_{\max}$ given by

$$k_{\max} \coloneqq \left\lfloor \frac{1}{2}\left(1 + \sqrt{8m+1}\right)\right\rfloor \leq \frac{1}{2}\left(1 + \sqrt{8m+1}\right). \tag{7}$$

This follows from the observation that at least $1/2k(k+1)$ merges have been checked until iteration k of the outer loop. The algorithm terminates after $O(\sqrt{m})$ iterations of the outer loop and $O(m)$ iterations of the inner loop. The number of changes of the vector that are necessary until the non-existing edge is found is at most m, afterwards there are at most $k_{\max} - 1$ additional state transitions to replace the solution (one for each column). As a result, the total complexity of the first search step is also $O(m)$.

Overall, the initial computation of the lookup table can be performed in $O(n \log(n))$ by sorting. Searching for a suitable merge step requires at most $O(m)$ lookup steps. The update of the adjacency matrix $\mathbf{A}$, the vector of degrees d and the lookup table s after each step can be performed in $O(n)$. Since the algorithm terminates after at most $n - 1$ iterations, the total complexity is $O(n(m + n))$.

Greedy algorithm with anti-modularity (GrAM). For our second approach, we assume that the anti-community structure of a graph is characterized by a global maximum of the anti-modularity $M_A(g)$. Since computing the global optimum is infeasible for large graphs, we propose the following greedy method. For a pseudo code description see Algorithm 2 and Function FindMergeAM.

Given the adjacency matrix $\mathbf{A}$, the first step is to compute the squared adjacency matrix $\mathbf{B} = [b_{ij}]$ with elements $b_{ij} \coloneqq \sum_k a_{ik} a_{kj}$. The algorithm starts with n clusters containing a single vertex, loops through all non-zero entries b_{ij} of $\mathbf{B}$ and computes the anti-modularity change by merging the two clusters containing $v_i \in V$ and $v_j \in V$. The merge that results in the largest gain in anti-modularity (or the smallest anti-modularity decrease) is then performed and the best solution found so far is tracked. For a connected graph, the algorithm terminates after at most $n - 1$ merges when only a single cluster is left. The result of the algorithm is the partition with the largest anti-modularity value found in all iterations.

To avoid the re-computation of the anti-modularity changes $\delta\mathbf{M}_A = [\delta m_{ij}]$ at each step, we use a similar method as in the algorithm above. In the initial configuration, where each vertex is assigned to a separate cluster, the anti-modularity is given by

$$M_A \coloneqq \frac{1}{n}\left[\mathrm{Tr}(\mathbf{B}) - \sum_{i=1}^{n} \frac{d_i^2}{n}\right]. \tag{8}$$

The change for merging two clusters i and j is

$$\delta m_{ij} \coloneqq \frac{1}{n}\left(b_{ij} + b_{ji} - \frac{2 d_i d_j}{n}\right) = \frac{2}{n}\left(b_{ij} - \frac{d_i d_j}{n}\right). \tag{9}$$

When a suitable merge is found, both $\mathbf{B}$ and the vector of degrees d can be updated by adding the corresponding rows and columns to reflect the changed connectivity.

$$\begin{aligned} \forall k : b_{ik}^{(\mathrm{tmp})} &\leftarrow b_{ik} + b_{jk} \\ \forall k : b_{ki}^{(\mathrm{new})} &\leftarrow b_{ki}^{(\mathrm{tmp})} + b_{kj}^{(\mathrm{tmp})} \\ d_i^{(\mathrm{new})} &\leftarrow d_i + d_j \end{aligned} \tag{10}$$

The total execution time of the algorithm is dominated by the time it takes to compute the squared adjacency matrix $\mathbf{B}$ and by the iterative clustering algorithm. Computing the squared adjacency matrix can be done in $O(nm)$ by using a naive matrix multiplication algorithm. Depending on the input data, the squared adjacency matrix might be dense. However, even in the worst case, the clustering algorithm still finishes in $O(n^3)$, which means that the total complexity is at most $O(n^3)$.

3.4 Vertex Similarity Algorithms

While the above algorithms use *modularity* or *anti-modularity*, we can also use a clustering based on vertex features. Here, the idea is to define a mapping $M : V \to \mathbb{R}^p$ of vertices to points (or *feature vectors*) in a higher dimensional vector space (here, p dimensions). The mapping is chosen in such a way that similar vertices are mapped to points close to each other, whereas feature vectors of unrelated vertices have a larger distance [8]. If multiple vertices have a similar relation to other vertices in the graph, we can assume that they form a community or anti-community, even if they are not adjacent themselves. This concept is called *structural equivalence*

Algorithm 2: GrAM($\mathbf{A}$): Greedy alg. for *anti-modularity* max.

Input: Adjacency matrix $\mathbf{A} = [a_{ij}]$ of size $n \times n$
Output: Vector of vertex labels g, Anti-modularity M_A

```
g ← empty vector of size n;
d ← empty vector of size n;
B ← A^2;
for i ← 1 to n do  g_i ← i;                          ▷ assign initial labels
for i ← 1 to n do  d_i = Σ_{j=1}^{n} a_ij;           ▷ compute vertex degrees
M_A ← (1/n) Σ_{i=1}^{n} (b_ii − d_i^2/n);            ▷ compute anti-modularity
while i, j ← FindMergeAM(B, d) do
    M_A ← M_A + (2/n) (b_ij − d_i d_j/n);            ▷ update anti-modularity
    for k ← 1 to n do  b_ik ← b_ik + b_jk;           ▷ add rows
    for k ← 1 to n do  b_ki ← b_ki + b_kj;           ▷ add columns
    d_i ← d_i + d_j;                                 ▷ update vertex degrees
    for k where g_k = j do  g_k ← i;                 ▷ update labels
    delete row/column j from B and d;
```

Function FindMergeAM($\mathbf{B}$, d)

Input: Squared adjacency matrix $\mathbf{B} = [b_{ij}]$, vertex degrees d
Output: Best merge $b = (i, j)$ or *None*

```
L ← −∞;
b ← None;
foreach non-zero entry b_ij do
    v ← (2/n) (b_ij − d_i d_j/n);                    ▷ compute anti-mod. change
    if v > L then                                    ▷ better merge was found
        b ← (i, j);
        L ← v;
```

and also used for many other similarity measures [16]. To obtain the community structure, we use *k-means clustering* [17] and determine the optimal number of clusters k with the *silhouette score* [27].

Vertex similarity with adjacency mapping (VSA). As a starting point, we group vertices that share common neighbors. This idea is closely related to the work by Burt [3], who suggests that a dissimilarity measure can be defined on a graph $\mathcal{G}$ as:

$$d_{ij} \coloneqq \sqrt{\sum_{k \neq i,j} (a_{ik} - a_{jk})^2} \tag{11}$$

The condition $k \neq i, j$ ensures that edges between vertices v_i and v_j do not increase the dissimilarity. Since we are looking for anti-communities in this case, we expect that the dissimilarity increases when two vertices in the same anti-community are connected to each other. This allows us to simplify the formula to

$$d_{ij}^{\mathrm{A}} \coloneqq \sqrt{\sum_{k} (a_{ik} - a_{jk})^2} \tag{12}$$

which is equivalent to distances in an euclidean vector space when vertices are mapped to rows of the adjacency matrix, i.e.,

$$M_{\mathrm{adj}}(v_i) \coloneqq [a_{ij}]_j. \tag{13}$$

After mapping vertices to their corresponding feature vectors, *k-means clustering* can be used to partition the graph. This concept is closely related to the anti-modularity measure, which is also motivated by the idea that two vertices v_i and v_j belong to the same anti-community if they have many neighbors in common.

Vertex similarity with distance mapping (VSD). Alternatively, vertices can be mapped to vectors containing shortest distances to all other vertices. Assume $d(v_i, v_j)$ is the shortest distance between two vertices v_i and v_j, then such a mapping function is

$$M_{\text{dist}}(v_i) \coloneqq [d(v_i, v_1), \ldots, d(v_i, v_n)]. \tag{14}$$

Based on this definition, two vertices are grouped when they have similar distances to other vertices in the graph. In practice, we can obtain the shortest distances between all vertices, e.g., by using the *Floyd Warshall algorithm* or *Johnson's algorithm* [5].

In Table 1, we provide an overview of all baseline algorithms, as well as the proposed algorithms, along with their time complexities. The parameter i is used to denote the number of iterations.

4 EVALUATION

From a theoretical point of view, all of the above methods are able to detect anti-community structures. In practice, however, they often do not agree on the same partition. Vertex similarity methods, for example, tend to return partitions with more groups than other methods. To evaluate which algorithm performs best under certain conditions, we propose adapted versions of the *Erdős-Rényi* and *Barabási-Albert* random graph models. These generative models can be used to sample random graphs with specific community or anti-community structure, which in turn can be used to compare the performance and runtime between different algorithms.

4.1 Synthetic Evaluation Data

The original versions of the *Erdős-Rényi* and *Barabási-Albert* random graph model can be used to generate graphs without community or anti-community structure. For evaluation purposes, however, we need graphs that contain such a structure. This can be achieved by extending the graph models with additional parameters. For both models, we add parameters for the number of groups k, for the probability of connections within each group p_{int}, and for the probability of connections between different groups p_{ext}.

Adapted Erdős-Rényi model. The original *Erdős-Rényi* random graph model allows us to construct graphs $G(n, p)$ based on the number of vertices n and a fixed probability p that two vertices are connected to each other [6]. To generate graphs with group structure, we extend this parameterization to $G(n, k, p_{\text{int}}, p_{\text{ext}})$, with k, p_{int}, and p_{ext} as described above. The proposed algorithm works as follows: First, all n vertices are evenly distributed between the k groups. Afterwards, random numbers between 0 and 1 are drawn for each edge $(v_i, v_j) \in V^2$. Edges are added only when the number is less than p_{int} for edges within a community, or p_{ext} for edges between different communities. Note that this is equivalent to a *block model* with entries on the diagonal set to p_{int} and remaining entries set to p_{ext}. The pseudo code of the proposed method is shown in Algorithm 3. The notation $p_{(g_i)(g_j)}$ is used as a shortcut for

$$p_{(g_i)(g_j)} \coloneqq \begin{cases} p_{\text{int}} & \text{if } g_i = g_j, \\ p_{\text{ext}} & \text{otherwise.} \end{cases} \tag{15}$$

Table 1: Overview of baseline algorithms, as well as proposed methods, along with their complexity.

Algorithm	Complexity
Graph complement + Greedy mod. (GCM) [18]	$O(n^3)$
Label propagation algorithm (LP) [4][a]	$O(n^2)$
Stochastic block model (SBM) [21]	—
Nested stochastic block model (NSBM) [23]	$O(in \log^2 n)$
Greedy alg. with modularity (GRM)	$O(n^2 + nm)$
Greedy alg. with anti-modularity (GRAM)	$O(n^3)$
Vertex sim. with adjacency mapping (VSA)[b]	$O(ik^2n^2)$
Vertex sim. with distance mapping (VSD)[b]	$O(n^3 + ik^2n^2)$

[a] The publication has no thorough proof that the complexity holds for all graphs.
[b] Including a linear search for the optimal number of clusters.

Algorithm 3: Adapted *Erdős-Rényi* random graph model

input : Parameters n, k, p_{int}, and p_{ext}
output: Adjacency matrix $\mathbf{A} = [a_{ij}]$

$\mathbf{A} \leftarrow$ empty matrix of size $n \times n$;
$g \leftarrow$ empty vector of size n;
for $i \leftarrow 1$ **to** n **do** $g_i \leftarrow \left\lfloor \frac{(i-1)k}{n} \right\rfloor + 1$; ▷ assign vertex labels
apply random permutation to g; ▷ randomize vertex labels
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow i + 1$ **to** n **do** ▷ initialize adjacency matrix
    $a_{ij} \leftarrow a_{ji} \leftarrow \begin{cases} 1 & \text{if } \textsc{Rand}() < p_{(g_i)(g_j)}, \\ 0 & \text{otherwise.} \end{cases}$

Algorithm 4: Adapted *Barabási-Albert* random graph model

input : Parameters n, m_0, k, p_{int} and p_{ext}
output: Adjacency matrix $\mathbf{A} = [a_{ij}]$

$\mathbf{A} \leftarrow$ empty matrix of size $n \times n$;
$g \leftarrow$ empty vector of size n;
for $i \leftarrow 1$ **to** n **do** $g_i \leftarrow \left\lfloor \frac{(i-1)k}{n} \right\rfloor + 1$; ▷ assign vertex labels
apply random permutation to g; ▷ randomize vertex labels
$N \leftarrow \{1, \ldots, n\}$;
$C \leftarrow$ empty associative array;
while $|N| > 0$ **do** ▷ add vertices one after another
  $s \leftarrow \textsc{RandChoice}(N)$; ▷ select source vertex
  remove s from N;
  if C *is empty* **then** ▷ compute probabilities
    $D \leftarrow \{t : p_{(g_s),(g_t)} \text{ for } t \in N\}$; ▷ first iteration
  else
    $D \leftarrow \{t : w p_{(g_s),(g_t)} \text{ for } t : w \in C\}$;
  for t **in** $\textsc{WeightedRandSample}(D, m_0)$ **do**
    $C[t] \leftarrow C[t] + 1$; ▷ update vertex degrees
    $a_{st} \leftarrow a_{ts} \leftarrow 1$; ▷ add undirected edge
    remove t from N (if not done yet);
  $C[s] \leftarrow C[s] + m$; ▷ update vertex degrees

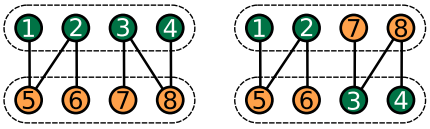

Figure 2: Example graph with two connected components. Based on the connectivity pattern, both clusterings are plausible, but only the left one has an ARI and NMI of one.

Adapted Barabási-Albert model. While edges are added independently in the *Erdős-Rényi* random graphs, the *Barabási-Albert* model uses the concept of preferential attachment [1]. Intuitively, this means that vertices with many existing edges are more likely to receive new edges. The original model $G(n, m_0)$ takes two parameters n and m_0, with n being the number of vertices. The first m_0 vertices are added to the graph immediately without any connections. Afterwards, the remaining $n - m_0$ vertices are added one after another. Each newly added vertex establishes connections with m_0 existing vertices in the graph. The edges are chosen with probabilities proportional to the vertex degrees. To generate graphs with community or anti-community structure, we propose a new parameterization $G(n, m_0, k, p_{\text{int}}, p_{\text{ext}})$. Since the absolute value of the probabilities does not matter (the values are normalized again) we can fix one of them by imposing the additional constraint $p_{\text{int}} + p_{\text{ext}} = 1$. Similar to the *Erdős-Rényi* model, we assume that each community consists of exactly n/k vertices. The main difference compared to the original method is that, in addition to the vertex degrees, p_{int} and p_{ext} are also taken into account to make certain edges more likely than others. The pseudo code of this method is shown in Algorithm 4. Note that WeightedRandSample degenerates to a uniform random choice when all weights are zero. A graph generated with this model is expected to have $(n - m_0)m_0$ edges and has only a single connected component.

4.2 Evaluation Metrics

Whenever ground-truth labels are available, they can be used to rate the quality of the partitions that are returned by the algorithms. In our evaluation, we use the *adjusted Rand index* measure as well as the *normalized mutual information* measure, which are both well-established methods for cluster analysis.

However, we find that these measures do not work as expected when a graph has multiple connected components. Anti-community detection algorithms have no information about how labels in one component are related to labels in a different component, so there are many different equivalent solutions to the expected partitioning. An example of this effect is shown in Figure 2 for two possible clusterings of the same graph. Note that a similar situation occurs when applying traditional community detection methods to a network with two (or more) fully connected subgraphs. In the following, we discuss the original measures and our proposed modifications.

Adjusted Rand Index (ARI). Assume that we want to compare a label vector $g \in \mathbb{N}^n$ with the ground-truth label vector $h \in \mathbb{N}^n$. Both the *Rand index* and the *adjusted Rand index* measure are based on counting pairs of matches. Suppose that *a*,*b*,*c*, and *d* are the true/false positive/negative label matches when comparing all pairs of vertices, as shown in the following table.

	$h_i = h_j$	$h_i \neq h_j$
$g_i = g_j$	true positive (a)	false positive (b)
$g_i \neq g_j$	false negative (c)	true negative (d)

Using these variables, the *Rand index* R can be defined as [26]:

$$R := \frac{a+d}{a+b+c+d}. \tag{16}$$

The *Rand index* evaluates to values between 0 and 1, where 1 means that both structures are identical and labels can be bijectively mapped to each other, and 0 means that they have nothing in common. Despite its popularity, the *Rand index* is usually not the preferred choice as it suffers from the problem that the score of two random partitions is not a constant value. Moreover, the score approaches a value of 1 when the number of clusters is increased. The *adjusted Rand index*, which tries to solve these problems, is defined as:

$$\text{ARI} := \frac{R - E(R)}{1 - E(R)} \tag{17}$$

with $E(R) := [(a+b)(a+c) + (c+d)(b+d)](a+b+c+d)^{-2}$. Intuitively, the normalization ensures that random matches have an average adjusted Rand index of zero. The adjusted Rand index has a negative value if R is smaller than the expected index. Two clusters are identical when the ARI reaches a value of 1.

To avoid misleading results when there is no unique solution, we propose a modification, such that only those pairs of vertices are counted that are both part of the same connected component. Assume $\bar{a}$, $\bar{b}$, $\bar{c}$, and $\bar{d}$ are the true/false positive/negative label matches when counting only pairs of vertices belonging to the same connected component. Then the modified *Rand index* R_{c} is:

$$R_{\text{c}} := \frac{\bar{a}+\bar{d}}{\bar{a}+\bar{b}+\bar{c}+\bar{d}}. \tag{18}$$

Similarly, the modified *adjusted Rand index* can be obtained as:

$$\text{ARI}_{\text{c}} := \frac{R_{\text{c}} - E(R_{\text{c}})}{1 - E(R_{\text{c}})} \tag{19}$$

with $E(R_{\text{c}}) := [(\bar{a}+\bar{b})(\bar{a}+\bar{c}) + (\bar{c}+\bar{d})(\bar{b}+\bar{d})](\bar{a}+\bar{b}+\bar{c}+\bar{d})^{-2}$. For graphs with a single components $\text{ARI} = \text{ARI}_{\text{c}}$, otherwise the modified measure returns a value of 1 for all plausible assignments.

Normalized mutual information (NMI). Mutual information describes the information (in bits) that can be obtained about one variable from another variable, and can also be used as a similarity measure for clusterings. Assume that $X = \{X_1, X_2, \ldots, X_r\}$ with $X_i \subset V$ are the r distinct clusters returned by the algorithm and $Y = \{Y_1, Y_2, \ldots, Y_s\}$ with $Y_i \subset V$ are the s distinct clusters based on the ground truth labels. Further assume that there is a total of $n := \sum_i |X_i| = \sum_j |Y_j|$ vertices. $|X_i \cap Y_j|$ counts the number of vertices that are in both X_i and Y_j. The mutual information of X and Y is then defined as:

$$I(X,Y) := \sum_i \sum_j \frac{|X_i \cap Y_j|}{n} \log_2 \frac{n|X_i \cap Y_j|}{|X_i| \cdot |Y_j|}. \tag{20}$$

The normalized mutual information NMI (also called *symmetric uncertainty* [32]) with a value between 0 and 1 is given by

$$\text{NMI}(X,Y) := \frac{2I(X,Y)}{I(X,X) + I(Y,Y)}. \tag{21}$$

Assume that a graph contains multiple connected components $C = \{C_1, C_2, \ldots, C_t\}$ with $C_i \subset V$ being the distinct connected components of the graph. For each component, $X_i = \{X_{i,1}, X_{i,2}, \ldots\}$ refers to distinct clusters returned by the algorithm and $Y_i = \{Y_{i,1}, Y_{i,2}, \ldots\}$ are the distinct clusters based on the ground truth labels. We can then define a modified *normalized mutual information* measure NMI_c compatible with multiple connected components as:

$$\mathrm{NMI}_c(X, Y) := \frac{2\sum_{i=1}^{t} |C_i| I(X_i, Y_i)}{\sum_{i=1}^{t} |C_i| \left[I(X_i, X_i) + I(Y_i, Y_i)\right]}. \quad (22)$$

4.3 Anti-community Detection Evaluation

For both Erdős-Rényi and Barabási-Albert random graphs we have introduced parameters p_{int} and p_{ext} to control the probability of edges within and between groups. To evaluate the detection of anti-communities, we can sample graphs with certain properties, and then verify if algorithms are able to correctly identify the contained groups. For the following experiments, the number of vertices is fixed to $n = 30$ and the number of (anti-)communities is set to $k \in \{2, 5\}$. For each choice of $\Delta p = p_{\text{ext}} - p_{\text{int}}$ various random graphs are generated and evaluated using the measures above. While $\Delta p = +1$ corresponds to a strong anti-community structure, $\Delta p = -1$ represents a strong community structure. Intuitively, we expect that algorithms perform best when $|\Delta p|$ is large.

For Barabási-Albert random graphs, we can set $p_{\text{ext}} = 1/2(1 + \Delta p)$ and $p_{\text{int}} = 1/2(1 - \Delta p)$, such that $p_{\text{ext}} + p_{\text{int}} = 1$. In this case we use a fixed number of 300 runs per choice and divide the interval $\Delta p \in [-1; +1]$ into 100 steps. The model always generates graphs with a single connected component, so $\mathrm{ARI} = \mathrm{ARI}_c$. For Erdős-Rényi random graphs p_{ext} and p_{int} can be chosen independently. Still, we find that the absolute values do not have a big influence, so we look at the average over all possible combinations. The interval $\Delta p \in [-1; +1]$ is divided into 100 steps and for each choice we use about 2000 random graphs with different combinations of $(p_{\text{int}}, p_{\text{ext}})$. The difference between ARI and ARI_c is also negligible here because it only has an effect when $p_{\text{ext}} \approx 0$ and $p_{\text{int}} \approx 0$.

The results of the experiments are shown in Figure 3. Error bars are omitted for the sake of readability. Nevertheless, since we are primarily interested in the relative performance and not in exact values, we can still draw valid conclusions. As expected, the plots show that certain algorithms only work for anti-communities (GCM, LP, and GRM), while other algorithms work for both communities and anti-communities (SBM, NSBM, GRAM, VSA, and VSD). In a scenario with pure cliques or multipartite structures, all algorithms are able to achieve an *adjusted Rand index* of 1.

For community structures (i.e., $\Delta p < 0$), we notice that SBM fails first when the structure becomes more ambiguous. The nested stochastic block model NSBM works better, but still performs worse than other algorithms in this experiment. For a small number of groups, both VSA and GRAM work approximately equally well. This is surprising since GRAM uses a heuristic and is much faster than computing an exact clustering of feature vectors as in the similarity method. It also confirms that both methods use similar information to decide which vertices should be grouped. The VSD algorithm is the best method for community detection in our test. However, note that we do not include all standard community detection algorithms in our tests since we focus on anti-communities.

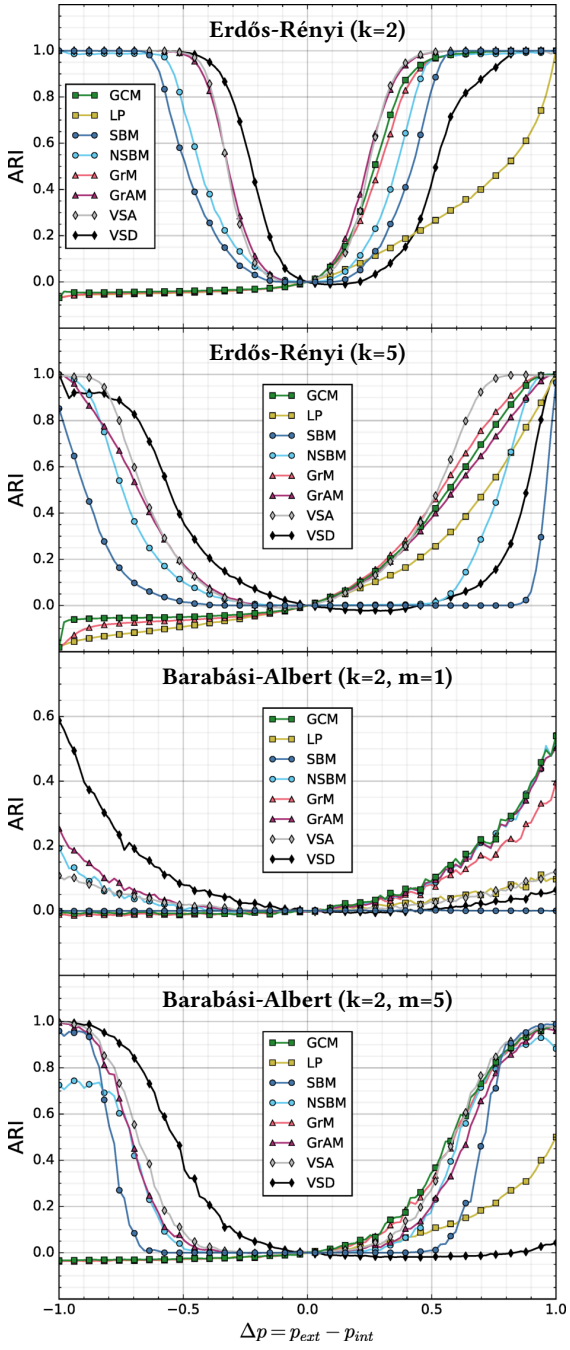

Figure 3: Performance evaluation of all methods on Erdős-Rényi and Barabási-Albert random graphs.

For anti-community structures (i.e., $\Delta p > 0$), the methods LP, VSD, SBM, and NSBM turn out to be unreliable and produce results that are much worse when compared to the other methods. The actual order depends on the number of communities: for graphs with many communities and structures close to the resolution limit (i.e., $k \propto \sqrt{n}$ [22]), LP performs significantly better than SBM or VSD. This complex of problems is also what motivated the creation

of the nested version NSBM. For VSD, it is not immediately clear what causes this effect. The other methods GC, GRM, GRAM and VSA perform approximately equally well and have the highest performance according to our experiments. For Erdős-Rényi graphs with many communities, VSA seems to be the preferred choice.

The fact that Barabási-Albert random graphs perform worse can be explained by the way the graphs are constructed. For fixed parameters n and m_0, the number of edges is always exactly $(n - m_0)m_0$. This means that even for $\Delta p = \pm 1$, the groups are only weakly connected internally or externally, which makes the structure more difficult to detect. Indeed, we can see an improvement when changing m_0 from 1 to 5. Another difficulty is that Barabási-Albert random graphs always consist of a single connected component, i.e., there are always edges between different groups of vertices.

4.4 Runtime Evaluation

In the following, we take a closer look at the runtime behavior. Although the asymptotic worst-case complexities are known, the average case runtime is much more important in practice. Furthermore, the theoretical complexity does not give us any insight into the runtime of a specific implementation, which also depends on the constant factors, the processor speed, and optimizations by the programmer or compiler. To analyze how fast the proposed algorithms are in practice, we sample graphs from the *Erdős-Rényi* random graph model and measure the runtime experimentally. We fix the number of clusters to $k = 5$ and set $p_{\text{int}} = 0$. For dense graphs we set $p_{\text{ext}} = 1$, for sparse graphs we set

$$p_{\text{ext}} := \frac{n}{\binom{n}{2} - k\binom{n/k}{2}}. \tag{23}$$

By repeating this experiment for different choices of n, we can visualize the runtime behavior $T(n)$ for each of the algorithms. The experiments were run on an Intel Xeon CPU E5-2650 v3 @ 2.3 GHz without multithreading to ensure that the results are comparable. The measurements are shown in Figure 4.

According to this test, LP is the fastest of all algorithms. For sparse graphs, the proposed algorithms GRM and GRAM are slower by a constant factor, but show a similar scaling behavior. Methods based on computing the graph complement like GCM have a much steeper slope, which confirms that they are less suitable for large networks. For SBM, no data could be obtained for $n < 25$ because the implementation consistently crashed with a segmentation fault. Even for $n \geq 25$, we still encountered sporadic crashes of the application (these measurements were omitted from the final result). The scaling behavior of SBM looks a bit better, but it is important to keep in mind that the algorithm uses a fixed number of iterations. NSBM is the slowest algorithm, despite the fact that the performance-critical parts are implemented in C++. As suggested by Peixoto, maximizing the posterior probability instead of using sampling could be used to improve the runtime, at the cost of producing less generic results [24].

4.5 Performance Discussion

Choosing a suitable algorithm is a matter of balancing performance and runtime constraints. If the graph is sufficiently small (e.g., if it has less than 10^2 vertices), the runtime is not really a concern. All

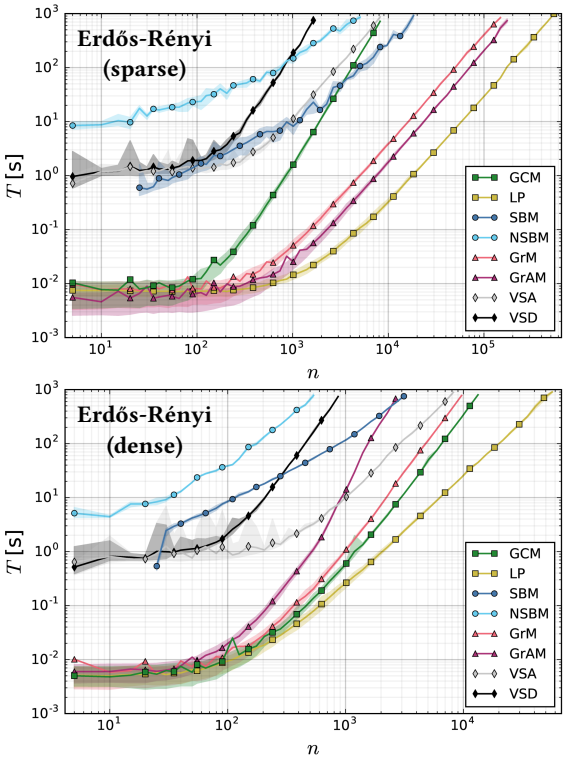

Figure 4: Runtime comparison of the anti-community detection algorithms on Erdős-Rényi random graphs.

algorithms terminate within at most 20 seconds. Based on the performance experiments, VSA performs best on average, followed by the greedy methods GRM and GRAM. According to our evaluation, NSBM with sampling is the slowest of all algorithms. Nevertheless, as can be seen in the following section, NSBM has clear advantages for networks with a nested structure. For performance critical tasks, or when the network is large enough, methods like GRM are more suitable. While LP is even faster, it only provides a very imprecise approximation when applied to complex networks. GRM can process sparse graphs with about 10^5 vertices in less than 10^3 seconds on a single core. Using multiple cores and parallel or distributed computing can reduce the runtime even further. Explicit computation of the graph complement, as done by the GCM method, is infeasible when trying to process large networks.

Based on our findings here, we discuss the detection of anti-community structures in real networks in the following section.

5 EXPLORATORY ANALYSIS

Traditional community structures can be found in many networks, which also helped to establish community detection as a standard tool of network science. Anti-community structures are much harder to recognize but are nevertheless present even in community-rich networks. To demonstrate this, we apply anti-community detection algorithms to the well-known *Zachary's karate club* network. As a second example, we apply anti-community detection algorithms to a network of spectral line transitions. This shows that anti-communities also have a use-case in the context of Physics.

5.1 Zachary's Karate Club Network

Zachary's karate club network was created by observing the members of a karate club in the United States [33]. Vertices in the graph correspond to members and edges connect individuals if they also had interactions outside of the activities of the club. The graph contains 34 vertices and a total of 78 edges. At some point, a conflict between the two leaders occurred, which resulted in a fission of the club. Zachary observed that the groups after the fission closely matched the result of a minimum cut algorithm applied to the original network, i.e., the groups were already present before and only weakly connected with each other. Nowadays the network is primarily used for benchmarking community detection algorithms [8]. The community structure of the network has been extensively analyzed in existing work, so we focus on uncovering the anti-community structure.

For evaluation purposes, different algorithms are applied to the unweighted version of the network. Since there are no ground-truth labels for anti-communities, we instead take a look at the modularity and anti-modularity measures. As previously discussed, both a small modularity and a high anti-modularity can indicate the presence of anti-community structures. The experimental results are summarized in Table 2. Surprisingly, even SBM, which is designed to return the most likely community structure in the network, finds multiple anti-communities, instead of a partition into two communities. The smallest modularity is obtained by the greedy modularity minimization algorithm GrM, the largest anti-modularity measure by the VSD method. The result of the GrM algorithm is also shown in Figure 5. Unfortunately no further information about the members is available, so the exact meaning of the partition remains unclear. Both leaders with the aliases Mr. Hi (vertex 1) and John A. (vertex 34) were assigned to a single anti-community, but the same partition also contains mixed membership from both groups.

5.2 Atomic Spectral Line Network of Helium

A network of spectral line transitions serves as our second example. Spectral lines occur as a result of the interaction between photons and atoms or molecules. When a photon has the right amount of energy, it can change the energy state of the system and transfer an electron to a higher orbital. When the system spontaneously falls back to the original state, the same energy is re-emitted, either as a single photon or in a cascade. The result becomes visible as one or more *emission* or *absorption lines*, which can be measured experimentally. A collection of such measurements has been aggregated by the National Institute of Standards and Technology (NIST) as part of the *Atomic Spectra Database* [13]. In the following, we use the data set to construct a graph. Vertices are used to represent states, and edges are added between two vertices if a spectral line has been observed between the corresponding states. The resulting network contains 183 vertices and 2,282 edges. In theory, networks of spectral line transitions are fully connected. In practice, however, many transitions are suppressed by selection rules, which follow directly from the laws of Physics, and thus cannot be observed. Vertices with similar neighbors often play similar roles in the network and, for example, share the same quantum number.

From a physical point of view, it makes sense to group vertices (energy states) based on three quantum numbers: orbital angular

Table 2: Modularity, anti-modularity, adjusted Rand index, and normalized mutual information for both real networks.

	GC	LP	SBM	NSBM	GrM	GrAM	VSA	VSD
Zachary's karate club network								
M	-0.246	-0.202	-0.223	0.030	**-0.249**	-0.053	-0.058	0.290
M_A	111.2	73.7	122.2	71.9	111.7	149.5	49.6	**154.1**
Atomic spectral line network								
M	**-0.485**	-0.419	-0.083	-0.049	**-0.485**	-0.431	-0.038	-0.050
M_A	388.4	353.7	241.6	145.4	388.4	**395.4**	116.0	153.1
ARI	0.067	0.074	0.543	0.791	0.067	0.110	0.754	**0.842**
NMI	0.345	0.395	0.864	0.939	0.345	0.517	0.891	**0.957**

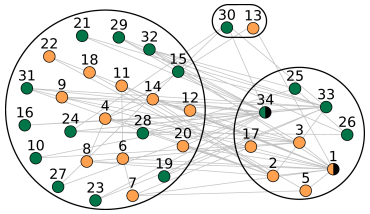

Figure 5: Anti-communities detected with the GrM algorithm in the Zachary's karate club network (denoted by circles). Colors show membership after the fission.

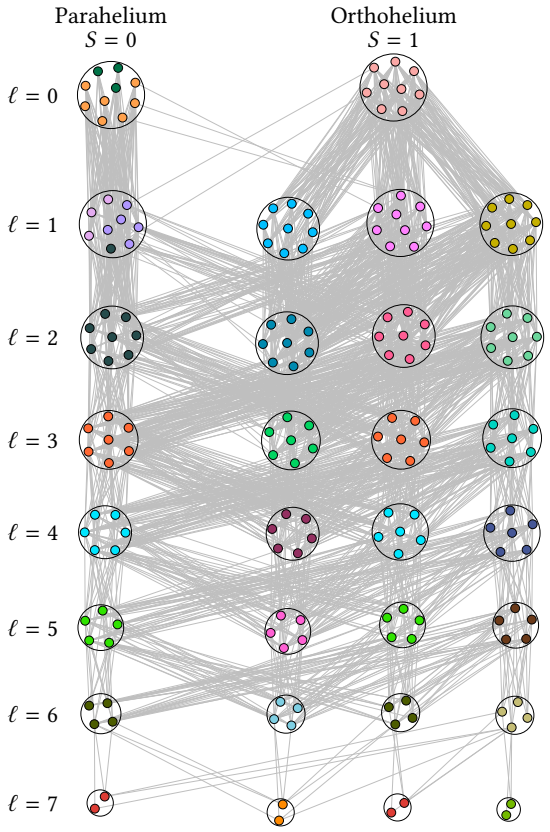

Figure 6: Anti-communities detected with the VSD algorithm in the spectral line network of helium (denoted by colors). Circles show the ground-truth partition.

momentum (ℓ), total angular momentum (j), and spin (s). In the following, such a partition is considered as ground-truth. The results for different algorithms are shown in Table 2. According to the experiments, the smallest modularity is achieved by GrM, the highest anti-modularity by the GrAM algorithm. Closer investigation shows that these methods cannot resolve the fine-grained structure of the data set. According to both ARI and NMI, the VSD algorithm works best, followed by NSBM. It reaches an ARI of 0.842 when comparing the algorithm result with the ground-truth structure. A visualization of the result is shown in Figure 6, which suggests that we can indeed identify states with similar quantum numbers just by analyzing the network structure. A remaining problem is that certain states that only differ in their spin cannot be distinguished very well. Such states are assigned to the same anti-community, although we expect a partition into two groups.

6 SUMMARY AND OUTLOOK

In this paper, we discussed different approaches for efficient anti-community detection in networks. We presented four novel methods and compared them to various baseline methods with respect to their performance and runtime behavior, using adapted versions of the *Erdős-Rényi* and *Barabási-Albert* random graph models. We also presented modified versions of the *adjusted Rand index* and *normalized mutual information* measures, which should be used when evaluating partitions of graphs with multiple connected components. Exploratory analyses of real networks showed that anti-community structures are present even in the well-known *karate club* network. For spectral line networks, anti-community detection can be used to group states with similar quantum numbers.

Ongoing work. Many of the ideas investigated in this paper are not limited to anti-community detection. The concepts that we developed for the GrM algorithm, for example, can be used to improve agglomerative hierarchical clustering techniques for community detection. Certain methods, like GrAM and VSD, also work surprisingly well for community detection, which we plan to further investigate. Last but not least, it might be useful to investigate algorithms for multigraphs with both attracting (→ community) and repelling (→ anti-community) types of edges [31]. Such an approach might allow the unification of community and anti-community detection in a single generalized concept.

ACKNOWLEDGMENTS

The authors would like to thank David Wellnitz, Armin Kekić, Julian Heiss, Kathinka Gerlinger, and Erich Schubert for constructive discussions and helpful suggestions.

REFERENCES

[1] A. Barabási and R. Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512. https://doi.org/10.1126/science.286.5439.509
[2] E. R. Barnes. 1982. An algorithm for partitioning the nodes of a graph. *SIAM J. Alg. Discr. Meth.* 3, 4 (1982), 541–550. https://doi.org/10.1137/0603056
[3] R. S. Burt. 1976. Positions in networks. *Soc. Forces* 55, 1 (1976), 93–122. https://doi.org/10.1093/sf/55.1.93
[4] L. Chen, Q. Yu, and B. Chen. 2014. Anti-modularity and anti-community detecting in complex networks. *Inf. Sci.* 275 (2014), 293–313. https://doi.org/10.1016/j.ins.2014.02.040
[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms, 3rd Edition.* The MIT Press.
[6] P. Erdős and A. Rényi. 1959. On random graphs I. *Publicationes Mathematicae* 6 (1959), 290–297. https://doi.org/10.2307/1999405
[7] D. Fasino and F. Tudisco. 2018. A modularity based spectral method for simultaneous community and anti-community detection. *Linear Algebra Appl.* 542 (2018), 605–623. https://doi.org/10.1016/j.laa.2017.12.001
[8] S. Fortunato. 2010. Community detection in graphs. *Phys. Rep.* 486, 3 (2010), 75–174. https://doi.org/10.1016/j.physrep.2009.11.002
[9] S. Fortunato and M. Barthélemy. 2007. Resolution limit in community detection. *Proc. Natl. Acad. Sci.* 104, 1 (2007), 36–41. https://doi.org/10.1073/pnas.0605965104
[10] M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proc. Natl. Acad. Sci.* 99, 12, 7821–7826. https://doi.org/10.1073/pnas.122653799
[11] R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral. 2004. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E* 70, 2 (2004), 9. https://doi.org/10.1103/PhysRevE.70.025101
[12] B. W. Kernighan and S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* 49, 2 (1970), 291–307. https://doi.org/10.1002/j.1538-7305.1970.tb01770.x
[13] A. Kramida, Y. Ralchenko, J. Reader, and NIST ASD Team. 2015. NIST Atomic Spectra Database (ver. 5.3), [Online]. Available: `http://physics.nist.gov/asd` [2017, July 4]. National Institute of Standards and Technology, Gaithersburg, MD.
[14] X. Liu, D. Li, S. Wang, and Z. Tao. 2007. Effective algorithm for detecting community structure in complex networks based on GA and clustering. In *ICCS*. https://doi.org/10.1007/978-3-540-72586-2_95
[15] B. Long, X. Xu, Z. Zhang, and P. S. Yu. 2007. Community learning by graph approximation. In *ICDM*. https://doi.org/10.1109/ICDM.2007.42
[16] F. Lorrain and H. C. White. 1971. Structural equivalence of individuals in social networks. *J. Math. Sociol.* 1, 1 (1971), 49–80. https://doi.org/10.1080/0022250X.1971.9989788
[17] J. Macqueen. 1967. Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*. 281–297.
[18] M. E. J. Newman. 2004. Fast algorithm for detecting community structure in networks. *Phys. Rev. E* 69, 6 (2004), 5. https://doi.org/10.1103/PhysRevE.69.066133
[19] M. E. J. Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* 74, 3 (2006), 19. https://doi.org/10.1103/PhysRevE.74.036104
[20] M. E. J. Newman and M. Girvan. 2004. Finding and evaluating community structure in networks. *Phys. Rev. E* 69, 2 (2004), 15. https://doi.org/10.1103/PhysRevE.69.026113
[21] M. E. J. Newman and G. Reinert. 2016. Estimating the number of communities in a network. *Phys. Rev. Lett.* 117 (2016), 5. Issue 7. https://doi.org/10.1103/PhysRevLett.117.078301
[22] T. P. Peixoto. 2013. Parsimonious module inference in large networks. *Phys. Rev. Lett.* 110 (2013), 5. Issue 14. https://doi.org/10.1103/PhysRevLett.110.148701
[23] T. P. Peixoto. 2014. Hierarchical block structures and high-resolution model selection in large networks. *Phys. Rev. X* 4 (2014), 18. Issue 1. https://doi.org/10.1103/PhysRevX.4.011047
[24] T. P. Peixoto. 2017. Bayesian stochastic blockmodeling. (2017), 44. https://arxiv.org/abs/1705.10225
[25] C. Pizzuti. 2009. A multi-objective genetic algorithm for community detection in networks. In *ICTAI*. https://doi.org/10.1109/ICTAI.2009.58
[26] W. M. Rand. 1971. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.* 66, 336 (1971), 846–850. https://doi.org/10.2307/2284239
[27] P. J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7
[28] R. Shang, J. Bai, L. Jiao, and C. Jin. 2013. Community detection based on modularity and an improved genetic algorithm. *Physica A* 392, 5 (2013), 1215–1231. https://doi.org/10.1016/j.physa.2012.11.003
[29] P. R. Suaris and G. Kedem. 1988. An algorithm for quadrisection and its application to standard cell placement. *IEEE Trans. Circuits Syst.* 35, 3 (1988), 294–303. https://doi.org/10.1109/31.1742
[30] M. Tasgin, A. Herdagdelen, and H. Bingol. 2007. Community detection in complex networks using genetic algorithms. (2007). https://arxiv.org/abs/0711.0491
[31] V. A. Traag and J. Bruggeman. 2009. Community detection in networks with positive and negative links. *Phys. Rev. E* 80, 3 (2009), 6. https://doi.org/10.1103/PhysRevE.80.036115
[32] I. H. Witten and E. Frank. 2005. *Data Mining: Practical machine learning tools and techniques, 2nd Edition.* Morgan Kaufmann.
[33] W. W. Zachary. 1977. An information flow model for conflict and fission in small groups. *J. Anthropol. Res.* 33, 4 (1977), 452–473. https://doi.org/10.1086/jar.33.4.3629752
[34] M. Zarei and K. A. Samani. 2009. Eigenvectors of network complement reveal community structure more accurately. *Physica A* 388, 8 (2009), 1721–1730. https://doi.org/10.1016/j.physa.2009.01.007
[35] J. Zhu, Y. Liu, Y. Zhang, X. Liu, Y. Xiao, S. Wang, and X. Wu. 2017. Exploring anti-community structure in networks with application to incompatibility of traditional Chinese medicine. *Physica A* 486 (2017), 31–43. https://doi.org/10.1016/j.physa.2017.04.175