Beyond Dense Subgraphs: Nestedness, Hierarchies, and Community Structures in Complex Networks

PhD Thesis

Imre Gera

Supervisors: András Pluhár, PhD and András London, PhD

Doctoral School of Informatics

Department of Computational Optimization

Faculty of Science and Informatics

University of Szeged

Szeged
2024

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Models play a key role in how we analyze and predict the behavior of our environment. They create a space in which we can simulate what would happen to the real system if we changed its parameters. Such systems can be the Internet, where we need to route traffic so that it reaches the destination in the fewest hops; transportation networks, where we can tweak the programs of traffic lights to reduce congestions; or social networks, where we can identify people who are capable of influencing large groups. Models that represent various types of interactions between entities in systems, like the previous examples, use the toolkit of network science. Network science has proven itself in a wide variety of fields, including social sciences [1], economics [2], scientometrics [3], and ecology [4]. It has been successfully applied to gain a better understanding of neurological diseases [5], to show that community structures in social networks help slow the spread of diseases [6], or to identify communities in criminal networks [7], among many others.

Brandes et al. describe network science as the "study of the collection, management, analysis, interpretation, and presentation of data" [8]. Its strength lies in the ability to represent data and systems as networks, making it easier to understand a complex system by describing it as a collection of *nodes* and *links*. This simple concept allows us to characterize a wide variety of systems. To highlight a few examples, links can have directionality (as in road networks), they can run only between nodes of different classes (e.g., buyers and shops), or we can attach numerical information to the links to represent, for example, capacity. Note that the term *network* is often used interchangeably with *graph*, with the latter often seen as the abstract, mathematical representation of the former. As such, they will be used interchangeably in this thesis, too.

In the 21st century, with the popularity of Internet of Things (IoT) devices, an increased supply of Internet-connected devices, a huge increase in computing power and the new dawn of large neural networks, the amount of data that needs to be processed has increased enormously. As a result, the graphs modeling these data sets have also become much larger, requiring a change in how we process them. One feature that requires processing the entire graph (sometimes multiple times) is its *community structure*: groups of vertices with a high amount of edges among members

of the group relative to the number of edges between members of the group and outside vertices. These groups typically share common properties or play similar roles within the network. One of the most commonly used terms for methods that detect these structural patterns is *community detection*. The use of the expression is often confusing, as we sometimes see *overlapping community detection* and *partitioning* (or *clustering*, or even *graph coloring*) used to refer to the same concept.

The definition of a *community* is also far from being precise, and the term is often used interchangeably with the word *cluster*. Since I will work a lot with both terms, I will make a clear distinction between *(overlapping) community detection* and *(non-overlapping) clustering* in this work. Namely, I will talk about *communities* when an entity can belong to multiple groups – just as we, humans, are part of groups of friends, family, or workplace acquaintances, – and I will use the term *clusters* when each entity belongs to exactly one group. The latter is perhaps a less realistic representation to think in, but it is much simpler to work with, and in some cases, even preferable. For example, the karate club studied by Zachary [9] split into two clubs, where nobody remained a member of both clubs. In this case, we face a clustering problem when trying to find the new clubs within the interaction network.

Locating groups is only one dimension of the problem of community detection, though. While we frequently work with the karate club as an example of finding the two new clubs in the interaction network, these groups often have another driving force: hierarchies. Typically, not everyone in a community or cluster is equal. In the simplest example, the two new karate clubs had leaders: one part of the old club stayed with the club administrator "John A", and a new club formed around the instructor "Mr. Hi". This leads us to the notion that these groups not only have (in some representations, common) members, but also hierarchical relationships within or among themselves.

In a general sense, hierarchies form an important extension of communities and are a defining factor of our lives. In economics, one area of focus is how hierarchies help firms scale to today's employment numbers. Radner argues that hierarchies in firms can represent organizations of tasks and work, and that they are "remarkably effective in decentralizing the activities of information processing" [10]. In the social sciences, there is extensive research on how we, humans, are part of formal (e.g., corporate, military, government) and other, difficult-to-observe informal hierarchies, such as a central person of a group of friends [11]. These can be studied as macro-, meso-, and micro-level processes, where macro-level hierarchies, for example, help us in our collective coordination and to take actions together. As Redhead and Power mention, however, not all hierarchies are explicit [11]. Expanding beyond the scope of human structures and interactions, we can also discover hierarchies in nature, e.g., in food webs [12]. This naturally opens up the question of whether we can identify them, even if they have never been formalized. Hierarchical clustering [13] presents an attempt at solving this task, which I will discuss in more detail later.

We can combine the notion of hierarchies with other types of graph structures, too, since the definition of a community lets us interpret it broadly. Some bipartite networks, such as pollination networks of plant species and their pollinators, or trade networks of countries and their exports

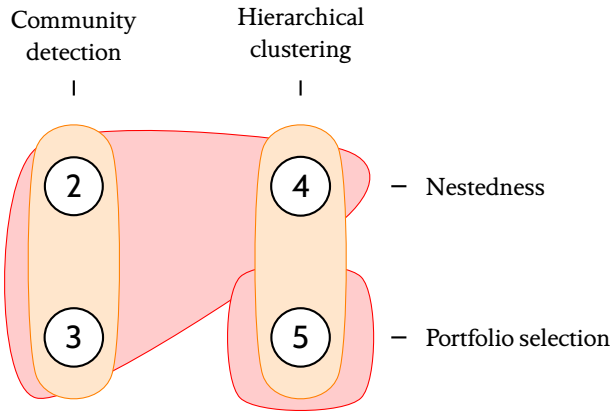


Figure 1.1: *Two possible community structures of the chapters of this thesis. The red structure marks the clustering based on the problem, while the orange one shows the grouping based on the approach (the actual structure of the thesis).*

and imports, show the presence of *nestedness* [14, 15, 16, 17]. Its definition is strict, compared to that of a community, as it is an arguably more well-defined concept. As we will see in Section 1.1, nestedness implies a highly constrained structure, and so-called fully nested graphs are quite scarce among all the possible graph configurations. Nevertheless, some degree of nestedness is still widely observed in real-world networks [16]. Its discovery originates from biogeography [18], from the spatial distribution of various species. Since then, it has been identified in a wide range of systems, including trade networks [19], ecological interaction networks [20] and interbank payment networks [21], among others. Its detection, however, contrary to traditional (overlapping) community detection, has mostly been limited to quantification, and not to localization.

In this thesis, I will present methods for detecting overlapping and hierarchical relationships between the vertices of a graph. The chapters of the work I will present here can be grouped based on either the approaches taken to solve the problems, or the problems themselves. This is illustrated in Figure 1.1. The first grouping (marked in orange) is based on approaches, which is what the actual thesis structure follows: the following two chapters cover overlapping community detection and the last two cover hierarchical clustering. The other grouping (marked in red) is based on the problem under study: the following three chapters will cover nestedness, a special graph structure that can be treated as an overlapping community structure, and the last chapter covers portfolio selection, a problem in which an investor wants to find an optimal portfolio for their investment. These groupings also illustrate the differences between overlapping community detection (the possible thesis structures for these four chapters) versus clustering (the one grouping I had to pick in the end), on a tiny scale.

This thesis is structured as follows. In the first part, I am going to introduce overlapping community detection methods for nestedness. **Chapter 2** describes a heuristic, edge-based approach for finding overlapping nested communities. **Chapter 3** introduces an algorithm that reveals all

local nestedness relationships of a network using a deterministic, node-based approach. In the second part, I will apply hierarchical clustering to solve two problems where a cluster structure directly or indirectly helps us reach a solution. In **Chapter 4**, I continue the notion of detecting nestedness by introducing multiple approaches using hierarchical clustering, aiming for a simpler representation this time. Finally, **Chapter 5** sees the application of general purpose hierarchical clustering for portfolio optimization, showing that portfolio selection models can be improved by utilizing a layered approach to clustering.

1.1 Core definitions

Throughout this thesis, when working with graphs, I will denote them by $G = (V, E)$, where V is the set of vertices, and $E = \{(i, j) \mid i, j \in V\}$ is the set of edges. When referring to the vertices or edges of a specific graph, I will also use the notation $V(G)$ to refer to the set of vertices of the graph G, and $E(G)$ to refer to its set of edges. In most cases I will be working with graphs where there are no loops, i.e., $(i, i) \notin E\ (\forall i \in V)$, unless noted. I will denote the number of vertices by $|V| = n$ and the number of edges by $|E| = m$. In the case of undirected graphs, the order of the vertices of an edge does not matter, and as such, $(i, j) \in E \Leftrightarrow (j, i) \in E$. When an additional $w : E \to \mathbb{R}$ function is given, we call the graph *weighted*. Weights can represent various concepts that can be expressed in numeric form through the links present in a graph: the distance (or closeness) of two vertices, the strength of a relationship, the amount of throughput on the road between two vertices, and much more. I will denote the neighborhood of a node i by $N(i)$, which is the set of nodes i is connected to: $N(i) = \{j : (i, j) \in E \text{ or } (j, i) \in E\}$. In the case of directed graphs, I will denote the incoming neighbors of a node i with $\text{in}(i) = \{j : (j, i) \in E\}$ and its outgoing neighbors with $\text{out}(i) = \{j : (i, j) \in E\}$.

To reiterate, in this thesis, I will make a clear difference between the problems of *(overlapping) community detection* and *(non-overlapping) clustering* – or partitioning, coloring. In the case of community detection, we are interested in finding groups of vertices that are connected by some property, where one vertex may be part of one or more groups. When performing clustering, though, each vertex will be part of exactly one group.

The formal definitions and specific notations will be introduced in the relevant chapters.

Nestedness

The community detection and hierarchical clustering results I will discuss in **Chapters 2 to 4** revolve around one special network structure: nestedness. A graph G is *fully nested* if, for any pair of vertices $i, j \in V(G)$ such that j has a higher or equal degree than i, $N(i) \subseteq N(j)$ holds [16]. In other words, the vertices of G can be ordered such that the respective neighborhoods (as sets) form a chain: $N(i_1) \subseteq N(i_2) \subseteq \cdots \subseteq N(i_n)$. In the case of bipartite graphs, i and j must be in the same (color) class. Also in bipartite graphs, if perfect or full nestedness holds for one class,

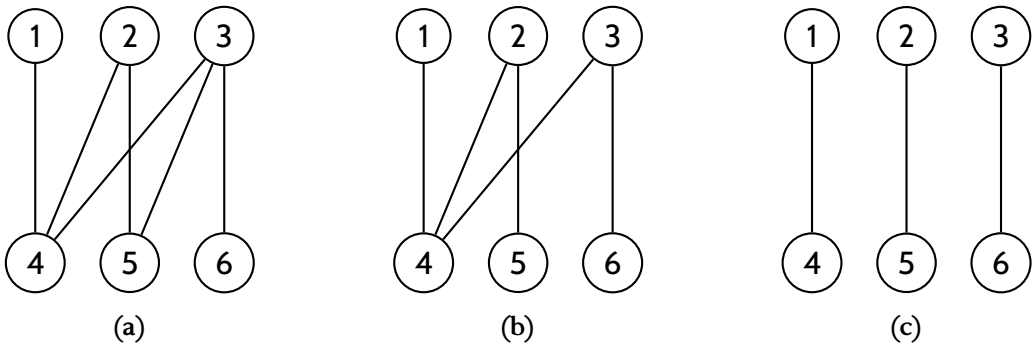


Figure 1.2: *Examples of (a) fully nested, (b) partially nested and (c) entirely non-nested graphs*

then it holds for the other as well. Having such strict requirements, the graphs we construct of real-world networks are seldom fully nested, but they might still contain fully nested subgraphs. For that reason, it is crucial to not only be able to decide whether a graph is fully nested, but also to measure *how nested* (i.e., how close to being fully nested) a graph is. Figure 1.2 illustrates our motivation to investigate. In Figure 1.2a, the definition applies to the entire graph, or more precisely, to both classes: $N(1) \subseteq N(2) \subseteq N(3)$, and similarly to nodes 4, 5, and 6, as the property is symmetric. Figure 1.2b shows a partially nested graph, where $N(1) \subseteq N(2)$ and $N(1) \subseteq N(3)$, but $N(2) \not\subseteq N(3)$. Lastly, the graph in Figure 1.2c displays no nestedness at all.

For a long time, nestedness has been a structure that was quantified, but not detected as a cluster or community structure, i.e., there was no way to identify the overlapping fully nested subgraphs of Figure 1.2b. The metrics that compute nestedness (for example, NODF [22], the binmatnest temperature [23] or discrepancy [24]) also operate on bipartite graphs, exclusively. However, while nestedness was largely observed in bipartite networks, it is not unique to them. In fact, it was shown that nestedness and its detection can be extended to non-bipartite graphs, too [25].

In order to identify local nestedness relationships, we need to define a nestedness value for a pair of vertices i and j. For that, I am going to use the following metric:

$$\mathrm{nest}(i,j) = \frac{|N(i) \cap N(j)|}{\min\{|N(i)|, |N(j)|\}}. \tag{1.1}$$

When $\mathrm{nest}(i,j) = 1, N(i) \subseteq N(j)$ or $N(j) \subseteq N(i)$ holds, otherwise both vertices have at least one neighbor to which the other is not connected. This is also called an induced $2K_2$, which is two parallel edges in a graph of four vertices[1]. In the edge case of either vertex having no neighbors (i.e., $N(i) = \emptyset$ or $N(j) = \emptyset$), we can skip calculating $\mathrm{nest}(i,j)$ altogether and assume its value is zero.

When $\mathrm{nest}(i,j) = 0$, it means that vertices i and j have no common neighbors. If the value is greater than 0, but smaller than 1, then they have common neighbors, but there is also at least

[1] Simple examples of multiple induced $2K_2$ structures can be seen on Figure 1.2c, by picking any two nodes from a class, and their neighbors, e.g., nodes 1 and 2, and their neighbors 4 and 5.

one $2K_2$ that blocks the pair of vertices from being fully nested. The metric allows us to build algorithms that can discover the local nested structure of a network, and I am going to heavily rely on it in the following chapters.

Nestedness in non-bipartite graphs

Since we are focusing on general – not just bipartite – graphs, we need to take into account the connection between a vertex pair when measuring their nestedness. This was not an issue in bipartite graphs, since, by definition, there are no edges between vertices of the same class.

If we use Equation (1.1) to measure nestedness between vertices i and j, and there exists an edge $(i, j) \in E(G)$, the two vertices will never be considered fully nested, because $i \in N(j)$ and $j \in N(i)$, but $i \notin N(i)$ and $j \notin N(j)$. Thus, $\mathrm{nest}(i, j) < 1$ when $(i, j) \in E(G)$. This would also mean that K_n ($n \geq 3$), the complete graph of n vertices, is not fully nested.

The approach we are going to follow when comparing two vertices is to ignore the edge between them, if there exists one. Thus, we may use the following equation instead of Equation (1.1):

$$\mathrm{nest}(i, j) = \frac{|(N(i) \setminus \{j\}) \cap (N(j) \setminus \{i\})|}{\min\{|N(i) \setminus \{j\}|, |N(j) \setminus \{i\}|\}}. \tag{1.2}$$

Note that considering the existence of edges between nodes when searching for fully nested parts depends on the application. If we are looking for fully nested bipartite subgraphs in graphs that are not necessarily bipartite themselves (e.g., as in [25, 26]), the fact that two nodes can be connected should not be ignored.

Hierarchical clustering

In Part II, I am going to take on similar problems – which are, at their core, community detection tasks, – but instead of overlapping communities, I will concentrate on finding hierarchical cluster structures. I have previously argued about the potential advantages and disadvantages of finding (disjoint) clusters as opposed to (overlapping) communities. Among these methods, hierarchical clustering lies in between the two tasks, encoding some overlap between the groups. It builds n *levels* of clustering (with the i-th level denoted by ℓ_i), where each level consists of $n - i + 1$ clusters and is a valid clustering of the vertices on its own. The method becomes hierarchical with the connection between the adjacent levels: the level ℓ_i is mostly the same as ℓ_{i+1}, with the exception that two clusters from ℓ_i have been merged into one in ℓ_{i+1} (or the other way around: a cluster in ℓ_{i+1} has been split into two in ℓ_i). The two main approaches to building this hierarchy are called bottom-up (also known as agglomerative) and top-down (or divisive). Bottom-up algorithms start with all vertices in their own clusters (at level ℓ_1), and iteratively merge two clusters in each step, until reaching ℓ_n, where all vertices are in the same cluster. Top-down algorithms, on the other hand, start at ℓ_n, with all vertices in the same cluster, and split one of the clusters into two at each step, until they reach ℓ_1, where all vertices are in different clusters.

The two approaches require different algorithmic concepts. For the bottom-up approach, I will use the algorithm implemented in R [27]. It uses a pairwise distance matrix $\mathbf{D}$ of the nodes and a function (often called a *linkage method*) that calculates the distance between two clusters. The most common linkage methods calculate the minimum (single-linkage), maximum (complete-linkage) and average (average-linkage) of the pairwise distances between two clusters as their distance:

$$d_{\mathrm{SL}}(C_1, C_2) = \min_{i \in C_1, j \in C_2} D_{ij} \tag{1.3}$$

$$d_{\mathrm{CL}}(C_1, C_2) = \max_{i \in C_1, j \in C_2} D_{ij} \tag{1.4}$$

$$d_{\mathrm{AL}}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} \frac{D_{ij}}{|C_1| \cdot |C_2|}. \tag{1.5}$$

For a top-down (divisive) method we need a different approach, though, as these distance metrics are only effective because the clusters whose distances are being calculated are known from the previous step. One possible way to do it is using the algorithm of Newman and Girvan [28], which uses a graph-based approach and removes edges from a network until the number of components in it increases. This is repeated until each node is in its own component. I discuss both algorithms in detail (and adapt them to a specific use) in Chapter 4.

1.2 Contributions

The ideas, figures, tables and results included in this thesis were published in scientific papers (listed at the end of the thesis). In a nutshell, the author is responsible for the following contributions:

Chapter 2.: Introduction of an edge-based community detection heuristic for nestedness. The introduced algorithm is adjustable, as it allows changing the minimum number of times an edge is re-evaluated, allowing the user to optimize the trade-off between runtime efficiency and the precision of detected communities. The heuristic is evaluated on a range of random and real-world bipartite networks. The algorithm detects a larger fraction of nested communities in its early steps. The community sizes detected by the algorithm show a high correlation with the discrepancy nestedness metric.

Chapter 3.: Introduction of an algorithm that can detect the overlapping nested community structure of networks, alongside an algorithm that can generate a bipartite network with a given nestedness structure. Analysis of the performance of the detection algorithm on various bipartite and non-bipartite networks, along with how the community structure revealed by the algorithm compares to other overlapping community detection methods. New metrics derived from the

results of the presented algorithm to measure the global nestedness of a network.

Chapter 4.: Extension of two hierarchical clustering algorithms (one agglomerative and one divisive) specifically to reveal nested clusters of networks. Analysis of the resulting hierarchical clustering structures on a wide variety of bipartite and non-bipartite networks. Comparison of various linkage methods, showing that average-linkage and complete-linkage agglomerative clustering can produce the largest fully nested clusters. Introduction of metrics to compare clustering results across algorithms.

Chapter 5.: Application of hierarchical clustering to improve the performance of the Markowitz portfolio selection model and comparison with hierarchical clustering-based portfolio selection methods. Extensive analysis of the results using real-world stock market data, using various metrics to measure portfolio performance. Our results show that the filtering methods were effective in reducing errors in the risk estimation, while the hierarchical clustering-based models offered higher returns, but also higher risks.

Part I

Nestedness as an overlapping community structure

Chapter 2

Heuristic for edge-based nested community detection

As seen in Chapter 1, nestedness has mostly been considered a global property, not a local one. In this and the following two chapters, I will present methods for discovering local nested relationships between groups of nodes. Various approaches have been developed in the literature to find local community structures, but they identify communities in the traditional sense: groups of nodes that have more connections inside the group than outside the group. In this part, I will show that nestedness, too, can be interpreted as an overlapping community structure, by providing algorithms that find these overlapping subgraphs.

First, in this chapter, I will introduce a heuristic algorithm for finding overlapping nested communities using the edges of the graph. While heuristic algorithms often guarantee only approximations rather than full solutions, they have the benefit of being performant, and the trade-off between performance and accuracy is often adjustable. The algorithm presented in this chapter is also adjustable, allowing to trade the size of the nested communities for the number of iterations performed.

2.1 Introduction

Although nestedness in networks has been quantified by several metrics [16, 24, 29, 30, 31], the problem of finding special clusters or communities having the property of nestedness has not been investigated yet from an algorithmic perspective. In the literature one can only find a few papers dealing with related problems. For instance, Junttila and Kaski [26] call a binary matrix – that is a matrix whose entries are either zero or one – *fully nested* if its rows and columns can be reordered such that the ones are in an echelon form. They call a binary matrix A fully k-nested if its columns can be partitioned into k pairwise disjoint submatrices, called blocks, each of which is fully-nested. Given a matrix A an obvious optimization problem is to find the smallest k such that A is fully

k-nested by also providing a partitioning into k fully-nested parts. They provided several heuristic algorithms for discovering k-nestedness. Since general $m \times n$ binary matrices can be considered as bipartite graphs, such methods provide nested clustering in this case.

For a graph G and a fixed bipartite graph H, London, Pluhár and Martin [25] defined the concept of *induced H-avoiding coloring* – meaning that the union of any two color classes spans an induced H-free graph – and defined $\chi_H(G)$ as the minimum number of colors in an induced H-avoiding coloring of G. In case of $H = 2K_2$ this coloring realizes a partitioning to bipartite, fully-nested clusters. Their approach and the one we present in this chapter are both applicable to general graphs as well.

Moving forward in this direction, here we present a heuristic algorithm that identifies nested communities of a network using a special edge labeling approach. In contrast to existing approaches, our method can handle general graphs, not only bipartite ones, and is able to find overlapping subgraphs with nested neighborhood structures as well. We also present multiple case studies to investigate the performance of the algorithm on synthetic and real networks. The chapter is organized as follows. In Section 2.2, we provide the most important definitions and concepts we are dealing with. In Section 2.3, we present and discuss in detail our heuristic algorithm. In Section 2.4, we present our case studies, and finally, in Section 2.5, we draw some conclusions and mention some potential research questions for the future.

2.2 Preliminaries

2.2.1 Nestedness in networks

The definition of nestedness can be extended to general graphs in various ways. In this chapter, we define a *fully nested subgraph* of a graph G as an induced $2K_2$-free graph [25] and try to identify these subgraphs as communities of G. Note, that in the case of bipartite (sub)graphs the $2K_2$-free property immediately guarantees nestedness. However, directly enumerating nested bipartite subgraphs (as communities) may prove too restrictive in real applications, moreover it leads to a counting problem that may not even be feasible. We handle this using the following formalism.

To test whether two vertices $v_i, v_j \in V$ are nested, i.e., one's neighborhood contains the other's, even in a unipartite graph, we use the $\mathrm{nest}(i, j)$ value from Equation (1.1):

$$\mathrm{nest}(v_i, v_j) = \frac{|N_i \cap N_j|}{\min\{|N_i|, |N_j|\}}.$$

In order to identify nested subgraphs, we group together those v_i and v_j nodes that have $\mathrm{nest}(v_i, v_j) = 1$, as seen on Figure 2.1, where $\mathrm{nest}(1, 2) = \mathrm{nest}(1, 3) = 1$, but $\mathrm{nest}(2, 3) < 1$. We should emphasize that in this work we look at a subgraph as a vertex set spanned by certain edges. A fully nested subgraph needs to satisfy the nested condition among all of its node pairs. If we restrict Equation (1.1) to the case when $(v_i, v_j) \notin E$, or use Equation (1.2), we obtain nested

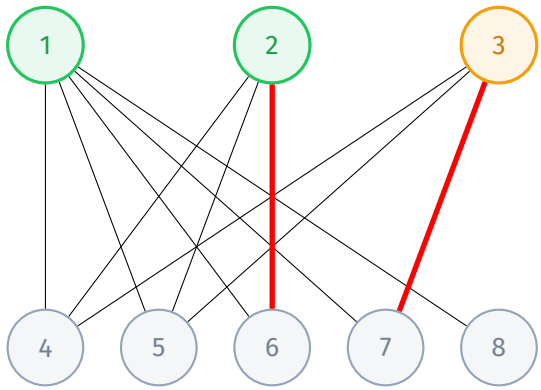


Figure 2.1: *Example bipartite graph with an induced $2K_2$ highlighted in red. The top half is colored based on one possible clustering of the nodes.*

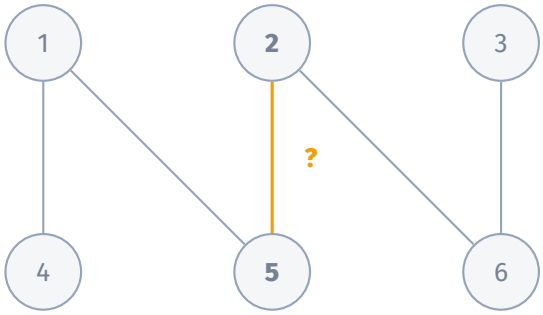


Figure 2.2: *Example graph where the placement of the edge $(2, 5)$ is ambiguous during nested clustering.*

subgraphs (as communities). On the other hand, if we do not apply this restriction, the $2K_2$-free property still holds for the edge structure of the identified communities, that are not necessarily bipartite.

2.2.2 Community detection

Our goal is to identify the maximal nested subgraphs of a network. This can be done both through clustering and community detection – the key difference being here that we allow overlapping groups (communities) in the latter. Figure 2.1 demonstrates this concept at node level. We can see that node 1 has only common neighbors with nodes 2 and 3, but nodes 2 and 3 have a $2K_2$ between themselves, thus they cannot be in the same group.

In contrast, Figure 2.2 demonstrates the concept at the level of edges. Looking at the nodes of the top row, if we perform clustering based on nestedness, we get clusters $\{\{1\}, \{2, 3\}\}$. If we take a closer look at the graph's edges instead of nodes, though, we can see that the subgraphs induced by edges $\{(1, 4), (1, 5), (2, 5)\}$ and $\{(3, 6), (2, 6), (2, 5)\}$ are both nested. Notice how edge $(2, 5)$ is part of two nested subgraphs at the same time. When performing clustering, we would have to

put $(2, 5)$ either into the first or the second nested subgraph, when in reality it is equally part of both.

To perform the aforementioned grouping, labels are assigned to nodes throughout the process. A labeling is a function $C : V \to \mathcal{P}(\mathbb{N})$ where $C(v)$ is the set of labels assigned to node v, referring to the groups (communities) v belongs to. In the case of clustering, only a single label is assigned to each node (in this case $C : V \to \mathbb{N}$), meaning that one node can belong to only one group, which creates a non-overlapping cluster structure. In contrast to clustering, when performing community detection we allow one node to be assigned to multiple groups.

Traditionally, these methods assign labels to vertices, but we can also do that for edges. To do so, we define the function $\tilde{C} : E \to \mathcal{P}(\mathbb{N})$ to denote the set of labels assigned to an edge. The set of communities that a vertex v belongs to is defined as the union of the communities of v's incident edges:

$$C(v) = \bigcup_{e \in E(v)} \tilde{C}(e), \tag{2.1}$$

where $E(v)$ denotes the set of edges incident to v.

2.3 Edge-based nested community detection

First, we note again that we refer to subgraphs as graphs induced by certain edges. Our algorithm labels edges, with one or more label, in a way that each subgraph induced by the edges having the same label (i.e., belonging to the same community) is fully nested. The key idea is that we use Equation (1.1) to check whether two vertices are nested and we assign *all* of their incident edges to the same community or to two different communities (one for one vertex's edges and an other one for the another's edges) based on the result.

Naively we may compare all $\binom{n}{2} \approx n^2$ possible vertex pairs, but we can safely avoid many unnecessary comparisons. For instance, most obviously, there is no reason to compare two vertices $u, v \in V$ not having a common neighbor (i.e., where $\mathrm{nest}(u, v) = 0$), since they are always part of an induced $2K_2$ (if none of them is an isolated vertex).

Instead of excluding these pairs, though, we focus on collecting pairs of candidates that have at least one common neighbor. For each vertex v we take the set of vertices that are exactly at distance two from v, that is:

$$N^{++}(v) = \{v_j : \exists v_k \, (v, v_k), (v_k, v_j) \in E, (v, v_j) \notin E\}. \tag{2.2}$$

To keep track of which vertices we need to examine, we use two lists n_1 and n_2. For each vertex n_1 contains the number of unlabeled incident edges (or unseen neighbors) and n_2 contains the unseen 2-neighbors. During the process we denote the two nodes in comparison with a and b. We use c_a and c_b to denote the new communities that will be assigned to nodes a and b (based on

Listing 2.1: The algorithm's main routine. Lines 8–9 filter the indicated sets twice, e.g. we get a by first selecting nodes with appropriate n_1 and n_2 values, then choosing one with a maximal n_2 value.

```
chk ← ∅                                                                  1
maxcomm ← 0                                                              2
for v ∈ V:                                                               3
    n1(v) ← |N(v)|                                                       4
    n2(v) ← |N++(v)|                                                     5

while ∃v : n1(v) ≥ T ∧ n2(v) ≥ 1:                                        7
    a ← V[n2 ≥ 1 & n1 ≥ T][n2 = max(n2)]                                 8
    b ← N++(a)[(a, b) ∉ chk][n1 = max(n1)]                               9

    if b = null:                                                         11
        // see 2.3.1                                                     12
        ca ← { maxcomm + 1 }                                             13
    else if nest(a, b) = 1:                                              14
        // see 2.3.2                                                     15
        ca, cb ← comm_same(a, b)                                         16
    else:                                                                17
        // see 2.3.3                                                     18
        ca, cb ← comm_diff(a, b)                                         19

    C̃(E(a)) ← C̃(E(a)) ∪ ca                                              21
    C̃(E(b)) ← C̃(E(b)) ∪ cb                                              22

    n1(a) ← n1(b) ← 0                                                    24
    n2(a) ← n2(a) - 1                                                    25
    n2(b) ← n2(b) - 1                                                    26

    maxcomm ← max(maxcomm ∪ ca ∪ cb)                                     28
    chk = chk ∪ {(a, b)}                                                 29
```

the new labels of the incident edges), respectively. The pseudocode of the algorithm is given in Listing 2.1.

In the following subsections we take a look at the key parts of the algorithm in more detail.

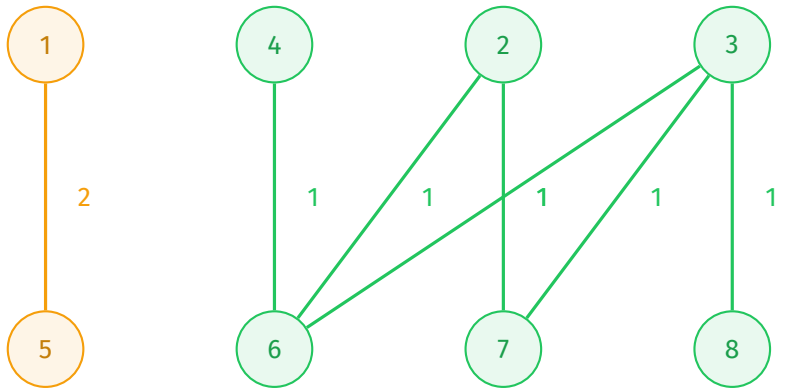


Figure 2.3: *Example of a disjoint graph where node "1" does not have a 2-neighbor.*

2.3.1 Choosing a suitable b

There is one case where we cannot select a suitable b. That is when a does not have any 2-neighbors, i.e., $N^{++}(a) = \emptyset$. In this case, we assign a yet unused community index to the edges incident to a. Such an example is demonstrated in Figure 2.3.

2.3.2 Assigning edges to the same communities

Listing 2.2: The comm_same function

```
comms ← C^{++}(a, b)                                    1
for c in comms:                                          2
    for v in {v : c ∈ C(v)}:                             3
        if nest(a, v) = nest(b, v) = 1:                  4
            continue                                     5
        else:                                            6
            comms ← comms \ c                            7

if comms = ∅:                                            9
    comms ← { maxcomm + 1}                               10

c_a, c_b ← comms                                         12
```

In this case we have to find suitable communities that both a's and b's edges can belong to. Note that the addition of these edges to communities must not create a $2K_2$, so we add the edges of a and b to all communities in which they do not create a $2K_2$. If there is no such community, we create a new one and assign the edges to that.

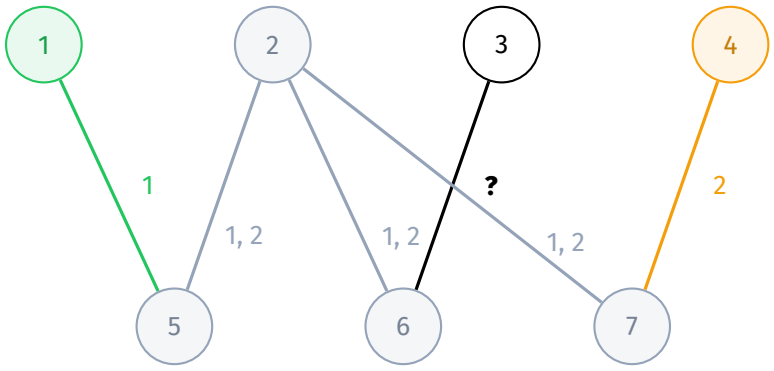


Figure 2.4: *Graph with three nested communities, the third one not yet found. The algorithm needs to decide what communities can be assigned to the edge labeled "?".*

In order to determine the common communities of a and b, first we find all already used communities on the edges of the 2-neighbors of a and b, that is:

$$C^{++}(a,b) = \bigcup_{v \in N^{++}(a) \cup N^{++}(b)} C(v). \tag{2.3}$$

Then for all communities $c \in C^{++}(a,b)$ we check whether a and b are nested with all of the vertices in community c. If there is a $2K_2$ on a or b, and a vertex of a community c, then c is removed from the set of considered communities.

If the set of the remaining communities becomes empty, we assign a new community to the edges of a and b, otherwise we assign all the remaining communities to their edges. This procedure is shown in Listing 2.2.

Figure 2.4 demonstrates this procedure as follows. We need to find what communities we can assign to the bold edge labeled with "?". We use $a = 2$ and $b = 3$ (and this is the only possible setup where node 3 is present, since node 3's only 2-neighbor is node 2). We take the communities they or their 2-neighbors are part of and iterate through them, that is $C^{++}(2,3) = \{1,2\}$. Community 1 cannot be assigned to the unlabeled edge because it would create a $2K_2$ with node 1 ($\mathrm{nest}(1,3) = 0$). Likewise, community 2 also cannot be assigned to the edge because it would create a $2K_2$ with node 4 ($\mathrm{nest}(4,3) = 0$). Since all communities were excluded, our only possible action is to assign a new community to both node 2 and 3's edges, and that will be community 3. This means that node 2's edges will be assigned to the communities $\{1,2,3\}$ and node 3's edge will get the label $\{3\}$.

2.3.3 Assigning edges to different communities

This case is much simpler. We need to ensure that at least one of $C(a) \setminus C(b)$ or $C(b) \setminus C(a)$ is non-empty, meaning that at least one of the edges of a or b is assigned to a community that contains

no edge of the other node. This can be done by assigning a new community to a's edges and another new community to b's edges. If all the edges of a node are assigned to a community, we can skip creating a new community. The procedure, repeated for both $v = a$ and $v = b$, is the following. If v has an edge that is not assigned to a community, then assign all of v's edges to a new community.

We do not need to check if a and b have common communities only, because a and b have not yet been compared. Because of this, their edges were either not assigned to communities yet or were assigned to different communities. The only case where they could have been assigned to the same community is when they have been compared and $\mathrm{nest}(a, b) = 1$ was true.

2.3.4 Removing excessive communities

Due to the previous case (Section 2.3.3) the algorithm sometimes introduces communities that are subsets of other communities. After the algorithm finishes, the procedure for removing them is to examine all pairs of communities, and if one community is a subset of the other, then it is removed.

2.3.5 Remarks

Although we used bipartite graphs to demonstrate how the algorithm works, we should emphasize that it does not exploit any properties exclusive to bipartite graphs, which means it can be used on general graphs as well, without any modifications.

The algorithm's computation time is strongly influenced by the iteration's adjustable stop condition, which is $n_1 \geq T \wedge n_2 \geq 1$, where T is the adjustable threshold parameter. There are three noteworthy cases for the threshold value:

a) $T = 1$. In this case the algorithm labels all edges at least once. Nodes are compared at most once.

b) $T = 0$. All nodes are checked again if their edges were labeled only once.

c) $T = -\infty$. All $\frac{1}{2}\sum_{v \in V} n_2(v)$ combinations of (a, b) pairs are checked ($a \in V, b \in N^{++}(a)$).

Further fine-tuning can be achieved by setting T to a negative value. Lower numbers will result in more iterations but also more complete nested communities. We discuss the effects of the threshold parameter in more detail in Section 2.4.3.

An R [27] implementation of the algorithm is also open-source[1].

[1] The implementation is available at `https://github.com/Hanziness/r-nested-comms/tree/v0.1`

2.4 Case studies

In the following subsections, we examine the output of the algorithm in more detail on real-world networks and then also compare the results with multiple stop conditions.

2.4.1 Host-parasite networks

We compare the community structures found in 50 host-parasite interaction networks from Web of Life [32, 33] and highlight the differences between the least and most nested networks. To be able to compare networks with different degrees of nestedness and measure how nested a network is, we use the discrepancy measure [24], which returns the number of 1s in the incidence matrix that must be shifted to get a fully nested matrix. We also separated the networks into components and then removed components with less than 5 nodes, so only the larger, connected networks were considered.

First, we can identify a highly nested and a less nested network in the data set: the discrepancy of `A_HP_017` is 1, while for `A_HP_031` we get 62 – here a lower number means higher nestedness. In the first case, our algorithm found two communities, each spanning 96% of the edges, while in the second case, it identified 79 communities, each containing 16.3% of the edges on average. In the first network, there was only one $2K_2$ that resulted in the algorithm creating two communities. Consistently the discrepancy value of this network is 1.

We then calculated the number of communities, and the size of the communities both edge-wise and node-wise. Looking at Figure 2.5 we can see that there is a non-linear relationship between the discrepancy and the average number of edges in the communities. The Spearman correlation coefficient of the two variables was -0.9374, the Pearson correlation of the community size and the logarithmic discrepancy was -0.8794.

2.4.2 Random networks

We generated 200 samples of random bipartite graphs using the $G(n, p)$ Erdős-Rényi model [34], with vertex counts ranging from 6 to 30 on each side of the bipartite graph and $p = 0.6$. We observed a similar relationship between discrepancy and the average number of edges in communities as seen in Section 2.4.1. Figure 2.6 depicts the community sizes in the function of discrepancy. We can see that highly nested networks, in the sense of discrepancy, show denser nested communities. The Spearman correlation coefficient of the two variables was -0.9876.

In this case, the node and edge count showed a 0.9979 and 0.9954 correlation coefficient with the discrepancy, higher than the values of 0.9485 and 0.8614 in the case of the host-parasite networks.

One, much more glaring difference was the number of star communities. In the random graphs, the average ratio of star communities – where a node is in its own community with its neighbors –

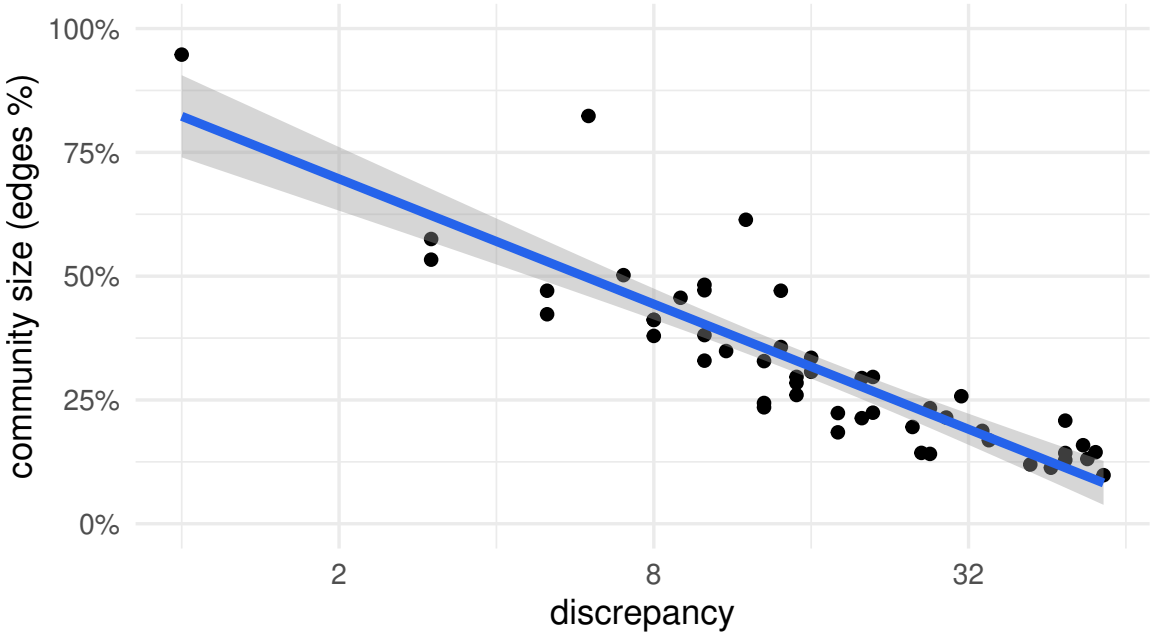


Figure 2.5: *Average number of edges in a community as the function of logarithmic discrepancy on the host-parasite network data set, with $T = -\infty$. The regression line is drawn along with the 95% confidence interval.*

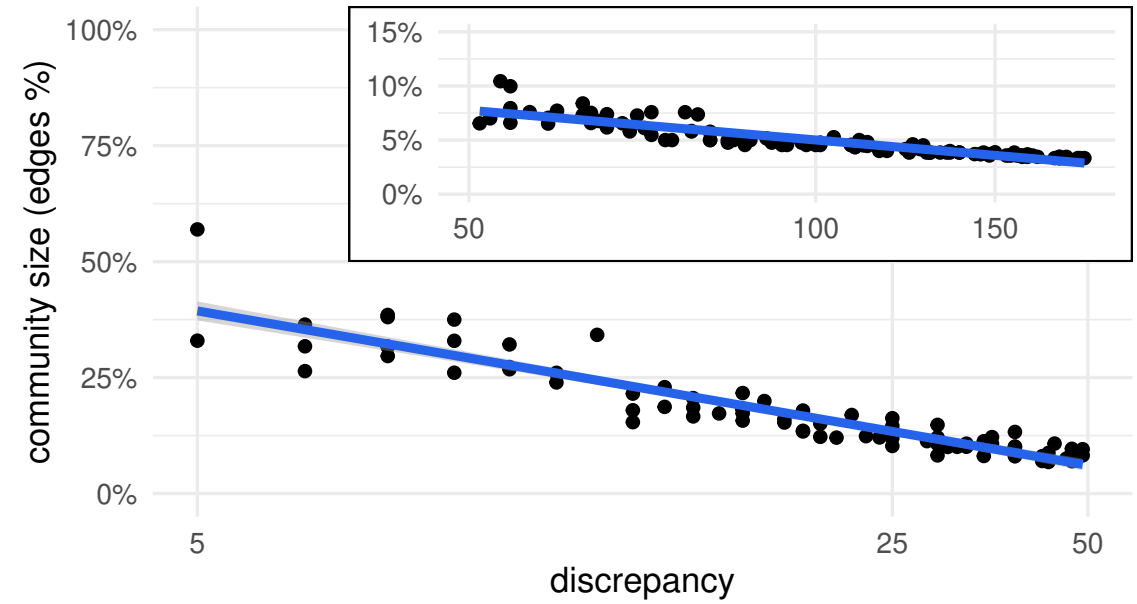


Figure 2.6: *Average community sizes in the function of discrepancy on random Erdős-Rényi bipartite graphs. The fit line is using the function* $\log(\log(x))$.

was 67.89%, while in the host-parasite networks it was only 12.78%. Such communities are direct results of the case described in Section 2.3.3. Here, that means that most of the compared nodes were not nested.

2.4.3 Comparison of parameters

Since the algorithm's stop condition can be adjusted, it is worth examining its effect on run time and the communities the algorithm detects. We compared the algorithm with n_1 thresholds $T = 1, 0, -1, ..., -7$ and $T = -\infty$. Again, setting the threshold to $T = -\infty$ results in that all 2-neighbor pairs being examined. We performed these comparisons on the previously introduced host-parasite networks.

As Figure 2.7 shows, by decreasing the n_1 threshold value (which results in more iterations) the average number of edges in a community increases. This means that the algorithm tends to grow existing communities more (i.e., making them denser) than create new ones. We can also see that while the number of iterations steadily increases, the growth of the community size starts to slow, requiring a lot more iterations in the end (at $n_1 = -\infty$) to reach the remaining 7% growth.

The threshold parameter on n_1 can still be useful, though. On some smaller graphs, the community sizes were very close to each other when using a finite threshold or the infinite one, but the algorithm performed much fewer iterations in the finite case. This means that the parameter can be used to strike a balance between a quick result of smaller and a slower result of denser communities.

2.5 Conclusions

We described an algorithm that is capable of unveiling the nested community structure on the edges of a graph using an iterative approach. The approach allows us to assign edges and nodes to multiple communities. The algorithm is highly adjustable: its main parameter allows one to reduce the number of performed iterations in exchange for a smaller, sparser community structure. While nestedness is mostly relevant in bipartite networks, the algorithm does not exploit properties exclusive to bipartite networks, and it can be applied to general networks as well.

We performed a case study on a set of bipartite host-parasite networks and found that the algorithm detects a larger fraction of nested structures in its early iterations. In our experiments, the average community size (the number of edges in a community) showed correlation with the discrepancy nestedness metric. We also examined the effect of the n_1 threshold parameter and found that it allows finer control over the speed of the algorithm and the detected community structure.

In the future, we would like to optimize the speed of the algorithm, possibly by introducing a better ordering of the examined nodes to reduce the need for post-processing. We also plan to

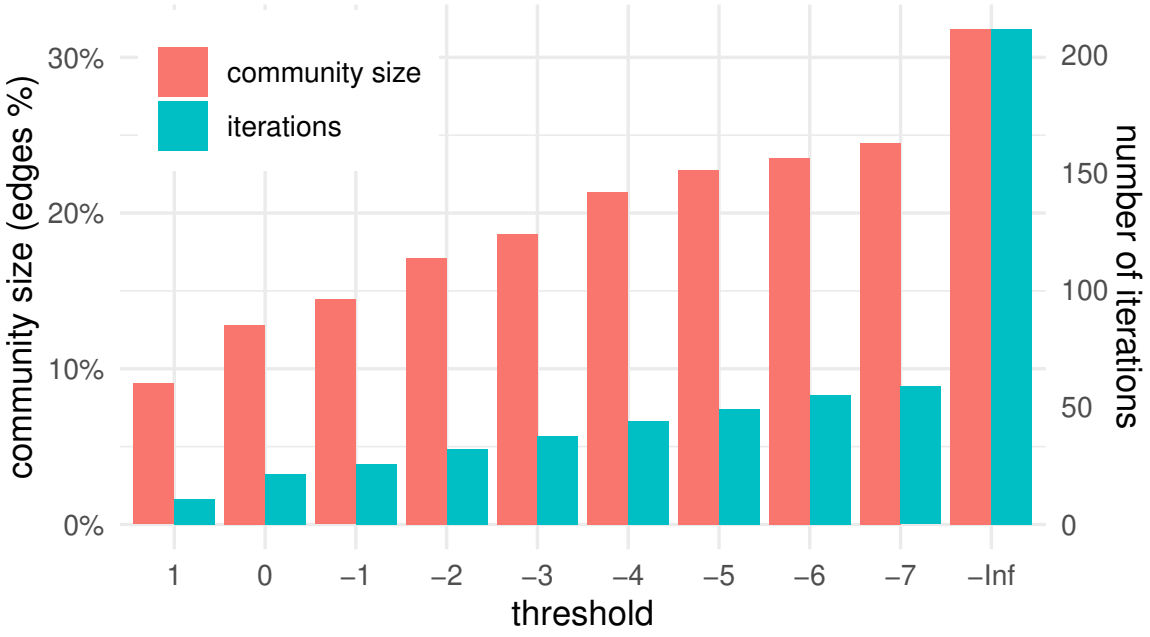


Figure 2.7: *Average community size and number of iterations performed for each n_1 threshold value on the host-parasite networks.*

further investigate the completeness of the algorithm to see whether it finds all maximal $2K_2$-free communities.

Chapter 3

Detecting nested communities

In this chapter, I will present an algorithm that addresses the drawbacks of using a heuristic for detecting overlapping nested communities. As we have seen in the previous chapter, heuristics have their own set of advantages, such as parameters that allow us to fine-tune their focus on performance versus accuracy. However, we sometimes want to see solutions that are guaranteed to contain all communities. The algorithm introduced in Chapter 2 does not have such a guarantee, so in this chapter, I will look for a different approach. The key idea here is to build an auxiliary nestedness graph, remove the redundancy and extract the communities from it. This auxiliary graph allows us not only to extract the nested community structure, but also to identify the role of each node in the network in terms of nestedness. The motivation for developing such a method was to overcome the potential inaccuracy of the previous heuristic approach, and to return to vertex-based community detection, since graphs usually have much fewer vertices than edges.

3.1 Introduction

The standard community detection methods found in the literature try to achieve lots of edges within clusters (or communities) and only a few between distinct clusters [35], imitating concepts of data clustering in statistics and machine learning [36]. In general, this approach works well and provides meaningful clusters for social networks due to some of their widely observed common properties. For instance, the number of triangles in a social network is much larger than in a random graph with similar edge density, they often show heterogeneous degree distribution and have small diameters [37]. Taking these properties into account helps to find the dense parts of the network. Many algorithms rely on maximizing the modularity function, which measures the quality of a given clustering [28], but there are lots of different approaches. While for clustering, where each node is assigned to exactly one cluster, both bottom-up and top-down type algorithms have been proposed, for community detection, where a node can be a member of several communities, mostly bottom-up algorithms are used, i.e., smaller initial communities are expanded with new

nodes during the community detection process [38]. On the other hand, some social networks and especially technological or transaction networks generally contain fewer triangles and often have tree-like structures [39]. Therefore, trying to find disjoint dense parts is inadequate in principle.

Bipartite interaction networks in ecology often display a nested structure in which specialist species interact with generalists, while generalists interact with each other and with specialists, too [40]. Networks that show the presence of nestedness are not necessarily bipartite, however, and the definition of nestedness can be extended to non-bipartite networks, as we have seen in Section 1.1 and Chapter 2 of the thesis. It is also an overlapping structure, so it is desirable to find all fully nested subgraphs in a network. Here, we present a novel method that identifies overlapping nested subgraphs and represents them as paths of a directed graph we call community graph. We also introduce a method that generates bipartite graphs with (any) ground-truth overlapping nested structure, making it possible to generate example nested graphs and test nested community detection algorithms. Since our algorithm detects nestedness in non-bipartite graphs, too, in order to be able to quantify nestedness in any graph, we derive a new metric from the output of our algorithm called *vertex presence*. To measure the "generalist-ness" of a vertex, we derive another metric called *vertex position*.

The rest of the chapter is organized as follows. First, we introduce the core definitions we are going to use. In Section 3.2, we introduce an algorithm for detecting overlapping fully nested subgraphs of an arbitrary input graph. We represent the resulting nested community structure with a *community graph* that encodes additional information about the hierarchy and relationship of the nested subgraphs in the network. Then, in Section 3.3, we introduce an algorithm that can generate a class of bipartite graphs that exhibits the nested community structure of the input community graph. This algorithm may also be suitable for testing nestedness metrics. In Section 3.4, we introduce two new metrics to measure nestedness on both node and graph level, and test our nested community detection algorithm on typical nested and non-nested artificial and real-world graphs. The artificial graphs are either generated using the algorithm introduced in Section 3.3, or bipartite nestedness benchmark networks and non-bipartite community detection benchmark networks are utilized. Finally, in Section 3.5, we summarize.

3.1.1 Nestedness

Rephrasing the definition in Section 1.1, a graph G is *fully nested* if, for any pair of vertices $i, j \in V(G)$ such that j has a higher or equal degree than i, $N(i) \subseteq N(j)$ holds [16]. In other words, the vertices of G can be ordered such that the respective neighborhoods (as sets) form a chain. In case of bipartite graphs, the two compared vertices must be in the same (color) class. If perfect or full nestedness holds for one class, then it holds for the other. Figure 3.1a shows a fully nested graph, while Figure 3.1b depicts one that is not fully nested. Observe that the presence of an induced $2K_2$, that is two independent edges (formed by the edges $(2, 5), (3, 6)$ in Figure 3.1b, colored in red), is responsible for breaking full nestedness. We want to emphasize the use of *not fully nested* instead

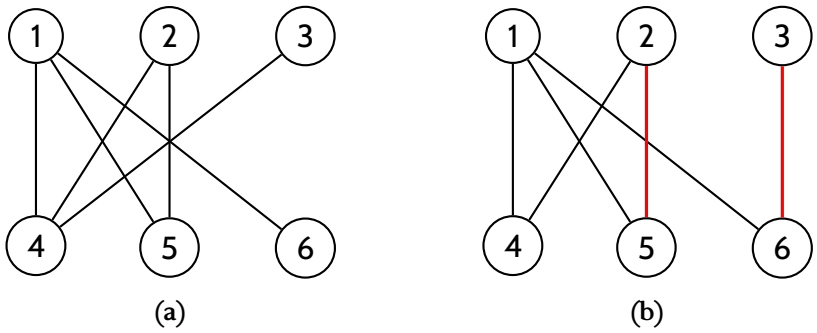


Figure 3.1: *Examples of (a) fully nested and (b) partially nested bipartite graphs. The $2K_2$ breaking nestedness is highlighted in red.*

of *not nested* as graphs may not be fully nested themselves, but may have fully nested *subgraphs*. In other words, the definition of nestedness may not hold for the entire graph, but it might be true for one or more subsets of vertices.

It is important to distinguish between the notions of nested *vertices* and nested *graphs*. As a building block to define nestedness of graphs, we first define nestedness of a pair of vertices. The amount (or strength) of nestedness between two vertices can be defined as

$$\text{nest}(i,j) = \frac{|N(i) \cap N(j)|}{\min\{|N(i)|, |N(j)|\}}.$$

While Equation (1.1) is not ideal for measuring the nestedness of the whole graph (it would require $\approx n^2$ calculations to get an average nestedness value, for example), we can use it to find groups of vertices that form perfectly nested subgraphs – which is our ultimate goal.

3.1.2 Existing methods

Clustering to nested parts

One way to find fully nested subgraphs is to use the incidence matrix to identify submatrices of echelon form [26] – this, however, works for bipartite graphs only. Another possibility is to use Equation (1.1) and assign vertices to a group where the pairwise nestedness values are equal to 1. This can be done by, for example, performing a $2K_2$-free coloring on the graph [25].

All of these methods exhibit the same problem, though. The graph in Figure 3.1b contains two nested subgraphs: one with the vertices $\{1, 2\}$ and another with $\{1, 3\}$. Notice that vertex 1 is present in both fully nested subgraphs, but we can only assign that vertex to a single group in the clustering task. This would create two ambiguous cluster structures: both $\{\{1, 2\}, \{3\}\}$ and $\{\{1, 3\}, \{2\}\}$ are valid clusterings of the same graph. Although this is not necessarily a problem as one will often search for a single clustering, the resulting structure does not encode such overlaps,

potentially losing valuable information. We further explore the concept of (hierarchical) clustering in Chapter 4.

3.2 Nestedness and community detection

In this section, we will present an algorithm that retrieves the nested community structure of the input (bipartite or general) graph. First, we will talk about how the order of nodes inside nested community structures allows us to store more information compared to traditional communities. We use this additional information to construct a so-called *community graph*. Then, we introduce an algorithm that reconstructs not only the nested communities, but the entire community graph from an arbitrary input graph.

While in this work we frequently mention *community detection*, we refer to it as a framework for detecting *overlapping* groups (communities) of vertices that are, in some sense, similar to vertices within the group, while, in the same sense, different to vertices in other groups. Traditionally, this similarity meant vertices being densely connected within a group, while vertices across different groups were less connected. Here, we are looking for overlapping groups (communities) of vertices that form fully nested subgraphs instead of being densely connected. We will call these *nested communities*.

3.2.1 Nested hierarchy from directed graphs

Using Equation (1.2) we are able to decide whether two vertices are nested, but the direction of nestedness, i.e., which vertex's neighborhood is a subset of the other, is not considered. This is important because we will use this information to determine the hierarchy of vertices. Knowing the direction of pairwise nestedness, we can use it to create a graph representation that encodes the nested relationships of the entire graph.

To do this, we construct a directed graph, where a directed edge $i \to j$ means $N(i) \subseteq N(j)$. We will refer to this graph as the *community graph*. Before we proceed, we first verify some basic scenarios from Figure 3.2.

1. If we have edges $i \to j$ and $j \to k$ (as in Figure 3.2a), we get $N(i) \subseteq N(j)$ and $N(j) \subseteq N(k)$. Nestedness is transitive, so this also means $N(i) \subseteq N(k)$. For simplicity, we omit these edges from our community graphs, or equivalently, we work with the *transitive reduction* of the community graph (see Section 3.2.2 for more details).

2. A node can have multiple out-neighbors (Figure 3.2b). If we have edges $i \to j$ and $i \to k$, then we get $N(i) \subseteq N(j)$ and $N(i) \subseteq N(k)$. This can be solved by letting both j and k have all the neighbors of i, but also making sure j and k each have at least one other neighbor the other does not.

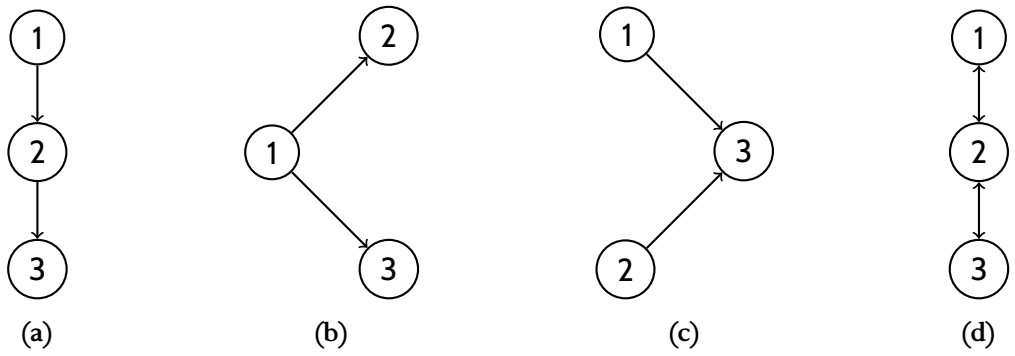


Figure 3.2: *Cases of nested relations in a community graph. From left to right: a fully nested graph (a), a node nested with different nodes (b), multiple nodes nested with the same node (c), and nodes having equal neighborhoods (d).*

3. A node can also have multiple in-neighbors (Figure 3.2c). Here we have edges $i \to k$ and $j \to k$ and get $N(i) \subseteq N(k)$ and $N(j) \subseteq N(k)$. This case can be solved by taking the union of the neighbors of i and j to create the neighborhood of k ($N(k) \supseteq N(i) \cup N(j)$).

4. Finally, nodes may have the exact same neighbors as other nodes (Figure 3.2d). This results in edges $i \leftrightarrow j$, and thus in both $N(i) \subseteq N(j)$ and $N(j) \subseteq N(i)$ ($N(i) \equiv N(j)$). To reduce complexity, we will draw a path with bidirectional edges instead of a clique.

It is important to note that a maximal (non-expandable) path of the community graph will represent a nested community or, in other words, a nested subgraph. For example, if the community graph of G is a single P_n (a path of n vertices, as in Figure 3.2a), the original graph G is fully nested, whereas if G is fully not nested (i.e., G does not have a single pair of vertices (i, j) where $N(i) \subseteq N(j)$), its community graph will be a graph with no edges.

3.2.2 Community detection algorithm

Now we introduce an algorithm to find overlapping nested communities, that is, fully nested subgraphs of G. The core parts of the detection algorithm reconstruct the community graph from the input graph and then find the community graph's maximal (non-expandable) paths to enumerate the communities. The main steps are the following.

Reconstructing the community graph

To reconstruct the community graph, we first enumerate all nested vertex pairs. Here, instead of greedily performing $\approx n^2$ comparisons, we can use the same trick used in Chapter 2 and [41]. That is, we do not compare vertices that have no common neighbors, since they are certainly not

nested. Instead, we calculate $\mathrm{nest}(i, k)$ by first going through $j \in N(i)$, and pick $k \in N(j)$, where $k < i$ is true[1]. This way, i and k have at least one common neighbor (j), so they are potentially nested. In practice, this can save us a lot of computational time (especially in sparse graphs), and the number of discarded comparisons can be large, according to our experience. Since we do not exclude potentially nested vertex pairs, we do not lose any information in this step.

When comparing vertices, we also need to know the direction of nestedness between the vertices, for example, by calculating $\mathrm{sgn}(|N(i)| - |N(j)|)$ when $\mathrm{nest}(i, j) = 1$. Once we have done all the comparisons, we build the directed edge list of the nested pairs, where $i \to j$ is an edge if $\mathrm{nest}(i, j) = 1$ and $\mathrm{sgn}(|N(i)| - |N(j)|) \leq 0$ (or equally, $N(i) \subseteq N(j)$).

However, the list will contain a lot more edges than we need. Let's revisit the fully nested graph from Figure 3.1a for an example. Here $N(3) \subseteq N(2)$ and $N(2) \subseteq N(1)$, but as such, $N(3) \subseteq N(1)$ will also hold. Since we need to find maximal paths in the resulting graph, the transitive $N(3) \subseteq N(1)$ relationship and its corresponding edge are redundant and certainly not part of the maximal path $3 \to 2 \to 1$. To remove them, we perform a transitive reduction on the graph built from the nested edge list. As a result, for all triples $i \to j \to k$ the edge $i \to k$ will be deleted. This significantly reduces the number of edges to consider in the next step, which greatly improves the performance of the algorithm. This completes the community graph discovery.

Finding nested communities

Now that we have a community graph, we need to retrieve the list of nested communities. To do this, we enumerate the maximal (non-extendable) directed paths in the graph. Listing these paths can be done using any traversal method, such as a depth-first search. Due to the transitive reduction performed in the previous step, this can be accomplished quite quickly.

Here, each directed path represents a fully nested community, with the order of the vertices also encoding hierarchy. For example, if K_n (a clique of n vertices) is the input graph, the community graph will be P_n (a path of n vertices), which will have a single community with all n vertices in it.

Vertex compacting based on neighborhood

There is one edge case that complicates the search for maximal paths, where cycles are created due to bidirectional edges between vertices. To solve this problem, and also improve search performance, we first find vertices with equal neighborhoods and merge them into a single vertex before building the community graph. Isolated vertices are not compacted, and edges between vertices are ignored when checking for neighborhood equality. This means that an isolated K_2 (two nodes with an edge between them), for example, is not compacted. This change has multiple positive effects. First, the community graph is now guaranteed to be a directed acyclic graph (DAG) as there are no other factors that can introduce cycles, making maximal path finding much easier.

[1] This condition enables us to skip the calculation of $\mathrm{nest}(k, i)$, which would have the same result as the previously calculated $\mathrm{nest}(i, k)$

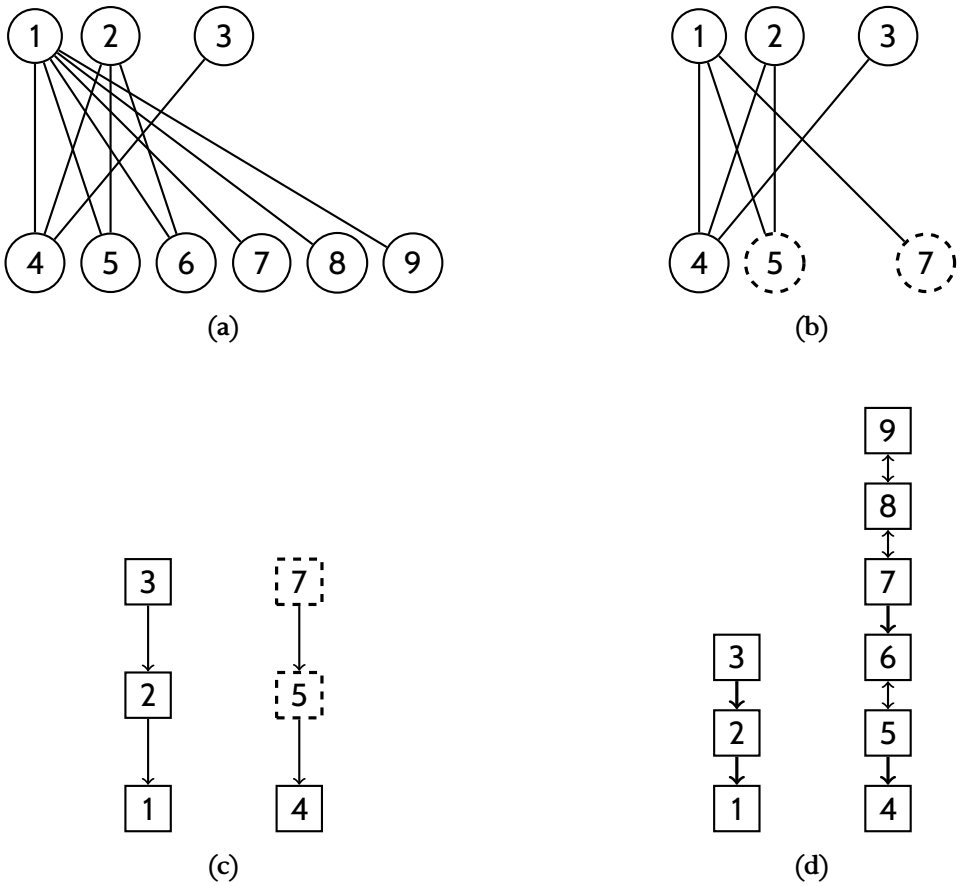


Figure 3.3: *Steps of the community detection algorithm. Starting from the input graph (a), we first compact vertices with equal neighborhoods (b), then build the community graph (c), and finally reverse the vertex compaction, adding the bidirectional edges.*

It also makes the resulting community graph smaller by having it be built from fewer vertices, improving performance. For example, a star graph with any number of nodes will have a compacted community graph of just two nodes and a single edge.

We then find the maximal paths as normal, treating the merged vertices as a single vertex. Finally, we recover the original vertices by expanding the merged vertices and inserting edges in both directions between them.

Figure 3.3 shows all the steps of the algorithm on an example graph. When enumerating the communities, we traverse the compacted graph (Figure 3.3c) and then insert the removed vertices into the paths.

Listing 3.1: Pseudocode of the nested community detection algorithm. Despite having three nested for loops, the algorithm only iterates over (i, k) pairs of vertices that have at least one common neighbor.

```
nested_pairs ← ∅                                                    1
for (i in V) {                                                      2
  for (j in N(i)) {                                                 3
    for (k in N(j)) {                                               4
      if (i ≥ k) continue                                           5
      if (nest(i, j) = 1) {                                         6
        if (sgn(|N(i)| − |N(j)|) = 1) {                             7
            nested_pairs ← nested_pairs ∪ (j, i)                    8
        } else {                                                    9
            nested_pairs ← nested_pairs ∪ (i, j)                    10
        }                                                           11
      }                                                             12
    }                                                               13
  }                                                                 14
}                                                                   15
nested_pairs ← transitive_reduction(nested_pairs)                   16
comms ← longest_paths(nested_pairs)                                 17
```

3.2.3 Remarks

Here we pinpoint some key areas in the behavior of the algorithm. The algorithm's pseudocode is available on Listing 3.1.

1. **Non-bipartite graphs.** A major advantage of the algorithm is that it does not exploit any property specific to bipartite graphs. In theory, this could make it directly applicable to any (unweighted) non-bipartite graph. In practice, we need to solve the problem of connected vertices described in Section 1.1. As with Equation (1.2) we ignore the connection between two vertices when comparing them. This, combined with vertex compacting, results in the algorithm correctly finding nestedness in non-bipartite graphs too (as later demonstrated in Section 3.4.1).

2. **On constrained community detection.** We also need to make some comments about the community structure detected by the algorithm. The algorithm is designed to detect certain types of overlapping communities (specifically communities that satisfy the constraint of being fully nested), essentially performing a "constrained" community detection. The algorithm is also not a heuristic to detect nestedness. This is due to the fact that we start by enumerating all possible $\approx n^2$ comparisons and then exclude only those pairs that are

guaranteed not to be nested. As a result, all remaining, potentially nested, vertex pairs are compared, and no stochastic elements are included in the process.

3. **Permissive nestedness.** As a possible future direction, we would also like to mention the potential of relaxing the requirements of nestedness for communities. So far, we have talked about how the algorithm detects communities that satisfy a certain "constraint". This constraint can be quite strict, as two nodes that share *largely* the same neighborhood (with a few deviations) are considered to be non-nested. There are many metrics that quantify the degree of nestedness of a graph. They allow us to see not only whether a graph is fully nested or not, but also *how much* nested it is. To increase the flexibility of our algorithm, we can similarly allow pairs of vertices that are not fully nested to belong to the same community, *above* a certain nestedness threshold, for example. The algorithm currently does not support this, but it is easy to implement.

3.3 Generation of overlapping communities

Previously, we have shown an algorithm that can reconstruct the community graph from an arbitrary input graph. Now, we present an algorithm that is capable of generating bipartite graphs with multiple overlapping fully nested groups of vertices, based on an input community graph. The method also returns the ground truth nested structure, making it suitable for use when benchmarking algorithms that find overlapping nested communities.

The generated structure is more general than in-block nestedness [42], where the graph is partitioned into disjoint, fully nested "blocks". Our method is capable of generating not only this structure, but making it a more versatile approach.

We believe that the proposed algorithm is useful not only for testing our nested community detection algorithm, but also for creating benchmark data sets for future methods that detect overlapping nested structures.

3.3.1 Benchmark generator algorithm

Now that we have a method for describing nestedness using a directed graph, we will present an algorithm that generates a bipartite graph that satisfies the nested structure described by the community graph. That is, if there is an edge $i \to j$ in the community graph, the resulting graph will have $N(i) \subseteq N(j)$. We will show that the algorithm is capable of generating not only fully nested graphs, but also graphs with overlapping nested communities.

To generate a bipartite graph from a community graph, denoted by G_c, we first perform a topological sorting on G_c, since we need to generate neighbors for each vertex v such that its predecessors (denoted by $\mathrm{in}(v)$) already have their neighbors. To do this, we must assume that the community graph is acyclic, as there must be at least one vertex with no predecessors, which

will be the starting vertex. When visiting a vertex, we add all the neighbors of its predecessors to the neighbors of the current vertex and generate a new neighbor for it (if we visit the i-th vertex, we can label the new vertex $n + i$). The first part guarantees nestedness, while the new neighbor makes sure that the two vertices do not have the same neighborhood (in which case there should be both an $i \to j$ and a $j \to i$ edge in G_c). Formally, we have the subroutine visible in Listing 3.2.

Listing 3.2: Pseudocode of the nested graph generator algorithm.

```
i ← 1                                                          1
order ← topological_sort(G_c)                                  2
while (i ≤ n) {                                                3
  v ← order(i)                                                 4
  N(v) ← { n + i } ∪ ⋃_{j∈in(v)} N(j)                           5
  i ← i + 1                                                    6
}                                                              7
```

3.3.2 Remarks

1. **Generating random nested community structures.** The algorithm we have described so far is used to generate a graph with a given nested structure from an input community graph. To generate random graphs with overlapping nested structures, we can use random input graphs. However, the input graph must be a DAG. This can be achieved, for example, by sampling a random spanning tree of the complete graph K_n and randomly orienting its edges.

2. **Cycles in the community graph.** Because the algorithm performs topological sorting and requires the input graph to have a vertex with no predecessors (i.e., one that is not part of a cycle), it is much simpler to work with an acyclic community graph. This prevents us from generating vertices with equal neighborhoods, however, the vertex compacting approach described in Section 3.2.2 could be adapted to the generator algorithm to make this possible.

3. **Generated graph sizes.** Since the algorithm takes a community graph of size n and generates a neighbor for each vertex, the resulting graph will have exactly $2n$ nodes. This also makes the algorithm incapable of generating bipartite graphs with classes of different sizes – a trivial example is the star graph.

4. **Multiple blocks.** We have not touched on whether the input G_c DAG has to be weakly or strongly connected yet. The algorithm can handle community graphs with multiple components and will render a component of the community graph as a component in the

generated bipartite graph. For example, a graph with multiple disjoint P_k components (that is, a directed path of k nodes) results in an in-block nested bipartite graph [42].

5. **Nested structure of the other class.** As mentioned in the description of the algorithm and in remark 2, the input to the algorithm is a DAG that describes the community structure of *one class* of the graph. The other class of the generated bipartite graph is not part of the input community graph (and thus the ground-truth). However, generating the bipartite graph from the community graph of the other class, we get a final bipartite graph that is isomorphic to the one generated using the original input.

3.4 Experiments

In this section, we will examine the performance of the nested community detection algorithm from several perspectives. First, we verify that the algorithm is able to detect the nested structures in a few basic examples, then whether it can find all ground-truth communities of the benchmark graphs generated by the algorithm described in Section 3.3, fully reconstructing the input community graph. We then examine the community structure of graphs commonly used when benchmarking nestedness metrics. Then, in a separate section, we compare the results of our method with other community detection algorithms to identify key differences in the discovered community structures. Finally, we measure the execution time of our algorithm in the function of node and edge counts in various graphs.

To evaluate our results and quantify nestedness, we use the NODF [22], discrepancy [24] and temperature (using the Binmatnest algorithm) [23] nestedness metrics on bipartite graphs. To make it easier to compare these metrics with our community structure, we derive our own nestedness metric from the community graph: the average fraction of communities a vertex is part of, called *vertex presence*. For normalization purposes, this number is multiplied by 2 in bipartite graphs, since perfectly nested bipartite graphs have 2 perfectly nested communities, one for each class. Vertex presence ranges from 0 to 1, where 1 means every vertex is part of every community, i.e., there is only a single community with all vertices in it. When vertex presence is low, it means that vertices are part of few communities while the total number of communities is high. When vertex presence is calculated specifically on the nested community structure, a maximal presence means there is one nested community (path), so the graph is fully nested, and a minimal presence means that every vertex is in its own nested community, having no nestedness in the network at all. Intuitively, a larger value means a vertex is part of a larger portion of the nested communities, which increases the overall nestedness of the network. We also note that since our community detection algorithm works on non-bipartite graphs, unlike the previously mentioned metrics, vertex presence is not restricted to bipartite graphs. Formally, we can obtain vertex presence for a vertex v by

calculating

$$\mathrm{pres}(v) = \frac{|\{C : C \in \mathcal{C}, v \in C\}|}{|\mathcal{C}|}, \tag{3.1}$$

where $\mathcal{C}$ is the set of nested communities (maximal paths in the community graph). Since the domain of $\mathrm{pres}(v)$ will depend on n (its lowest value is $\frac{1}{n}$), we can normalize it into the $[0, 1]$ range, so that 0 means entirely not nested in all cases and 1 still means fully nested. This can be achieved using

$$\overline{\mathrm{pres}}(v) = \frac{n}{n-1}\left(\mathrm{pres}(v) - \frac{1}{n}\right). \tag{3.2}$$

For compactness, we will use *average vertex presence* as a metric of the entire *graph* by averaging the normalized vertex presence across all vertices.

As opposed to regular communities, an interesting property of nested communities is that the *position* of a vertex inside a community is informative, too. If a vertex is at the beginning of a community (path), then its neighborhood is a subset of the other vertices of the community. If, on the other hand, a vertex is at the end of a community, its neighborhood is a superset of the other vertices in the same community. We call the quantification of this *vertex position*, and it can be calculated using

$$\mathrm{pos}(v, C) = \frac{i-1}{\max\{1, |C|-1\}}, \tag{3.3}$$

where $\exists i : C_i = v$ (v is the i-th vertex of C), thus $\mathrm{pos}(v, C)$ is only valid if $v \in C$. We perform normalization in the denominator that makes the position of vertices at the beginning of a community 0, even if they are the sole vertex in a community, assuming indexing starts at 1. With this, we can calculate the position of a vertex on all nested communities it is present in, then take its average to get the *mean vertex position* of said vertex.

These two metrics allow us to measure nestedness both on a graph level (by calculating *average vertex presence*) and on a vertex level (through *vertex position*).

3.4.1 Results on typical examples

Before testing the algorithm on generated benchmark examples, we first demonstrate that the algorithm finds basic nested structures. Figures 3.4a and 3.4e show that the algorithm correctly identified the full bipartite graph that has two fully nested parts: nodes in one class belong to the same community. Figures 3.4b and 3.4f show the same concept, but in a special case: the star graph is also considered fully nested, and the algorithm correctly identifies the upper class as fully nested and the single node of the bottom class as another. Figures 3.4c and 3.4g show that the nodes of the fully non-nested bipartite graph are all correctly put into different communities.

Finally, to demonstrate that the algorithm works with non-bipartite graphs, through Figures 3.4d and 3.4h we show that all five nodes of the complete graph are correctly classified as a single community.

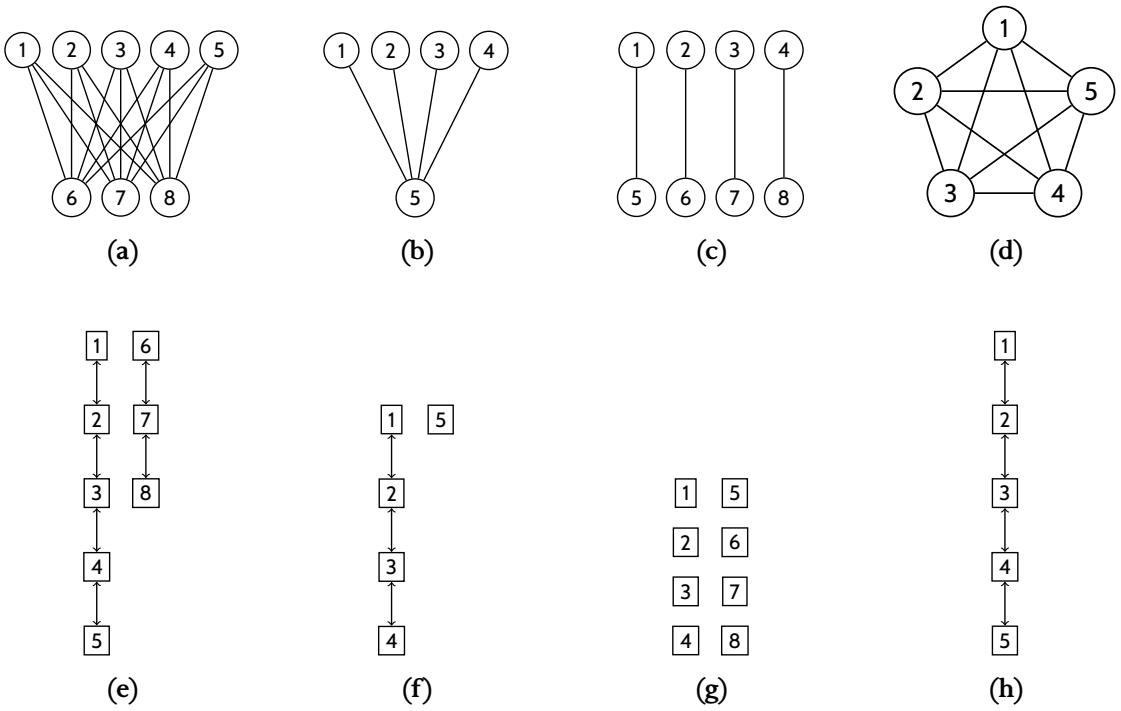


Figure 3.4: *Graphs showing typical nested configurations (a–d) and their community graphs (e–h).*

3.4.2 Results on benchmarks

In this section, we will compare the ground-truth community structure of the generated benchmark graphs with the one found by our algorithm. In order to create a benchmark graph, we use one or more random spanning trees (creating a spanning forest) sampled from complete graphs and orient their edges randomly. These oriented spanning trees will be the community graphs of the benchmark graphs.

The benchmark is set up as follows. We generated 2000 random graphs with ground-truth communities with 1 to 4 blocks (components) and 1 to 60 nodes per block. Let n_b denote the sum of the number of nodes in the input to the generator across all blocks. Note that we know the ground truth for the first n_b nodes as these are the nodes of the input community graph, the rest n_b nodes are generated; this was discussed in more detail in Section 3.3.2.

After generating the benchmark graphs, we run the algorithm on them and compare the first n_b nodes of the result with the known ground truth. Again, we cannot compare the rest due to the limitations of the generator algorithm, however, we do not need to, since nestedness is a symmetric property for bipartite graphs. Furthermore, correctly recovering the ground-truth community structure for the first n_b nodes means that the algorithm is capable of reconstructing the entire community graph on the input.

Our tests show that all community graphs in the benchmark set were correctly recovered and

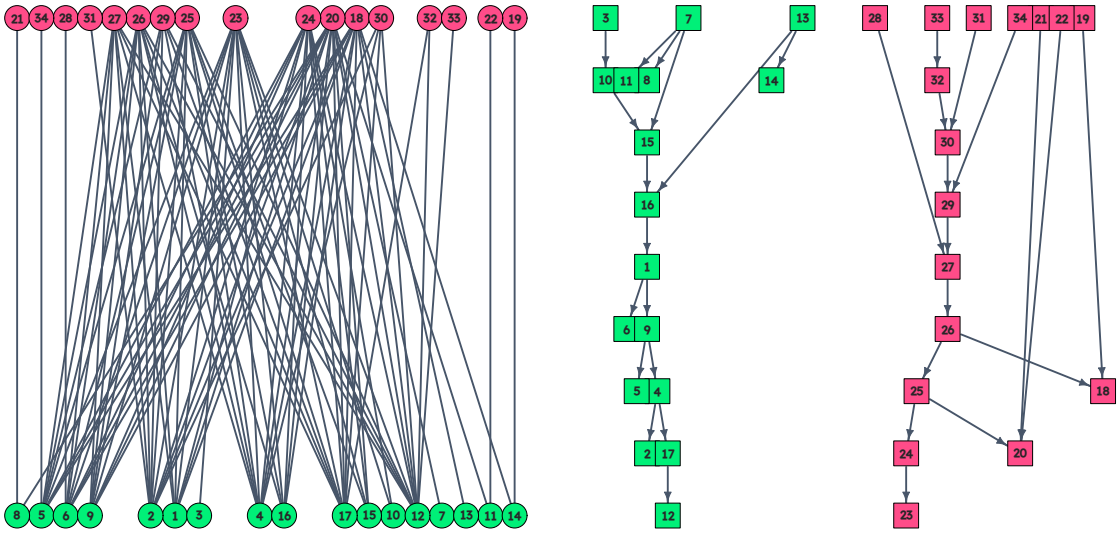


Figure 3.5: *A generated bipartite graph (a) and its detected community graph (b). The ground truth is known for the green (bottom) class.*

that the resulting community structures were exactly consistent with the ground truths. Figure 3.5 shows a generated benchmark graph and its detected nested community structure. Looking at this figure, we can presume that there are many communities, even on smaller graphs, with many of them overlapping over a large part of the vertices.

3.4.3 Results on real-world networks

Bipartite networks

After validating that the algorithm is capable of reconstructing the community graph, we take a look at real-world networks used to test nestedness metrics. First, we examine the nested structure of ecological networks from Web of Life [32]. We examine the algorithm's output on pollinator (mutualistic) and host-parasite networks. These are two sets of small bipartite networks of species interactions.

The first network we examine is `M_PL_069_03` [43] (Figure 3.6), a tiny pollination network of seven plant and four hummingbird species created from observations in eastern South America. Vertices 1–7 represent plants, while vertices 8–11 represent hummingbirds. For legibility reasons, we show the vertex IDs instead of species names on the plot and clarify where needed. The NODF value of the network is 75.926 (in the range $[0, 100]$, where 100 means fully nested), and its discrepancy value is 2 (where 0 means fully nested), suggesting that it is indeed a highly nested

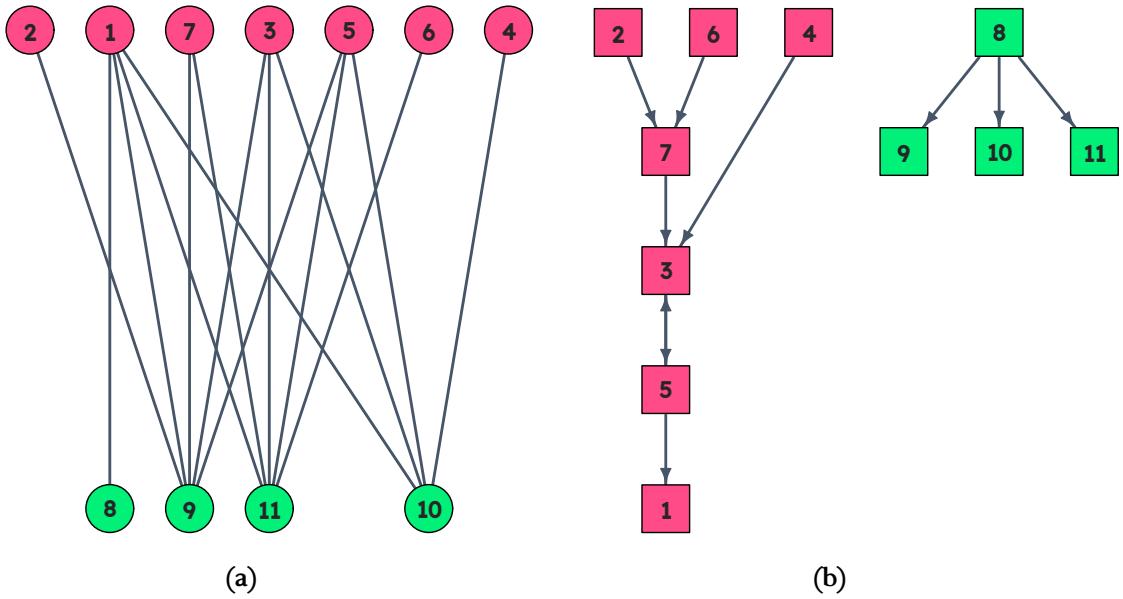


Figure 3.6: *The original* `M_PL_069_03` *graph (a) and its nested community graph (b).*

network. The average vertex presence of the community graph is 0.606.

We can see that the graph is clearly not fully nested, as there are multiple communities (paths) on its community graph in both classes. However, it does have large nested communities that cover most of the vertices in each class with high overlap. In both classes, we have three communities that cover all vertices of that class. In the upper class (plant species, purple), the communities cover a larger part of the class with 5 (71%) and 4 (57%) vertices. Three plant species (vertices 3, 5, and 1) also play a key role in these communities, as they are part of all three communities. They represent generalist entities in the network, connected to most vertices of the other class, i.e., most hummingbirds visit them. Vertex 7 is also part of two communities, only vertices 2, 4, and 6 are part of a single community. We can see that they are all visited by only one species of hummingbird.

Looking at the class of hummingbirds, the community structure is simpler because the class has four entities only. Interestingly, we can see that vertex 8 (amazilia versicolor) is part of the three communities but visits only a single plant (vertex 1, aechmea cylindrata), a plant that all other hummingbird species visit, too. This connection alone makes the amazilia versicolor nested with all other species. This is an important aspect of nested networks, where specialist species tend to pick the generalists in the other class.

`M_PL_069_01` [43] (Figure 3.7) is a slightly bigger pollination network of 18 plants (vertices 1–18) and 6 hummingbirds (vertices 19–24), with a lower connectance. Interestingly, the community structure shows less symmetry in terms of the classes, with eight overlapping plant communities

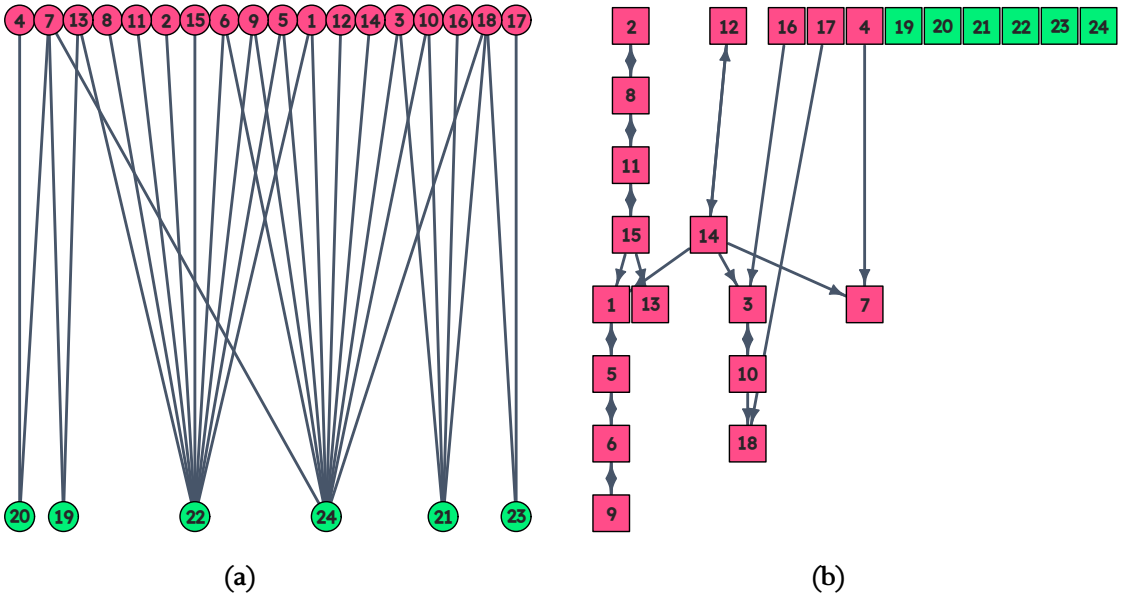


Figure 3.7: *The original* `M_PL_069_01` *graph (a) and its nested community graph (b).*

and the six hummingbird species all in their own class. On closer inspection, we can see that some hummingbird species visit largely the same plants as others, but there are always plants that one visits, but the other one does not, and vice versa. For example, vertex 21 (Clytolaema rubricauda) is connected to most of the neighbors of vertex 24 (Thalurania glaucopis), but vertex 16 (Vriesea erythrodactylon) is only connected to 21, creating a $2K_2$.

This graph also highlights the asymmetric nature of nested communities: while we can observe some degree of nestedness (with some paths covering half the vertices) in the class of plants, the class of birds is fully non-nested.

Moving on to larger networks, such as `M_PL_058` [44] (community graph visible in Figure 3.8), untangling the community structure becomes increasingly more difficult, with many communities (277 over two classes) overlapping each other. However, we can make some important observations on the community graph. For example, the community graph has few zero-degree vertices, which means that most vertices contribute to the overall nestedness of the graph, but they are nested only with *some* other vertices. We can also see that in both large components, there are only a few vertices with high total degrees. In the largest component of the community graph (colored in green, containing bees), there is a bee (of the Andrena genus) with a high in-degree, connecting lots of nested communities, by interacting with 22 plants out of 32. In the second-largest component, which consists of plants, there is a vertex with a high out-degree (a Vicea lutea), being part of a lot of communities at once by having only a single connection to the aforementioned Andrena bee.

Such large networks also show that partially nested networks can have a huge amount of

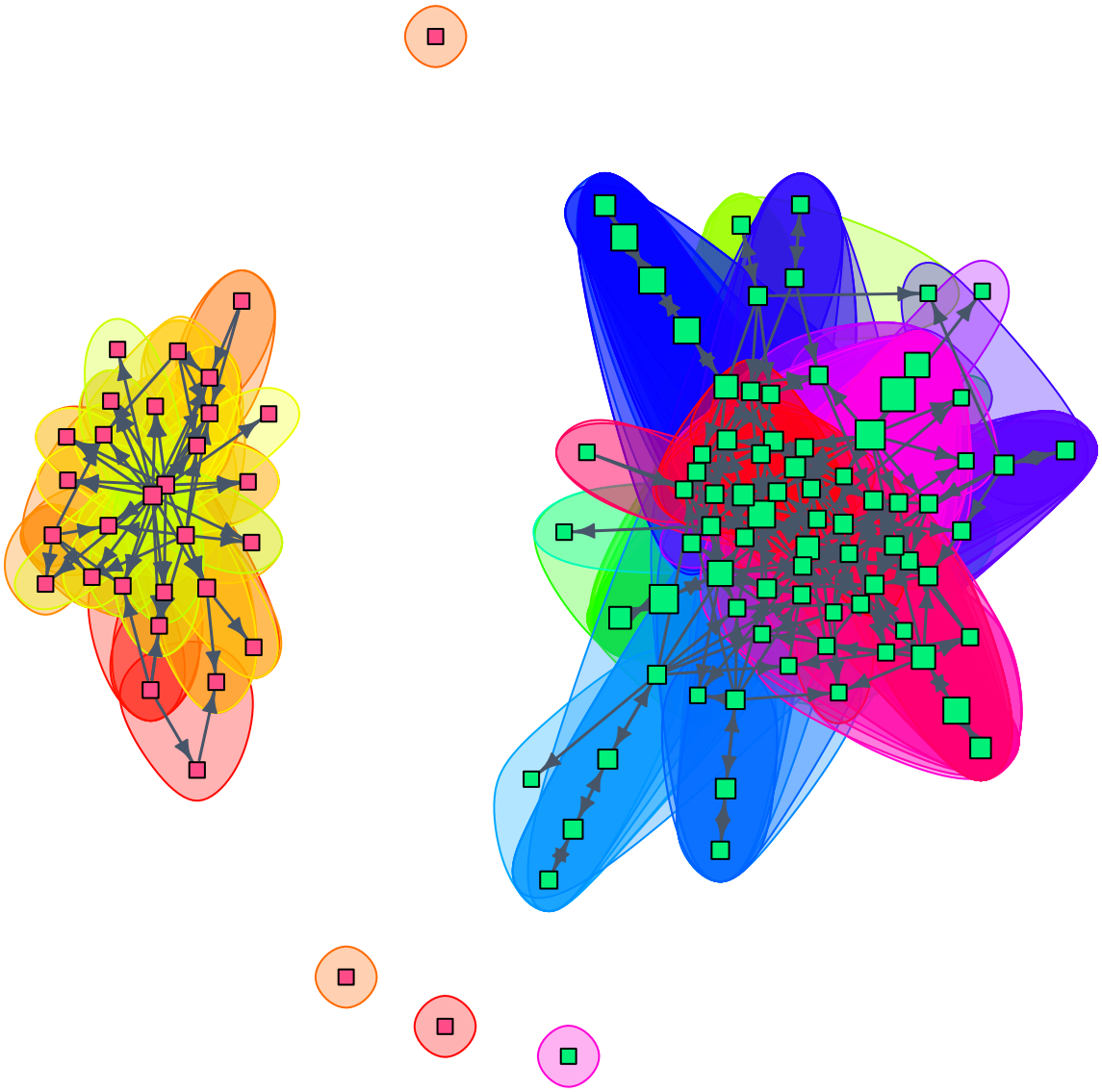


Figure 3.8: *Community graph of the slightly larger* `M_PL_058` *graph. Larger vertex sizes correspond to higher in-degrees (including transitive edges).*

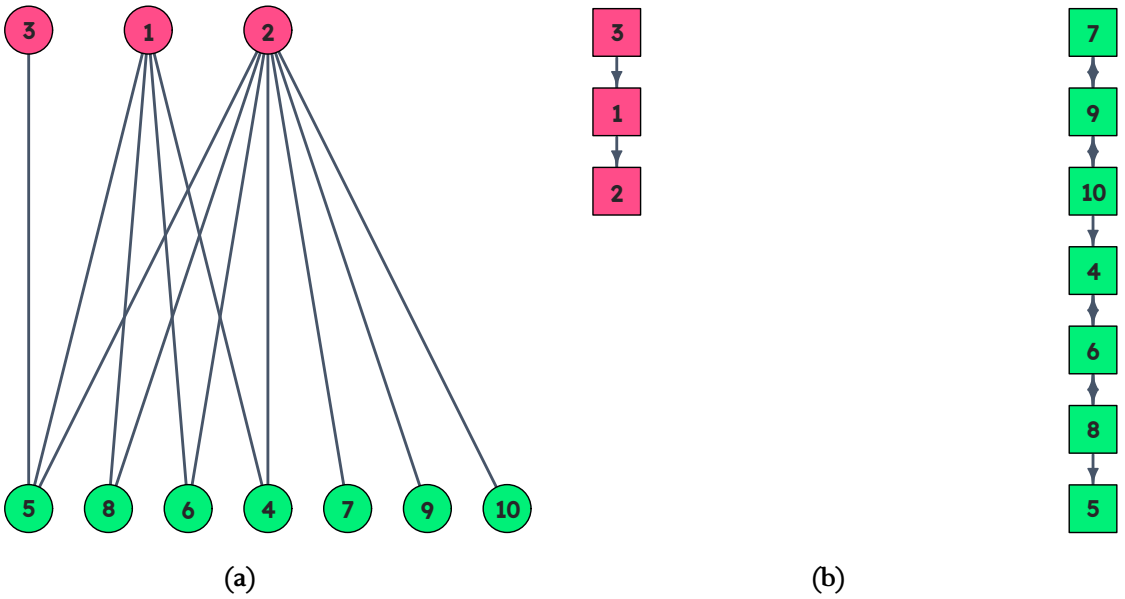


Figure 3.9: *The* A_HP_015 *host-parasite network (a) and its nested community graph (b).*

nested communities. We expect a fully nested bipartite network to have two communities (one for each class), a fully nested non-bipartite network to have a single community, and a fully non-nested network to have n communities, each vertex belonging to its own community. Partly nested networks, on the other hand, may have more than n communities: M_PL_057 has $n = 997$ vertices and $m = 1920$ edges but at the same time $2000 > m > n$ communities, with an NODF value of 7.23 and a vertex presence of 0.045. Thus, the number of communities is not a linear function of nestedness. This also means that, unfortunately, the number of communities does not perfectly reflect the network's nestedness.

Host-parasite networks

The host-parasite networks scored an average vertex presence of 0.28 versus 0.188 and an average NODF score of 52.033 versus 30.857 in the pollination set, suggesting that the host-parasite networks are on average more nested.

This set contains a fully nested network: A_HP_015 (Figure 3.9), a network of three rodents (vertices 1–3) and seven parasites (vertices 4–10). We can see that a specialist entity (Microtus oeconomus, vertex 3) interacts only with a generalist species (Amphipsylla marikovskii, vertex 5) and vice versa. The network has a vertex presence score of 1 and a discrepancy of 0, but its NODF value is 75 (where 100 would mean fully nested). Similarly, its temperature score isn't showing perfect nestedness, either, at a value of 1.05 instead of 0.

Table 3.1: *Computed properties on a subset of the full dataset. Notations: D: graph density; $\overline{C_i^w}$: mean local vertex transitivity (clustering coefficient); Q: Newman-modularity (based on the multi-level modularity optimization algorithm for finding community structure [45]); $|\mathcal{C}|$: number of nested communities; $\overline{\mathcal{C}_s}$: average nested community size; T_b: Binmatnest temperature (0–100, lower = more nested); T_{nodf}: NODF value (0–100, higher = more nested). Graphs marked with an asterisk (*) were directly mentioned and analyzed in the chapter. NA nestedness values mean the graph is not bipartite, and thus the metric could not be calculated.*

	n	m	D	$\overline{C_i^w}$	Q	$\lvert\mathcal{C}\rvert$	$\overline{\mathcal{C}_s}$	pres	T_{b}	T_{nodf}
M_PL_001	185	361	0.02	0.00	0.50	284	3.86	0.04	2.83	14.46
*M_PL_057	997	1920	0.00	0.00	0.53	2000	22.92	0.05	0.79	7.23
*M_PL_058	113	319	0.05	0.00	0.30	277	4.07	0.06	10.13	28.02
*M_PL_069_01	24	29	0.11	0.00	0.43	14	2.93	0.21	34.29	31.07
*M_PL_069_03	11	15	0.27	0.00	0.21	6	3.33	0.57	12.09	75.93
*A_HP_015	10	12	0.27	0.00	0.21	2	5.00	1.00	1.05	75.00
A_HP_016	27	52	0.15	0.00	0.24	11	5.18	0.36	21.82	56.25
A_HP_017	14	19	0.21	0.00	0.26	4	6.00	0.85	4.97	78.26
A_HP_025	58	107	0.06	0.00	0.39	56	3.88	0.12	21.56	25.22
A_HP_026	33	142	0.27	0.00	0.11	36	6.94	0.40	6.46	87.78
adjnoun	112	425	0.07	0.19	0.28	166	2.57	0.01	NA	NA
celegansneural	297	2359	0.03	0.31	0.39	238	1.66	0.00	NA	NA
dolphins	62	159	0.08	0.30	0.52	65	2.08	0.02	NA	NA
*karate	34	78	0.14	0.59	0.44	33	3.64	0.08	NA	NA
netscience	1589	2742	0.00	0.88	0.96	895	3.38	0.00	NA	NA
power	4941	6594	0.00	0.11	0.93	4256	1.96	0.00	NA	NA
dswomen	32	89	0.18	0.00	0.31	27	2.37	0.12	36.19	48.58
*families	15	20	0.19	0.22	0.40	13	1.92	0.07	NA	NA
les_miserables	77	254	0.09	0.74	0.57	77	5.78	0.06	NA	NA

Nestedness metrics

Examining the relationship between vertex presence and some nestedness metrics, we can see that while vertex presence and NODF behave similarly with few exceptions (Figures 3.10a and 3.10c), Binmatnest gave small scores (meaning high nestedness) to some graphs with low vertex presence (Figure 3.10b). These graphs had low NODF and high discrepancy scores, both suggesting low nestedness. Similarly, NODF gave a score of 75 to a fully nested network in the host-parasite network set, where the discrepancy was 0 and vertex presence was 1 (Figures 3.10c and 3.10d).

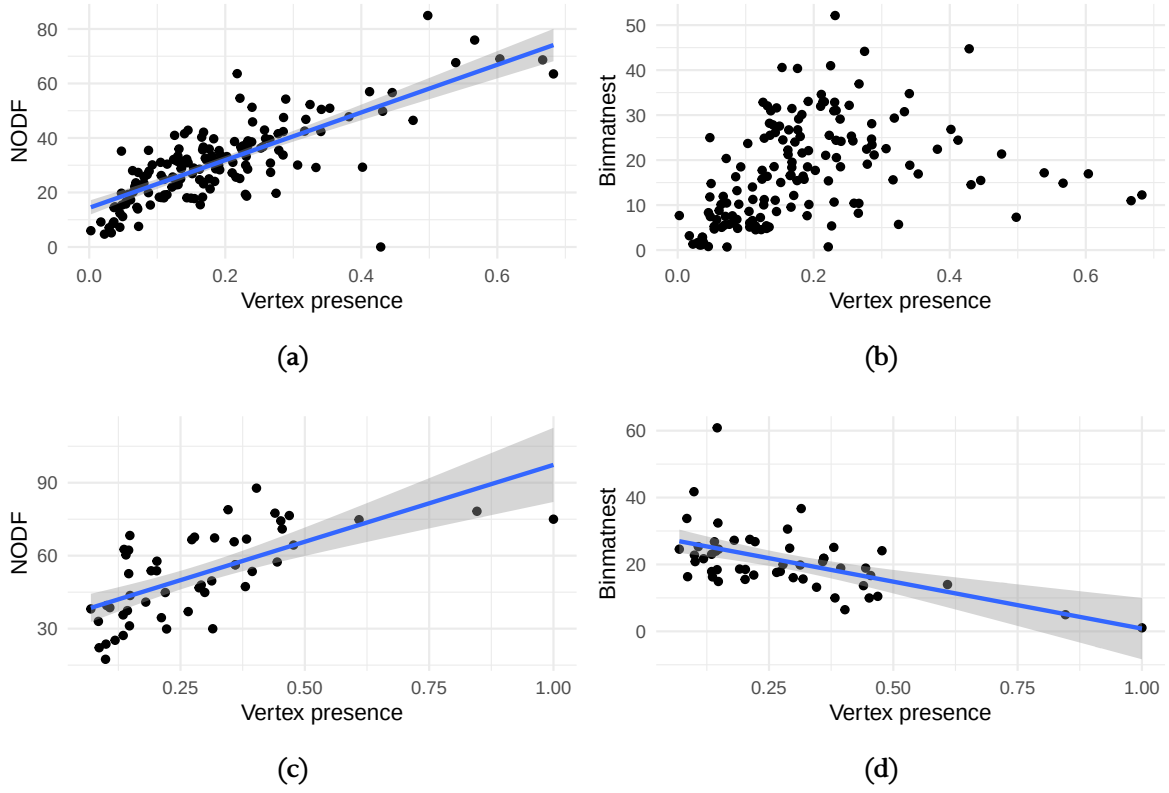


Figure 3.10: *Comparison of vertex presence with the NODF and Binmatnest nestedness metrics on the pollination (a–b) and host-parasite (c–d) network set.*

Non-bipartite graphs

Now we are going to examine the algorithm's output in some common non-bipartite, mostly social networks. Nestedness in a social network can be interesting in the sense of information exchange and domination. If some i is connected to all acquaintances of j and more, and they get into a conflict, i can spread their position to everyone j knows, potentially dominating j.

A common example used when testing community detection algorithms is Zachary's karate club network [9], visible in Figure 3.11a. This is a social network of 34 members of a karate club who interacted outside the club. The club split into two, creating two communities, marked with two colors.

Although this is not an ecological network, we can see that there are only three weakly connected components in the community graph (Figure 3.11b), two of them being isolated vertices and the third formed by the rest of the graph. Most nested relationships are between vertices of the same color (karate community), except for three nodes. Node 1 (the instructor) is in a nested relationship with 9 other members out of 16 in its karate community, while node 34 (the administrator) is with 7 out of 16. The graph contains 33 nested communities with an average size of 3.67 vertices per

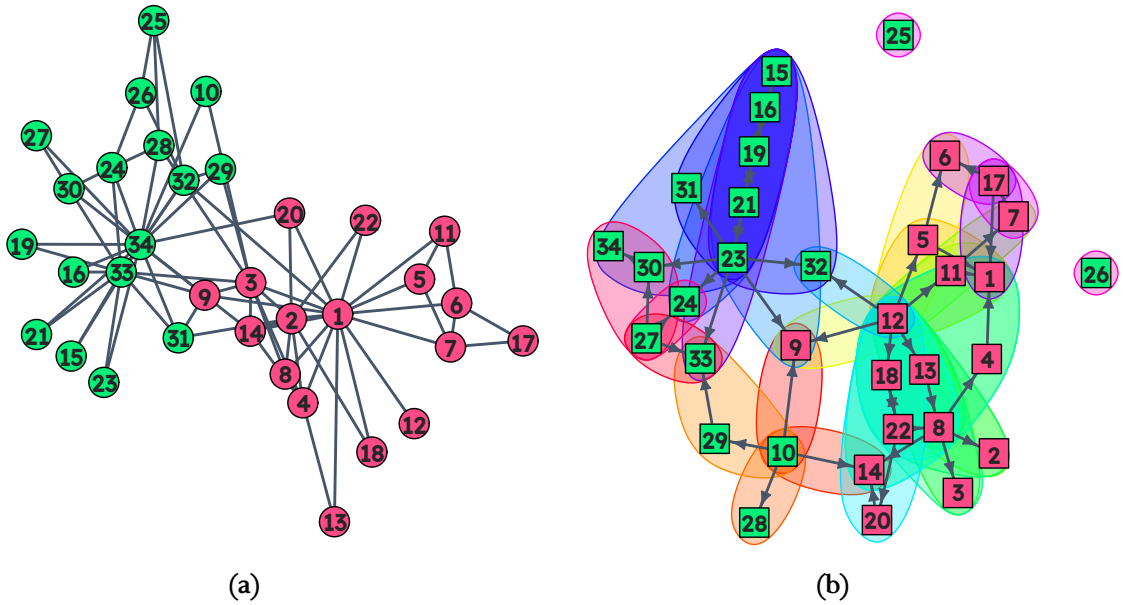


Figure 3.11: *Zachary's karate club network and its community graph. Vertex colors represent the ground-truth clusters.*

community and the mean vertex presence is 0.107. These observations lead us to believe that nestedness does not play a key role in the formation of the network.

The Florentine families network [46] (Figure 3.12a) contains marriage links between families during the Italian Renaissance. While this network ta to demonstrate centrality, the presence of nestedness may be interesting in the sense that families can be in a dominating position if they have connections to all neighbors of another family.

Looking at the community graph in Figure 3.12b we can see that there are few cases for this, most of them due to having a single neighbor that is common with one of the central families. For example, the Acciaiuoli family is nested with five others because they have a connection only with the Medici family. This means that if someone could influence the Medici family, they might also be able to influence the Acciaiuoli. Another interesting observation is that the Castellani family is not nested with anyone, so while they have their connections, they are not dominated by any other family. With a low mean vertex presence of 0.128 and a maximum community size of 2, nestedness does not appear to play a key role in this network, either.

3.4.4 Comparison with general community detection algorithms

Now, we compare the nested community structure found by our algorithm with the output of traditional community detection algorithms Linkcomm [47], MOSES [48] and CFinder [49]. These algorithms use different approaches for community detection, Linkcomm being a link partitioning

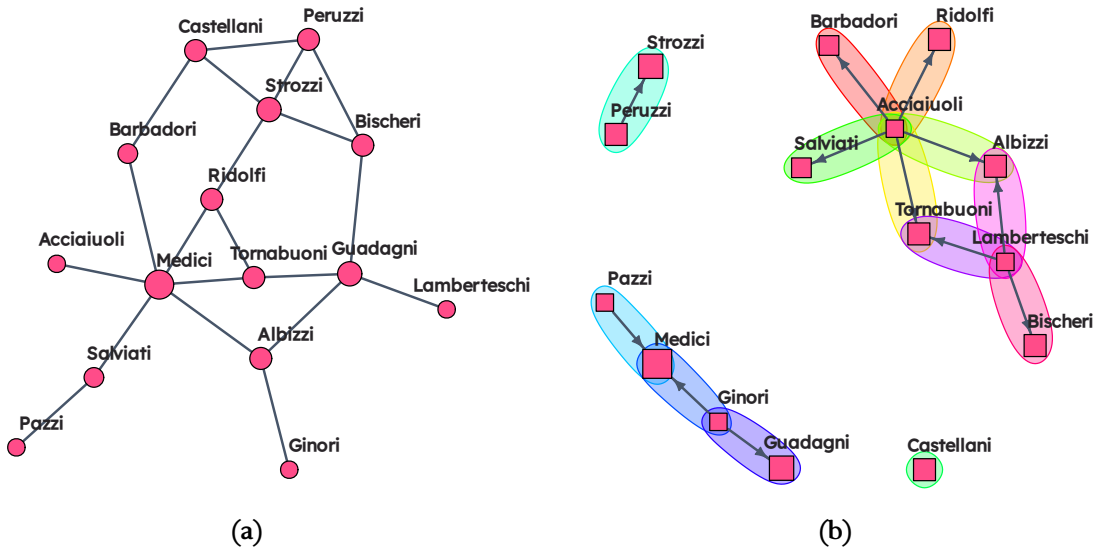


Figure 3.12: *The Florentine families network and its community graph.*

method, MOSES a fuzzy algorithm, and CFinder a clique search algorithm. Note that while Linkcomm, MOSES and CFinder propose to find communities where vertices within communities are more densely connected, our algorithm defines communities as fully nested subgraphs. While the community definitions are different, we perform these comparisons to highlight the differences between nested and traditional community structures.

Since our algorithm assigns each vertex to a community – vertices that don't belong anywhere else are put into their own communities – we modify the outputs of the community detection algorithms mentioned above so that omitted vertices are also assigned a community. This helps us to level the comparisons and also to avoid making false conclusions based on the number of communities, since a single community is only possible if all vertices are included in it.

Looking at the number of communities and their average sizes (Figures 3.13a and 3.13b), we can see that MOSES tends to identify fewer but occasionally larger communities, while CFinder created many – sometimes as many as $4n$ communities –, although its average community size was not the smallest in these cases either, meaning that there had to be greater overlap between them. This is confirmed by the vertex presence in Figure 3.13c. Vertex presence was low across all graphs and algorithms, which means that overall there is little overlap between the communities. The different community structures detected on Zachary's karate club network are shown in Figure 3.14.

Finally, we also calculate the Generalized Conventional Normalized Mutual Information [50] (GenConvNMI) index to measure Mutual Information between the community structure of our algorithm and the compared other algorithms. The NMI indices shown in Table 3.2 are high, especially in the case of CFinder.

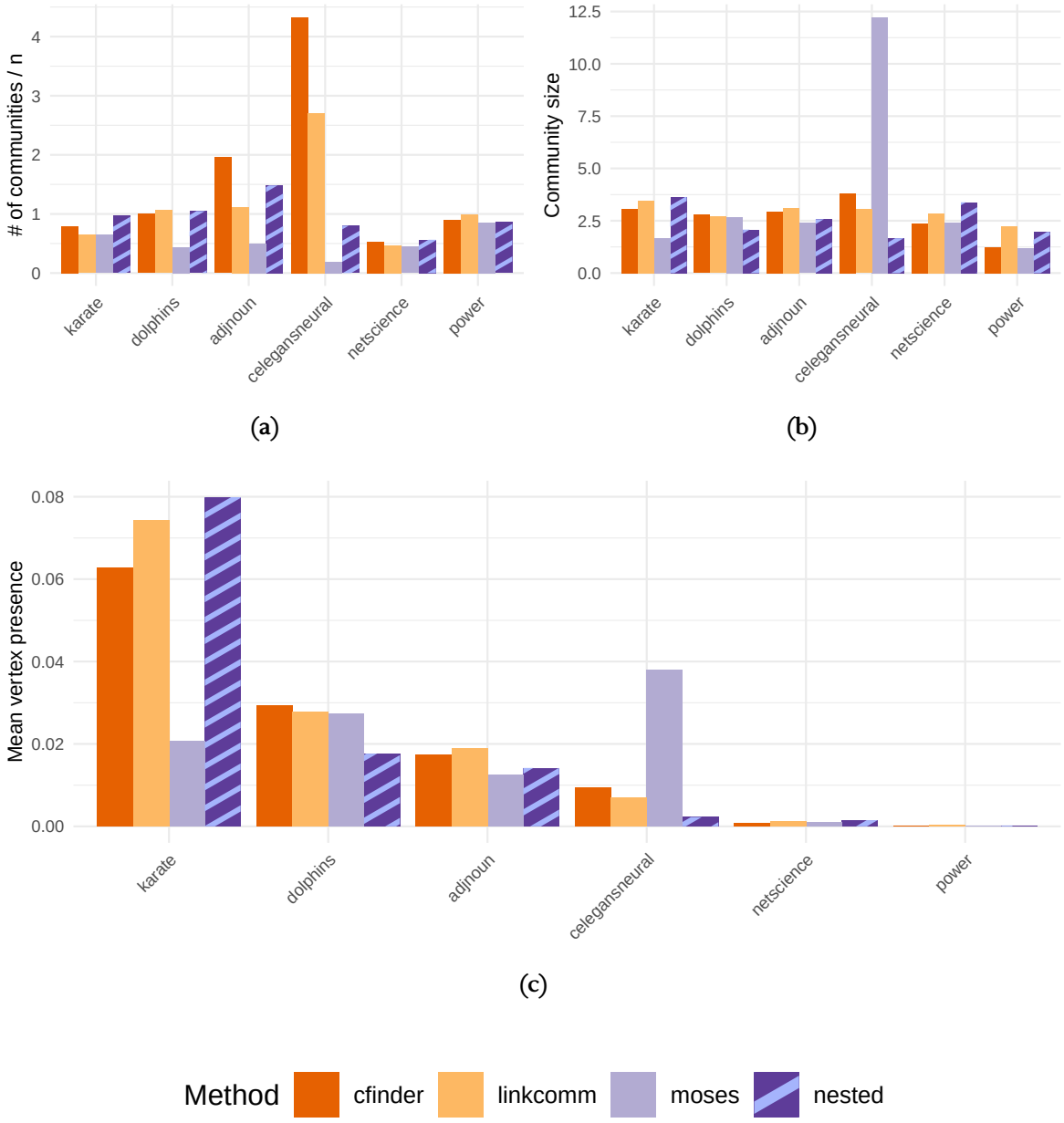
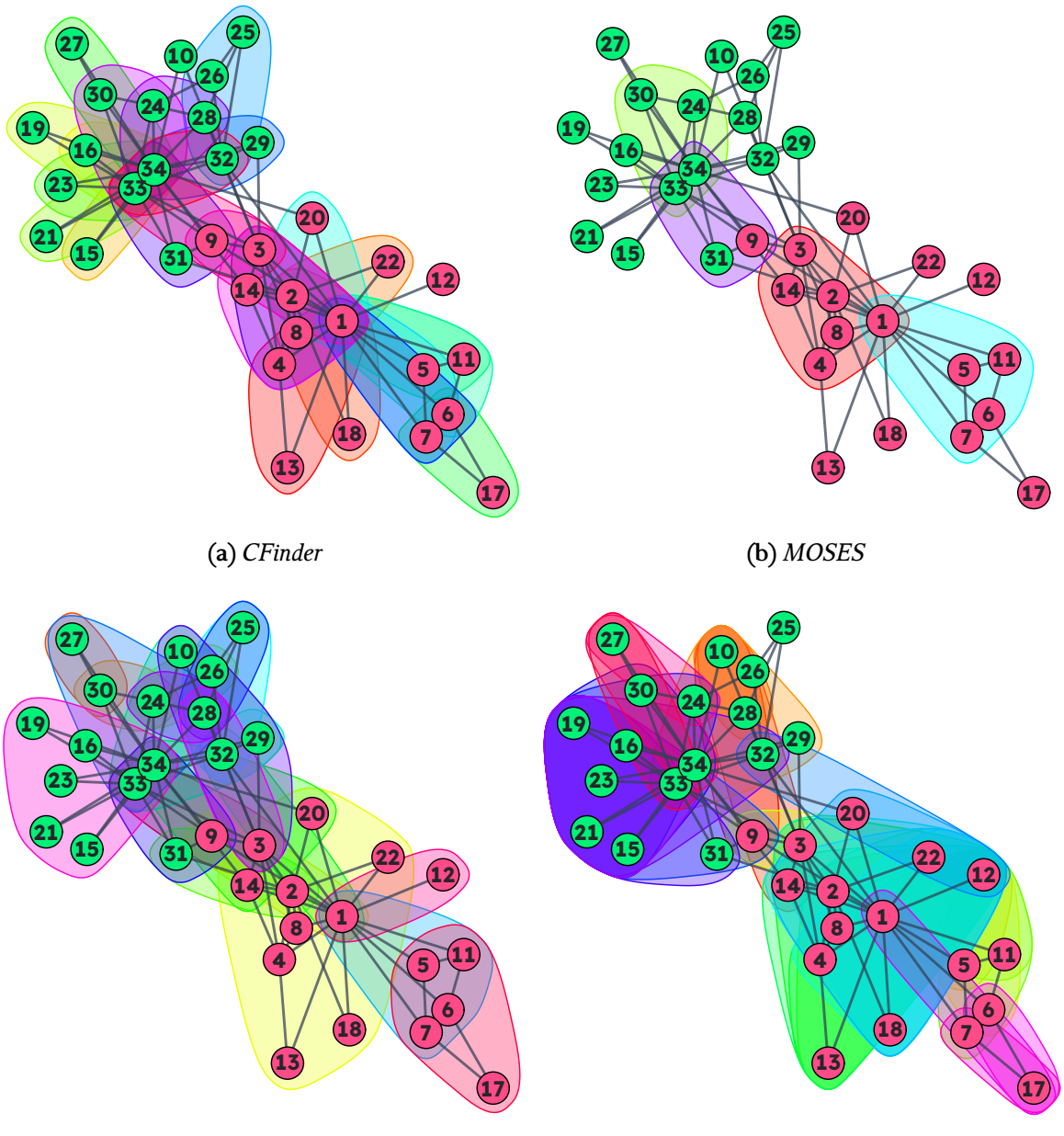


Figure 3.13: *Comparison of community statistics across algorithms and graphs.*



(a) *CFinder* (b) *MOSES*

(c) *Linkcomm* (d) *Our algorithm*

Figure 3.14: *Community structures detected by different algorithms on Zachary's karate club network. Communities with a single vertex are not plotted.*

Table 3.2: *Generalized Conventional Normalized Mutual Information [50] between the communities of our algorithm and the compared traditional community detection algorithms (higher = more similar). The subscripts* M, C *and* L *correspond to the algorithms MOSES, CFinder and Linkcomm, respectively. Graphs marked with an asterisk (*) were directly mentioned and analyzed in this chapter.*

	$\mathrm{NMI}_{\mathrm{M}}$	$\mathrm{NMI}_{\mathrm{C}}$	$\mathrm{NMI}_{\mathrm{L}}$
adjnoun	0.686	0.911	0.538
celegansneural	0.567	0.678	0.657
dolphins	0.655	0.821	0.676
*karate	0.645	0.699	0.533
netscience	0.941	0.953	0.949
power	0.963	0.974	0.927
dswomen	0.842	0.836	0.530
*families	0.879	0.894	0.724
les_miserables	0.576	0.698	0.617

3.4.5 Implementation and performance benchmarks

Finally, we show that our algorithm is scalable enough to be used on large networks. It was implemented in R [27] and C++ and its reference implementation, together with the example generator code, is open source[2]. Here, we measure the runtime of our algorithm across all analyzed non-synthetic graphs.

Figure 3.15 shows average runtimes of 100 executions on each of the analyzed bipartite and non-bipartite networks. The figure shows that while it is not perfectly linear, the algorithm's runtime doesn't scale steeply with the increase of the vertices or edges. We can see that it is capable of achieving runtimes of under a second on graphs with much more than 1000 vertices, or more than 10000 edges. For the details of the test environment, please refer to Section 3.5.

3.5 Conclusions

We introduced a novel constrained community detection algorithm that finds overlapping nested subgraphs of a given input graph and constructs a directed graph, called the community graph, representing this community structure in a compact form. In reverse, we also introduced an algorithm that generates bipartite graphs with any given nested community structure from the input community graph. Derived from the resulting community graph, we also introduced two metrics to measure nestedness in networks on both graph- and vertex levels. We have shown that our community detection method can uncover detailed nested relationships within the input graph. We demonstrated its capabilities through several benchmark networks and real-world networks

[2] `https://github.com/Hanziness/r-nested-comms/tree/v0.2`

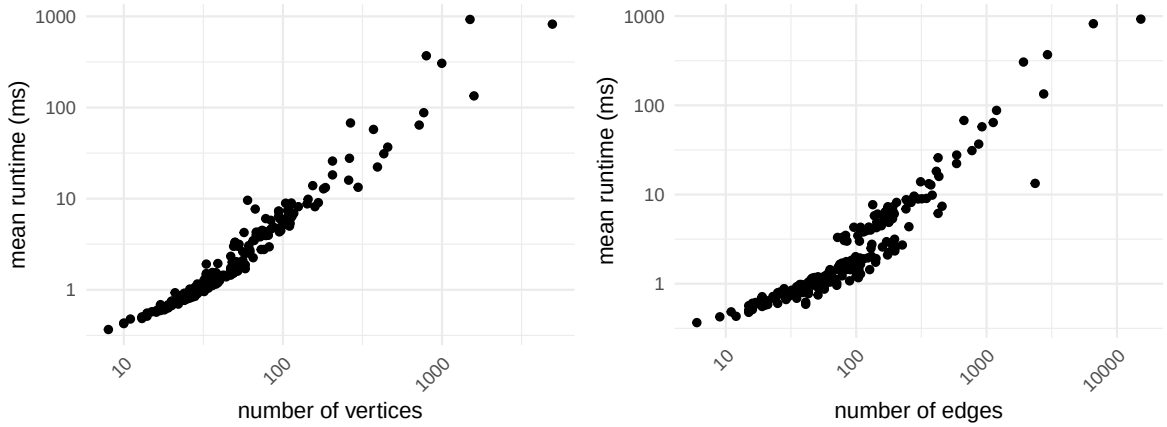


Figure 3.15: *Average execution times of the nested community detection algorithm's implementation over 100 runs, in the function of the number of (a) vertices and (b) edges.*

as well. Finally, we compared our method with multiple commonly used community detection algorithms that are able to find overlapping communities. We showed that our method can reveal a different type of community structure in both bipartite and non-bipartite graphs.

Appendix

Test environment

The performance tests were carried out on an ASUS ExpertCenter computer with an Intel Core i7-10700 CPU @ 2.90GHz (16 cores) CPU and 16 GB of RAM. The system was running Manjaro Linux (kernel version 6.3.5-2-MANJARO (64-bit)) with R 4.3.1 (latest packages as of 2023-07-01), compiled with the Intel Math Kernel Library (MKL). Execution time was measured using the `microbenchmark` package.

Part II

Applications of hierarchical clustering

Chapter 4

Hierarchical clustering for nestedness

Up to this point, I have described methods that detect overlapping communities, specifically fully nested structures. In this chapter, the center of attention on nestedness will not change, but instead of community detection, I will focus on clustering, albeit its extended variant. While clustering (or partitioning) itself may not provide information about the overlap between communities like the previous methods, disjoint partitions are sometimes preferable to overlapping groups as they are much easier to work with. Namely, we can assume that each entity belongs to exactly one cluster. This not only makes it easier to handle the clusters, but also to compare them. We can represent a clustering as a vector of length n (the number of nodes in the network), also called a *membership vector*, and there are several indices that compute the differences between two clustering membership vectors, such as the widely used Adjusted Rand Index [51].

Hierarchical clustering is a general tool that is able to create multiple levels of clustering of the same graph. This allows us not only to choose a clustering we deem optimal (based on some metric, for example), but also to extract information from the hierarchical relationship between the clusters. As we will see later, it also overcomes some limitations of traditional clustering approaches, such as the need to perform multiple independent clustering runs to obtain an ideal clustering. This chapter focuses on adapting hierarchical clustering to detect fully nested clusters in networks.

4.1 Introduction

The core of network science lies in the analysis of the structural and functional properties of networks, and one fundamental challenge is the partitioning of network nodes into groups, i.e., subnetworks or subgraphs. This is often done based on certain criteria. This process is essential in uncovering underlying structures within the network, facilitating the identification of communities, modules or special substructures that play specific roles in the overall system [16, 52, 53, 54, 55].

As we have seen in Chapters 2 and 3, nestedness in one such special substructure. The problem of identifying perfectly nested groups within a network has been recently addressed from

theoretical, algorithmic and data mining perspectives, considered as a graph coloring problem [25] and a graph clustering task [41, 56], respectively.

In the quest to enhance the precision of network analysis techniques, hierarchical clustering has emerged as a powerful tool for discerning hierarchical structures within networks [57, 58, 59]. By iteratively grouping nodes into clusters at varying levels of granularity, hierarchical clustering provides a multi-resolution perspective on network organization. This approach not only helps find cohesive communities or groups, but also captures the hierarchical relationships that exist among them, offering a much deeper understanding of network structure.

In this chapter, we address the problem of partitioning the nodes of the network into groups that are nested inside. To the best of our current knowledge, therefore, the novelty of this work is twofold. On the one hand, we are utilizing various hierarchical clustering techniques for finding nested subgraphs. On the other hand, we approach the problem from a data mining perspective, that is, we are interested in fast and robust solutions allowing to find not only perfectly nested groups, but subgraphs having large overall nestedness.

The structure of the chapter is the following. In the next section, we introduce the main definitions regarding nestedness and hierarchical clustering. In Section 4.3, we introduce a benchmark generating algorithm and various evaluation metrics in order to be able to systematically test our algorithms. Then, we apply our methods on different real-life bipartite and non-bipartite networks. Finally, in Section 4.4, we draw some conclusions and propose some potential research directions in the topic for the future.

4.2 Nested subgraphs from hierarchical clustering

As in the earlier chapters, we make a clear distinction between the definitions of nested *vertices* and nested *graphs*. As a building block to define the nestedness of graphs, we first define nestedness for a pair of vertices. The amount (or strength) of nestedness between two vertices can be defined using Equation (1.1):

$$\mathrm{nest}(i,j) = \frac{|N(i) \cap N(j)|}{\min\{|N(i)|, |N(j)|\}}.$$

If $\mathrm{nest}(i,j) = 1$, then $N(i) \subseteq N(j)$ or $N(j) \subseteq N(i)$, i.e., the nestedness criterion holds for the given vertices i and j. In the special case of either of the vertices being isolated (where $\min\{|N(i)|, |N(j)|\} = 0$), we consider the vertices non-nested and define $\mathrm{nest}(i,j) = 0$.

Since real-world networks are rarely fully nested (hence the numerous metrics that have been introduced to measure nestedness, e.g., [22, 23, 24]), one might be interested in finding the fully nested subgraphs in them. This is interesting from a theoretical and algorithmic perspective, however, finding only fully nested clusters might result in too many small chunks of the network [56], potentially losing valuable information about its structure. Thus, especially from a data mining perspective, it might be desirable to group together nodes that are *almost* fully nested. One way to

achieve this is by setting a nestedness *threshold level*: a nestedness value from which we consider two nodes or clusters nested, even if they are not fully nested. This approach suffers from the same problems as with any threshold-based method: the arbitrary threshold level has to be set manually beforehand. Even if a threshold does not need to be set, some clustering algorithms rely on a predetermined number of k clusters to find a solution [60], possibly forcing a trial-and-error scenario where multiple k cluster counts need to be evaluated separately in order to find the best one.

Hierarchical clustering provides a more flexible approach: we can view the same clustering at different *resolutions* – from coarse clustering results of only a few large clusters to detailed ones with more, smaller clusters. This not only allows us to choose an optimal clustering, and cluster count, later (e.g., based on a derived metric), but also to view the clustering process as a timeline: nodes that were merged earlier (or divided later, in a top-down scenario) in the process might be more closely related (in our case, more nested) to each other than those that were merged at the end (divided at the beginning). It also removes the limitation of having to choose an arbitrary threshold value in order to get a clustering.

There are multiple ways to perform hierarchical clustering [13, 61]. Most hierarchical clustering algorithms are *agglomerative* (also called bottom-up) or *divisive* (top-down), with some exceptions like [62], which relies on optimization. In the case of *agglomerative* (bottom-up) methods, each node starts in its own cluster, and in each step, two clusters are merged, until all nodes belong to the same cluster. *Divisive* (top-down) methods, on the other hand, start with all nodes being in the same cluster, dividing a cluster into two in each step, until every node is in its own, separate cluster.

In order to distinguish between the different concepts we are going to use, we need to define them. First, we define a cluster (partition) as a nonempty set of vertices, denoted by c_{ij} ($j = 1, 2, \ldots$). An agglomerative clustering (partitioning) level ℓ_i consists of $n - i + 1$ (or i in case of divisive clustering) mutually exclusive clusters: $\ell_i = \{c_{ij} \mid j = 1, 2, \ldots, n - i + 1\}$, where for all $m \neq n : c_{im} \cap c_{in} = \emptyset$ and $\bigcup_j c_{ij} = V(G)$. A hierarchical clustering $\mathcal{C}$ consists of n levels of clustering (partitioning): $\mathcal{C} = \{\ell_i \mid i = 1, 2, \ldots, n\}$. Figure 4.1 illustrates how hierarchies build up to form a clustering.

Since each step (or level) of the algorithm results in a valid clustering, with $n - k$ clusters making up the k-th level ℓ_k, we can choose any one of them as the final clustering structure or draw conclusions from the timeline of the process.

4.2.1 Bottom-up methods

Bottom-up (or agglomerative) hierarchical clustering methods begin with assigning all nodes to their own clusters: $\ell_1 = \{\{1\}, \{2\}, \ldots, \{n\}\}$. These algorithms merge two clusters in each step, until there is only one cluster left. Here, we will work with algorithms that use the pairwise distance of elements (often encoded in a distance matrix), merging the two clusters with the smallest distance

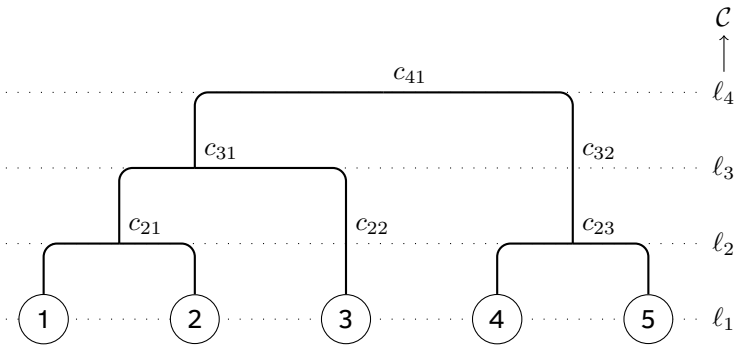


Figure 4.1: *Example dendrogram of hierarchical clustering $\mathcal{C}$ with notations. Agglomerative methods build the tree from the bottom up, while divisive methods start at the top and move downwards.*

in each step. The algorithm takes a distance matrix $\mathbf{D}$ as its input, where, in our case

$$\mathbf{D}_{ij} = 1 - \mathrm{nest}_{ij}. \tag{4.1}$$

The choice of distance definition is important here, as we want to group together highly nested pairs of nodes, and the algorithm will merge clusters with the smallest distance (in this case, with highest nestedness).

Another crucial parameter of the agglomerative method is the *linkage* method. Common linkage methods include single-linkage (which is closely related to minimal spanning trees of graphs [63]), where the distance of two clusters is the minimal pairwise distance between their items (see Equation (1.3)), and complete-linkage, where it is the maximal pairwise distance between cluster items (Equation (1.4)). We can also use average-linkage (Equation (1.5)), where the distance is the average pairwise distance. Additional distance metrics include centroid, median, and Ward clustering [13]. For easier reading, we repeat the definitions of the distances between two clusters, previously defined in Section 1.1. They are calculated as

$$\begin{aligned} d_{\mathrm{SL}}(C_1, C_2) &= \min_{i \in C_1, j \in C_2} D_{ij} \\ d_{\mathrm{CL}}(C_1, C_2) &= \max_{i \in C_1, j \in C_2} D_{ij} \\ d_{\mathrm{AL}}(C_1, C_2) &= \sum_{i \in C_1, j \in C_2} \frac{D_{ij}}{|C_1| \cdot |C_2|}, \end{aligned}$$

in the case of single-linkage, complete-linkage, and average-linkage, respectively. Since each intermediate step provides a valid clustering, we get n cluster levels with $|\ell_i| = n - i + 1$ clusters at the end of the process.

For further reading on the different metrics one can use with hierarchical clustering, we refer to [13].

4.2.2 Top-down methods

In the case of top-down hierarchical clustering, the algorithm starts by assigning all nodes to the same cluster, and then splits a cluster into two at each step, stopping when all nodes are in different clusters. Utilizing the top-down procedure for our purpose, we use a modified version of the algorithm proposed by Girvan and Newman [64]. The algorithm was originally designed for graph clustering. It does this by deleting edges with the highest betweenness (a metric also defined by Girvan and Newman) until the number of components in the graph increases. It repeats this step until each vertex is in its own component. To adapt the algorithm to nestedness clustering, we make two changes:

1. Instead of the original graph G, we use an auxiliary one (denoted G_{nest}). In G_{nest}, there is an edge between each pair of nodes i and j with weight nest_{ij}, but it only exists if this value is non-zero.

2. Instead of simply calculating edge betweenness on G_{nest}, we divide edge betweenness by the edge weight (nest_{ij}), increasing the likelihood for deleting edges with low nestedness values.

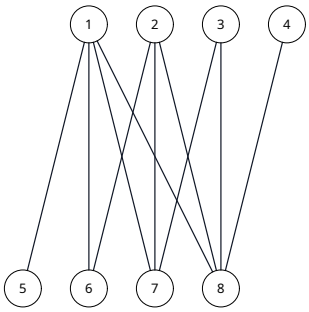
Figure 4.2 shows a fully nested and a randomized bipartite graph (panels a and d), their auxiliary nestedness graphs (panels d and f) and the clustering dendrograms (panels b and e). We can see that the fully nested graph's auxiliary graph forms cliques with all edge weights being 1, while, in the case of the randomized graph, some (i, j) edges are not shown when $\mathrm{nest}_{ij} = 0$.

Since recalculating edge betweenness for all remaining edges in the auxiliary graph in each step is computationally expensive, we have implemented an alternative version of the algorithm (we will refer to it as version "full"). In this version, we first pick the edges with minimal nestedness values, and if more than one edge has the same minimal value, we delete those with maximal edge betweenness. Thus, we only calculate edge betweenness when we need to decide between multiple edges having the same minimal nestedness value, and even then, only for the edges we need to decide between.

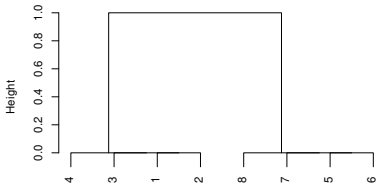
We should note here that the Girvan-Newman algorithm also has a faster method to recalculate edge betweenness, but its implementation is out of scope for this thesis.

4.2.3 Pre-filtering using statistical validation

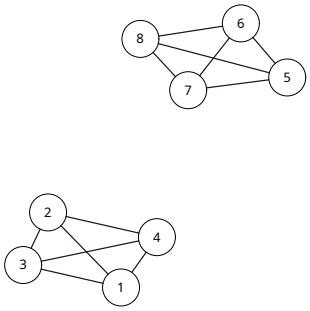
For community detection, several studies have recently chosen a subset of links of the network by applying a statistical test that evaluates a precisely defined null hypothesis [65, 66, 67]. These subsets are commonly referred to as statistically validated networks. A null model is created based on the expected distribution of a specific quantity. The model allows comparison with empirical data, facilitating a statistical test to determine whether the empirical value aligns with the null



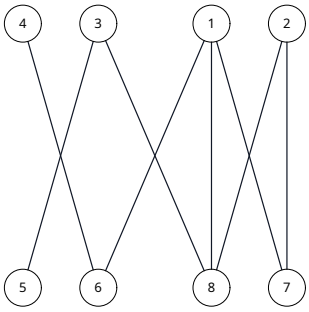
(a) *Original graph – fully nested*



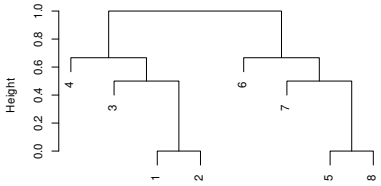
(b) *Clustering dendrogram – bottom up, average-linkage*



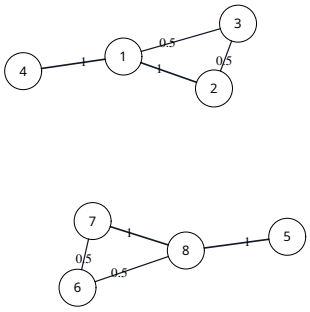
(c) *Nestedness graph – fully nested*



(d) *Original graph – randomized*



(e) *Clustering dendrogram – bottom up, average-linkage*



(f) *Nestedness graph – randomized*

Figure 4.2: *Examples of fully nested (a–c) and randomized (d–f) graphs and their nestedness graphs, along with their clustering dendrogram created using the bottom-up method and average-linkage distance metric. We note that the nestedness graphs are full graphs in all cases, but we don't show edges with zero weights (where* $\mathrm{nest}(i, j) = 0$*), which is why nodes 1, 2 and 4, for example, are not merged into a single cluster at the distance of 0 on panel (e).*

hypothesis or warrants rejection. Rejection of the null hypothesis suggests that the empirical data incorporates aspects not considered in the null model.

As we have seen, the auxiliary nestedness graph contains an edge for every non-zero nestedness relationship between two nodes. To improve the performance of hierarchical clustering algorithms, we can try to filter out some "unnecessary" edges from the nestedness graph. Removing these edges can not only improve performance, as there are fewer of them to consider, but also potentially improve the quality of the clustering by preventing the grouping of non-nested node pairs.

To find edges with low significance, we calculate the probability that the two ends of an edge (i and j) have at least t common neighbors in a null model, where t is the actual number of common neighbors in the network. In other words, we calculate the probability that the number of common neighbors would be as high as it currently is if the graph were a random graph of the same degree sequence. The probability of observing t common neighbors between nodes i and j is given by the hypergeometric distribution

$$H(t \mid n, k_i, k_j) = \frac{\binom{k_i}{t}\binom{n-k_i}{k_j-t}}{\binom{n}{k_j}}. \tag{4.2}$$

If it is unlikely that nodes i and j have at least t common neighbors (its probability is less than, or equal to α), we keep their connection in the auxiliary nestedness graph, otherwise we delete them, as they might as well be random edges. The probability of this can be calculated using the following equation:

$$P(k_{ij} \geq t) = 1 - P(k_{ij} < t) = 1 - \sum_{t=0}^{\min\{k_i,k_j\}-1} H(t \mid n, k_i, k_j). \tag{4.3}$$

Choosing an appropriate α value may prove difficult, as corrections might be needed instead of simply using $\alpha = 0.05$ or $\alpha = 0.01$ [68].

Since we perform the statistical test on all pairs of nodes, we perform a multiple hypothesis test comparison. We should note that multiple hypothesis test comparisons often require a multiple hypothesis test correction to control the amount of false positives (i.e., validated edges that should have been rejected). Such corrections are, for example, the Bonferroni correction or the false discovery rate. In our experiments, we did not see major differences when applying such methods, but a detailed investigation of this is beyond the scope of this chapter.

4.3 Experiments

In this section, we assess both bottom-up and top-down methods in various scenarios. Initially, we use synthetic, randomized bipartite graphs with a known cluster structure. In Section 4.3.1, we introduce an algorithm for generating such graphs while controlling the randomness of the

edges' density within and between clusters. After analyzing the results for synthetic networks, we examine commonly used real-world bipartite networks from the Web of Life database [32]. Finally, we evaluate our algorithms on non-bipartite networks as well. To assess algorithm performance, we introduce several metrics in Section 4.3.2 to characterize each level of hierarchical clustering and use them to select the best clustering level.

4.3.1 Data

In order to test our algorithms, we utilize data from multiple sources for evaluation. Bipartite graphs are obtained from (1) our own generator algorithm (as described in Section 4.3.1), and (2) the host-parasite and pollination datasets from the Web of Life database [32]. Additionally, we evaluate the proposed algorithms on various non-bipartite graphs commonly employed as traditional clustering and community detection benchmarks collected from [69, 70].

Generating benchmark graphs

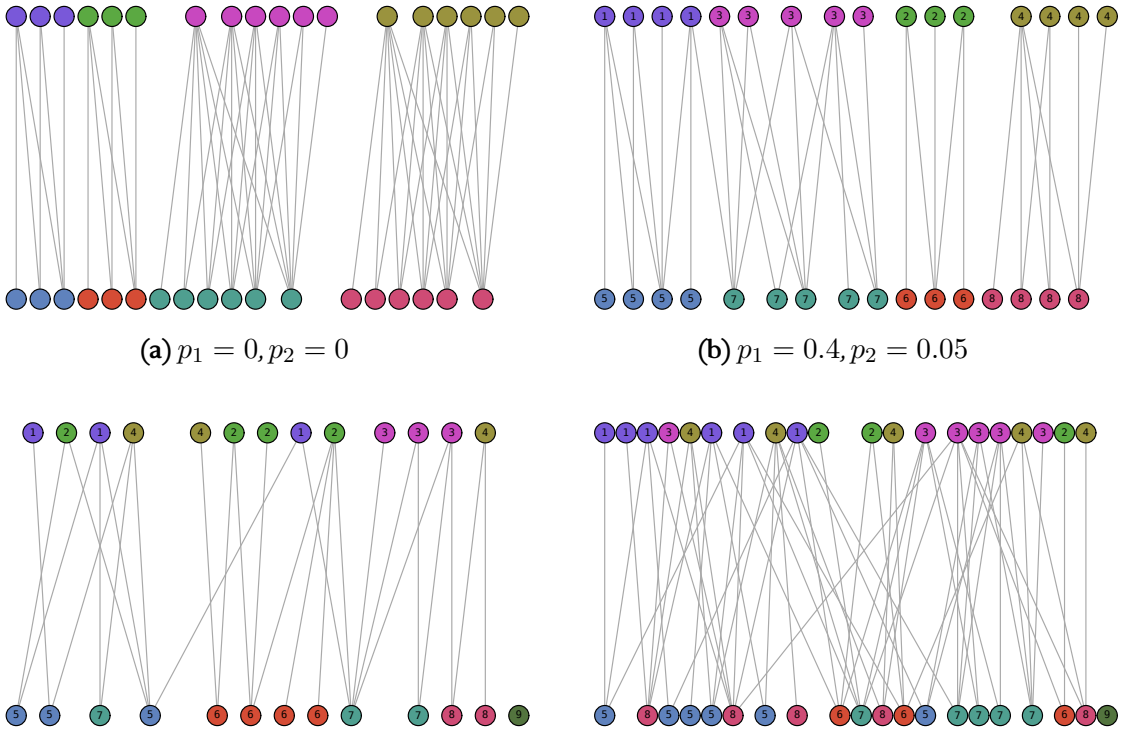
To gain a deeper understanding of the clustering algorithm's stability, we create random bipartite graphs with a known ground truth cluster structure. When generating these random graphs, we initially generate k disjoint, fully nested components, followed by executing two transformations within each component. With a probability of p_1, we rewire edges within the component. Subsequently, with a probability of p_2, we select an edge of a component and relocate one of its ends to another component. Figure 4.3 depicts three benchmark graphs generated using distinct perturbation probabilities. To assess the performance of the algorithms on the generated benchmarks, we compute the mean of the Adjusted Rand index (ARI) for each combination of p_1 and p_2 values.

Real-life data

We utilized two sets of networks from the Web of Life repository for our experiments: the pollinator and the host-parasite networks, both comprising small bipartite networks. The pollinator dataset's graphs contain an average of 89.84 vertices and 175.09 edges, while the host-parasite dataset is even smaller, with an average vertex count of 32.71 and an average edge count of 78.12.

The graphs included in the dataset are not fully nested, with some exceptions. The average NODF score of the pollination networks is 31, while for the host-parasite networks, it is 52. To confirm the low average nestedness, we also measured the networks' nestedness using our nestedness measure, vertex presence, combined with our overlapping community detection algorithm [56]. The mean vertex presence on the pollinator set was 0.19, and on the host-parasite set, it was 0.28.

An overview of some key properties of the two datasets is included in Table 4.1.



(a) $p_1 = 0, p_2 = 0$ **(b)** $p_1 = 0.4, p_2 = 0.05$

(c) $p_1 = 0.05, p_2 = 0.4$ **(d)** $p_1 = 0.3, p_2 = 0.5$

Figure 4.3: *Example benchmark graphs at different perturbation probabilities.*

	Pollinator	Host-parasite
Vertices	89.84	32.71
Edges	175.09	78.12
Density	0.08	0.16
NODF	31	52
Binmatnest	17.7	20.9
Vertex presence	0.19	0.28

Table 4.1: *Summary of the used bipartite network data sets. The NODF and Binmatnest values range from 0 (non-nested) to 100 (fully nested), while vertex presence ranges from 0 (non-nested) to 1 (fully nested).*

4.3.2 Evaluation methodology

Hierarchical clustering provides us with up to n distinct clustering levels for the same graph. However, we can only use a single level from the hierarchy when we want to evaluate clustering performance. To avoid relying on arbitrary threshold values, we can employ a metric to evaluate the clustering quality across different levels and subsequently choose the optimal level based on this metric's value. In this context, we introduce several metrics tailored for this purpose.

In order to standardize how we interpret the amount of clusters and the number of steps across bottom-up and top-down algorithms, we normalize the step count k by the number of vertices, so that we get a value between 0 and 1 ($1/n$ being all vertices being in one cluster, and 1 being $k = n$, or all vertices being in different clusters).

We compute most of the following values from the mean of pairwise nestedness values in each cluster c_i at ℓ. As such, this is a vector of means (of length $|\ell|$):

$$\begin{aligned} \mu_i &= \sum_{j,k \in c_i} \frac{\mathrm{nest}_{jk}}{|c_i|} \\ \mu &= \{\mu_i \mid i = 1, 2, \ldots, |\ell|\}. \end{aligned} \tag{4.4}$$

One such metric is the average nestedness in clusters. We take the mean of all pairwise nestedness values among the nodes of each cluster, then calculate the average of the averages:

$$\overline{\mu} = \frac{\sum_{C_i \in \ell} \mu_i}{|\ell|}. \tag{4.5}$$

Alternatively, we can create a weighted version, weighting each average with the cluster size:

$$\overline{\mu}^w = \frac{\sum_{C_i \in \ell} \left(\sum_{j,k \in C_i} \mathrm{nest}_{jk} \right)}{n}. \tag{4.6}$$

We can also measure the ratio of fully nested clusters in the clustering:

$$R = \frac{|\{C_i \mid \mu_i = 1\}|}{|\ell|}. \tag{4.7}$$

Similar to $\overline{\mu}^w$, the weighted version R^w can be defined.

When a hierarchical clustering algorithm returns n cluster structures, we calculate these metrics on each of them, and then pick the clustering with the best value of the chosen metric g:

$$\ell^* = \arg\max_{i=1}^{n} g(\ell_i). \tag{4.8}$$

The chosen clustering can then be compared to traditional nestedness metrics, outputs of other clustering algorithms or, in the case of benchmark graphs, the ground truth (e.g., by using the Rand index [71]).

4.3.3 Results

In this section, we evaluate the proposed hierarchical clustering algorithms using the previously introduced metrics. First, we measure how well the algorithms can reconstruct the original nested clusters on perturbed synthetic bipartite networks, then we turn to real-world bipartite and non-bipartite networks to evaluate their nested clustering.

Benchmark graphs

On the synthetic graphs, our main metric of comparison was the Adjusted Rand Index (ARI), since we had ground truth clusters available. Our main goal was to combine this with different perturbations to see how much noise the algorithm can handle and how far we can push the noise so that the algorithm is still able to reconstruct most of the original clusters.

When evaluated across different hierarchical clustering algorithms, and across all p_1 and p_2 perturbation probabilities, no single algorithm produced the best ARI value uniformly across all the cases. The same was true for the metrics, too, but in some cases, such as the Girvan-Newman ("nested") algorithm, $\overline{\mu}$ was better in most perturbation cases, shown in Figure 4.4. In the case of the algorithms, as shown in Figure 4.5, the top-down approach produced the best ARI in 48.8% of the cases, of which 48.4% were the full ("full") version. Of the remaining 51.2% for the bottom-up approach, the best performer was the single-linkage clustering in 50.8% of the total cases. It is worth noting that single-linkage clustering performed the best in most cases when p_2 (outer perturbation) was zero and when p_1 (inner perturbation) was large. In the intermediate ranges, the top-down algorithm almost exclusively provided the best ARI index.

Comparing the two top-down algorithms, the one that is closer to the original Girvan-Newman method (labeled version "full") performs better in most cases, as expected, and significantly better in some regions. Note, however, that this increased accuracy comes at the significant performance cost of re-evaluating the edge betweenness for all edges in the graph at every step. Figure 4.6 shows that the biggest advantage in accuracy can be achieved at smaller perturbation levels, mainly in the region $p_1 \leq 0.4$ (inner perturbation) and $0.05 \leq p_2 \leq 0.6$ (outer perturbation). This is seemingly due to the higher sensitivity of version "nested" to outer perturbations. At higher levels of randomness, the version "nested" performed similarly, sometimes slightly better, although in the overall comparison, the bottom-up method (using single-linkage) was better.

Across all algorithms, the bottom-up approach seemed to find a fully nested clustering first, meaning that it also found the largest fully nested clusters, as shown in Figure 4.7. There was no difference between different linkage methods: median, complete-linkage, average-linkage and Ward clustering all found the same size fully nested clustering first. The next best was the "full" version of the top-down algorithm, followed by the "nested" version. The single-linkage bottom-up clustering found a fully nested clustering latest, resulting in lots of small clusters (almost every node being in its own cluster).

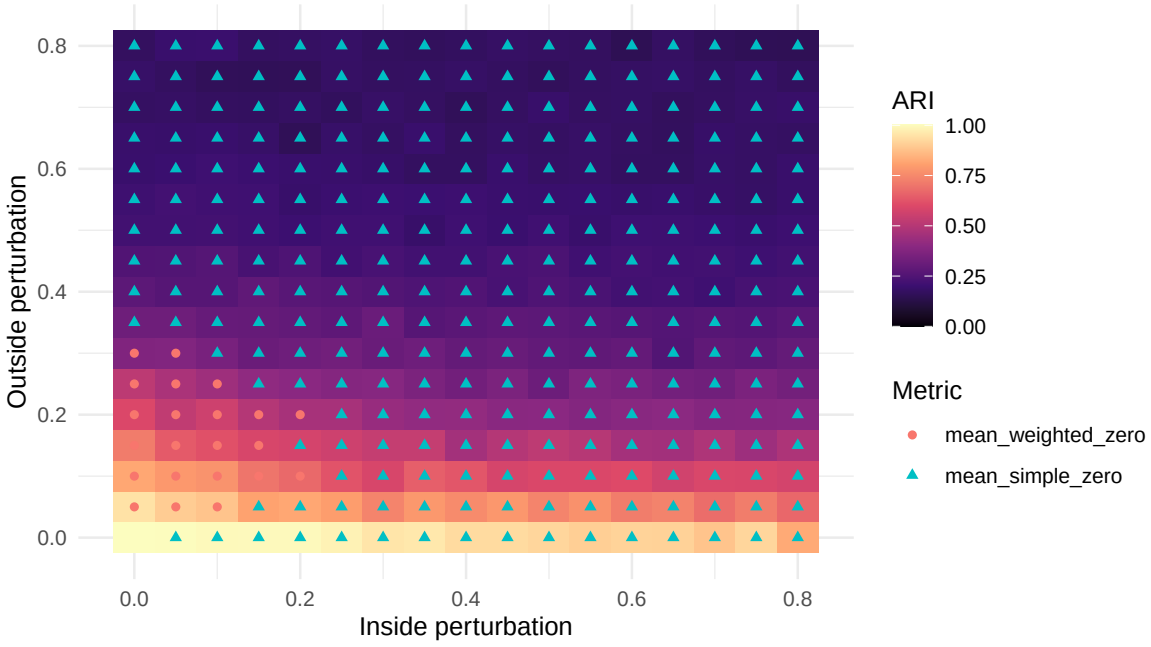


Figure 4.4: *Metrics that gave the best ARI values for each perturbation probability using the Girvan-Newman ("nested") algorithm.*

We have also compared the point at which the various algorithms found the first (and largest) fully nested clustering to the graphs' vertex presence [56] and other nestedness metrics, including NODF [22], the Binmatnest temperature [23] and discrepancy [24]. As shown in Figure 4.8, out of all metrics, vertex presence showed the highest correlation with the first point at which the algorithms reached a fully nested clustering. In this comparison, the highest correlation was between "full" version of the top-down algorithm, with a correlation of -0.9018. The next highest correlation was produced by the bottom-up algorithm using the average-linkage (and complete-linkage, median linkage and Ward clustering) with a correlation coefficient of -0.8817. While the correlations are not high enough to be considered significant, a link between the stop point and a nestedness metric indicates that the stop point might also be useful for measuring nestedness.

Bipartite graphs

Now we turn to the bipartite Web of Life dataset, which contains fully nested and nearly fully nested graphs. Figure 4.9 shows that, in our experiments, the bottom-up approach usually reached a fully nested clustering in fewer steps (with higher cluster counts) than the top-down algorithms. Here, the "nested" version of the top-down algorithm found larger clusters this time, compared to the results on the synthetic networks, where it found smaller fully nested clusters on average. This also means that different algorithms reach a fully nested clustering first in different steps, cluster

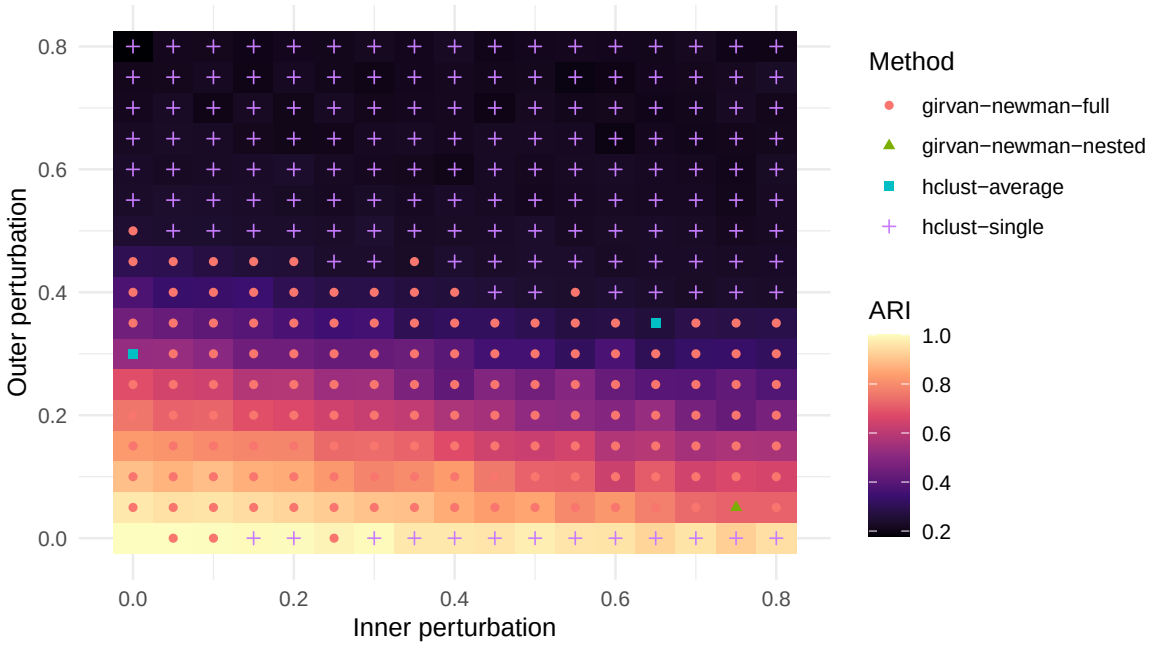


Figure 4.5: *Best method based on ARI values for each p_1 and p_2 value on the synthetic networks.*

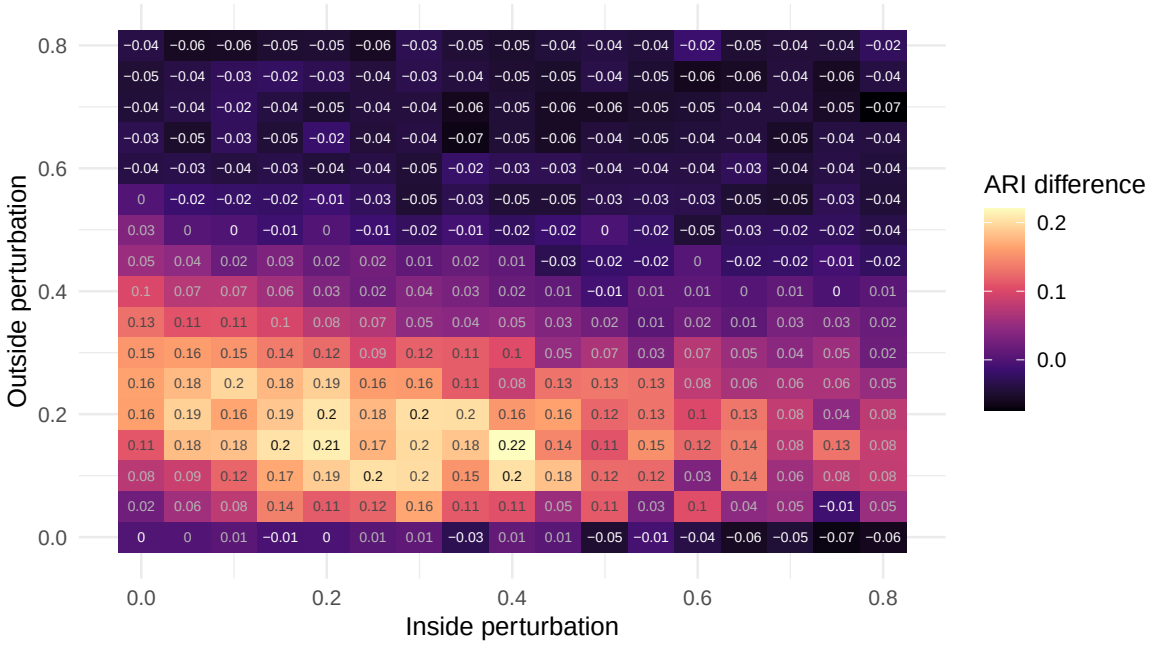
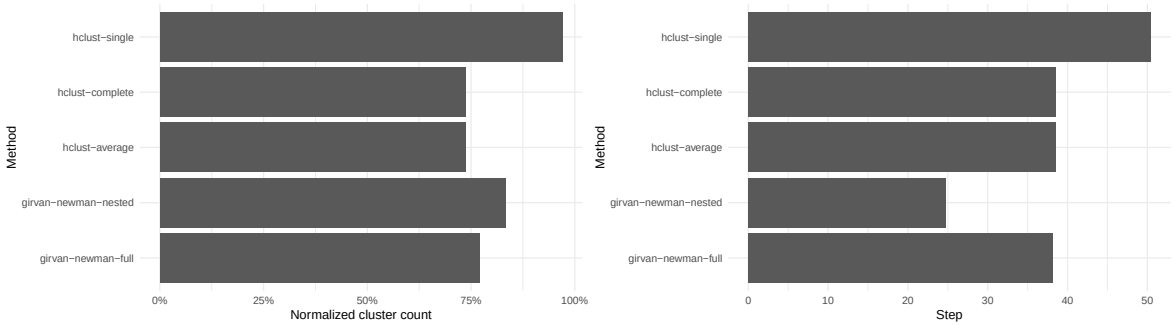
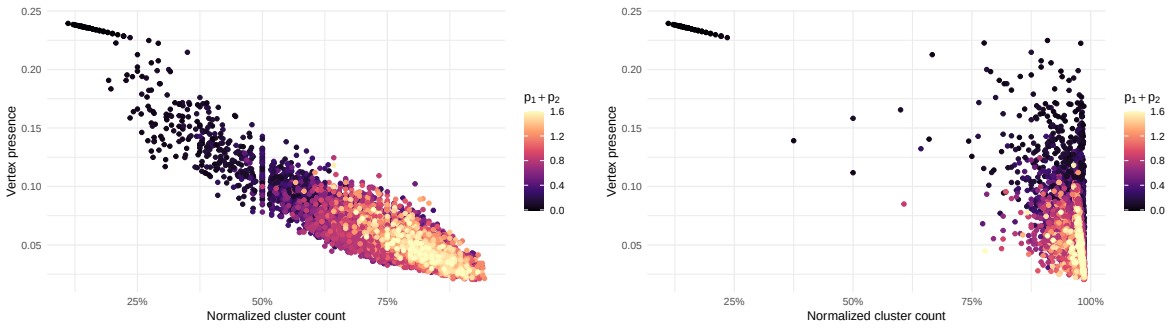


Figure 4.6: *Difference in ARI between top-down algorithms "full" and "nested". The score is* $\mathrm{ARI}(\mathrm{full}) - \mathrm{ARI}(\mathrm{nested})$.



(a) (b)

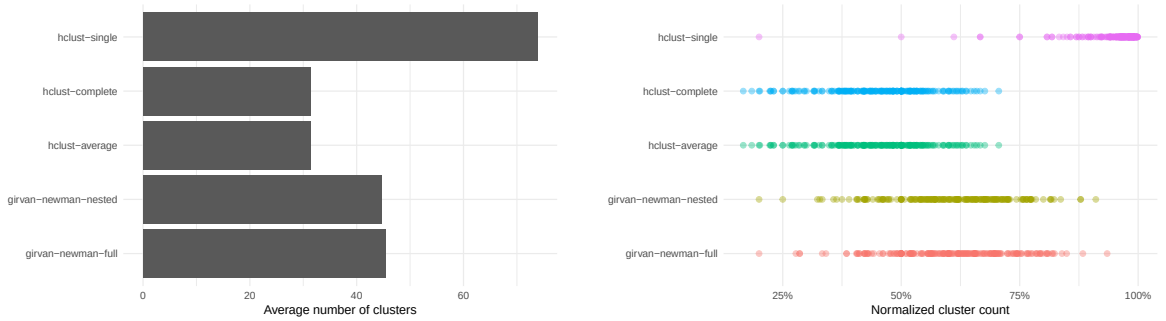
Figure 4.7: *Average state to first reach a fully nested clustering for each algorithm on the synthetic networks. The bottom-up algorithm's average- and complete-linkage variants found the largest fully nested clustering. The "nested" version of the top-down algorithm performed much fewer iterations to reach a similar sized, fully nested clustering as the "full" variant.*



(a) *Using the top-down algorithm's "full" version. The Pearson correlation coefficient is -0.9018.*

(b) *Using the bottom-up algorithm with single-linkage. The Pearson correlation coefficient is -0.5863.*

Figure 4.8: *Relationship between the largest fully nested cluster's point (normalized step count) and the mean vertex presence of the synthetic networks. Point colors show the cumulative permutation* $(p_1 + p_2)$.



(a) *Average number of clusters to first reach a fully nested clustering, for each algorithm, on the Web of Life database.*

(b) *Distribution of normalized cluster counts to reach first fully nested clustering, for each algorithm, on the Web of Life database.*

Figure 4.9: *First steps to reach a fully nested clustering with different algorithms on the Web of Life dataset.*

counts, and ultimately, configurations.

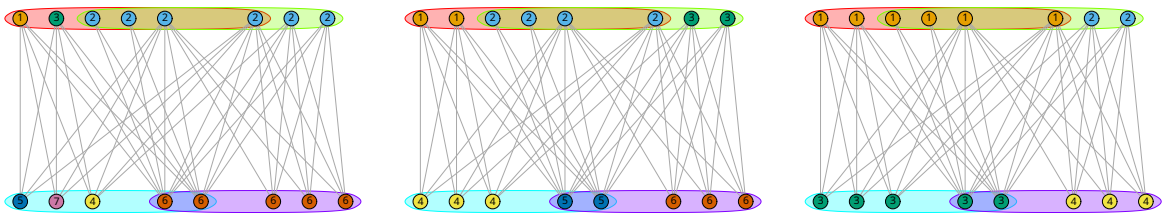
Figure 4.10 shows this by comparing the first fully nested clustering level of different algorithms with the overlapping fully nested communities detected by our previous algorithm, on the `M_PL_070` pollination graph. This example also highlights one disadvantage of clustering over overlapping community detection, as multiple fully nested cluster structures could be realized over the highlighted graph.

Interestingly, there does not seem to be a strong relationship between the point of the first fully nested clustering and the various nestedness metrics on this dataset. The strongest correlation coefficient of -0.6043 was with the "nested" version of the top-down algorithm. This is due to graphs like `M_PL_021`, where there are numerous vertices with the same neighborhood – for example, 136 nodes out of 768 have the same single neighbor. This causes the clustering algorithm to reach a fully nested clustering early (at 22% cluster size), but the graph has a larger number of overlapping nested communities, driving vertex presence down. Despite this, the stopping point may still be useful as another nestedness metric.

Generally, we have seen that the average-linkage method was ahead of the others in average nestedness, especially when we weigh the means with cluster size. An example of this can be seen on Figure 4.11, while the general trends are visible on Figure 4.12. This also shows that the "nested" version of the top-down clustering algorithm generally increased nestedness in a slower pace from step to step, compared to the "full" one.

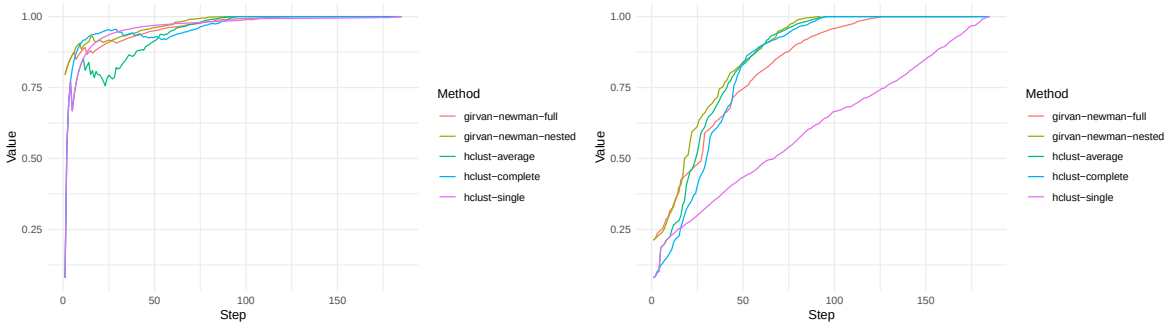
Non-bipartite graphs

Our results on non-bipartite graphs mirror those on bipartite graphs. Figure 4.13a shows the time it took each algorithm to reach a fully nested clustering first on each network analyzed, revealing



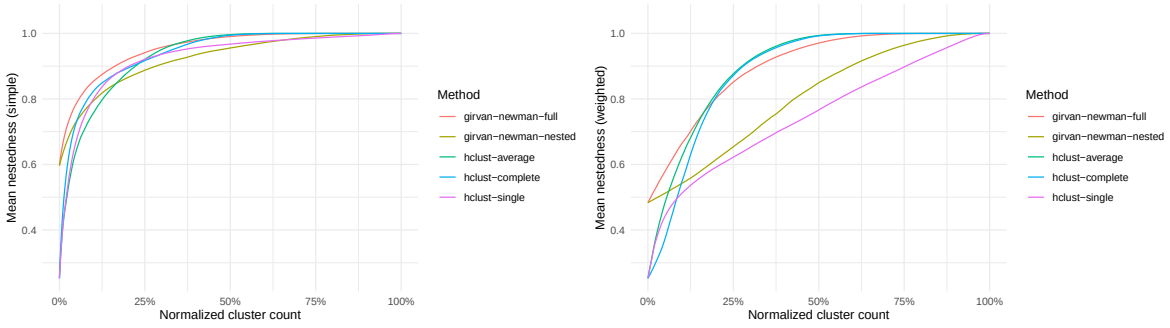
(a) *Top-down "full", step 6* (b) *Top-down "nested", step 3* (c) *Bottom-up, average-linkage, step 4*

Figure 4.10: *First step to reach fully nested clusters with different algorithms on the* `M_PL_070` *pollination graph. The outlines show the overlapping fully nested communities detected by our previous algorithm.*



(a) *Simple means* ($\overline{\mu}$) (b) *Weighted means* ($\overline{\mu}_w$)

Figure 4.11: *Mean cluster nestedness values on the* `M_PL_001` *pollination graph for the various clustering algorithms.*



(a) *Mean of simple means* ($\overline{\mu}$) (b) *Mean of weighted means* ($\overline{\mu}_w$)

Figure 4.12: *Average of mean nestedness values across all graphs of the Web of Life dataset in the function of normalized cluster counts, for all clustering algorithms.*

graph	$\|V\|$	$\|E\|$	ρ	$\overline{d}$	pres	cl. coeff.
families	15	20	0.1905	2.6667	0.2033	0.1915
ia-southernwomen	18	75	0.2451	8.3333	0.3804	0.6302
mammalia-proximity	24	305	0.5525	25.4167	0.4750	0.8574
mammalia-association	25	368	0.6133	29.4400	0.7992	0.8909
johnson8-2-4	28	210	0.2778	15.0000	0.0370	0.4286
karate	34	78	0.1390	4.5882	0.1901	0.2557
dolphins	62	159	0.0841	5.1290	0.0517	0.3088
eco-everglades	69	911	0.1942	26.4058	0.0385	0.4612
les_miserables	77	254	0.0868	6.5974	0.1389	0.4989
adjnoun	112	425	0.0684	7.5893	0.0373	0.1569
ia-infect-hyper	113	2196	0.1735	38.8673	0.0282	0.4952
email-enron-only	143	623	0.0307	8.7133	0.0218	0.3591
internet-partnerships	219	630	0.0132	5.7534	0.0351	0.1070
celegansneural	297	2345	0.0267	15.7912	0.0364	0.1941
USAir97	332	2126	0.0193	12.8072	0.0422	0.3964
662_bus	662	906	0.0021	2.7372	0.0030	0.0769
ia-crime-moreno	829	1474	0.0021	3.5561	0.0043	0.0076
DD199	841	1902	0.0027	4.5232	0.0023	0.4677
gene	1103	1672	0.0014	3.0317	0.0041	0.3172
inf-euroroad	1174	1417	0.0010	2.4140	0.0014	0.0339
netscience	1589	2742	0.0022	3.4512	0.0036	0.6934

Table 4.2: *Overview of the non-bipartite dataset.* ρ *denotes the edge density, and* $\overline{d}$ *denotes the mean degree, and* pres *denotes the vertex presence.*

that the bottom-up method combined with complete-linkage or average-linkage found a fully nested clustering first in all cases. The top-down algorithms performed very similarly, with one method overtaking the other in some cases. The averages are shown in Figure 4.13b. Single-linkage bottom-up clustering found fully nested clusters last, as expected. Figure 4.13c highlights that while the two top-down algorithms retrieved clusters of similar size, the "full" version achieved that in much fewer steps.

Figure 4.14 shows an example of fully nested clustering of Zachary's karate club network using the average-linking bottom-up method in step 16. The overlapping groups encode the overlapping fully nested communities, while the vertex colors indicate the fully nested clustering. However, the nestedness of the graph is low, as indicated by the many small overlapping groups and the large number of clusters.

Another, although more nested, example is the primate association network shown on Figure 4.15. The vertex presence of the network is 0.799, indicating a high level of nestedness, and this can be seen from the large, overlapping groups (as highlighted using the overlapping nested community detection algorithm). The clustering, where the average-linkage bottom-up method

found one large cluster in the network, and more tiny clusters. We can see that the clustering picks the vertices of one of the overlapping groups and assigns the rest to separate clusters.

The graphs examined show little nestedness overall (as seen in Table 4.2), which may make nested clustering less meaningful than overlapping nested communities.

This is reinforced by the average of the mean nestedness values in each step on Figure 4.16. We can see that the increase of nestedness is less steep than on the Web of Life networks (Figure 4.12), but the trends were similar, except for the "nested" version of the top-down algorithm increasing nestedness slower than the single-linkage bottom-up clustering method. We note that the values on this figure show a stark difference compared to Figure 4.9, since a value of 1 here means the step by which *all* clusterings were fully nested, as opposed to the average point of the first fully nested clustering.

Choice of linkage method in bottom-up algorithms

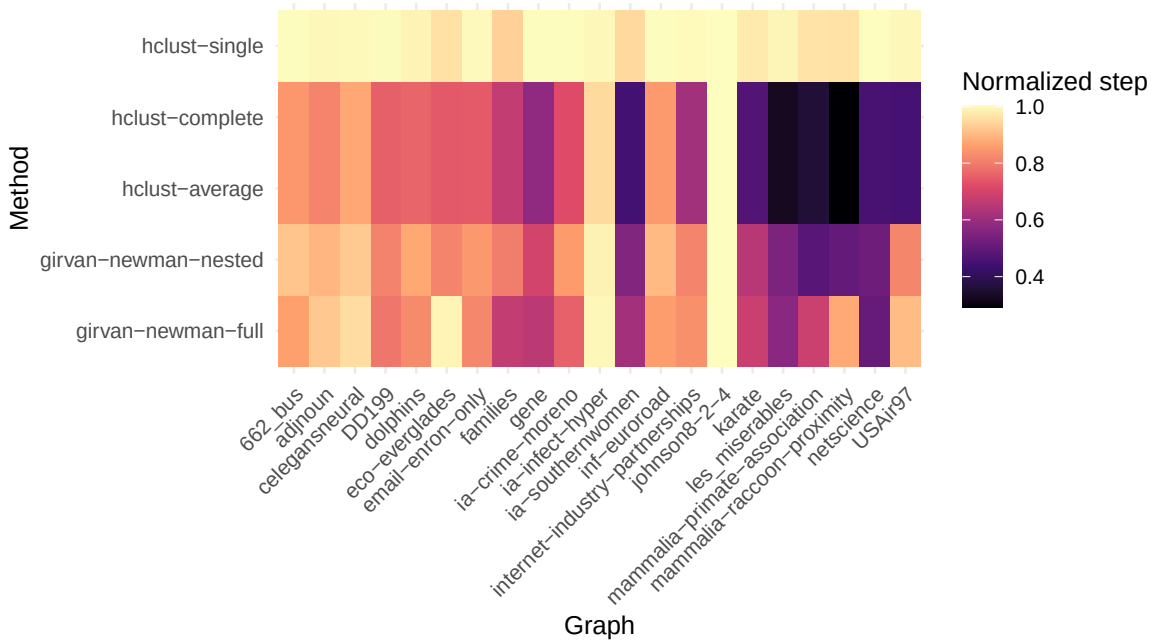
The linkage method is a key parameter of the bottom-up clustering algorithm, as it determines the distance (and therefore the order) of the clusters that are merged. Using the linkage method, we can control the type of clusters the algorithm prioritizes. Since we use $1 - \mathrm{nest}_{ij}$ as our distance metric, the algorithm will generally merge clusters with higher nestedness (low distance). We will now take a look at how each linkage method affects the created clusters.

If we aim to merge fully nested clusters first (where each pair of vertices is fully nested), then we may pick *complete linkage*. In this case, the distance of two clusters is the maximum pairwise distance between the two clusters, i.e., the lowest pairwise nestedness value among clusters determines the distance. This makes it a relatively strict merging strategy, focusing on merging clusters with the highest minimum nestedness value.

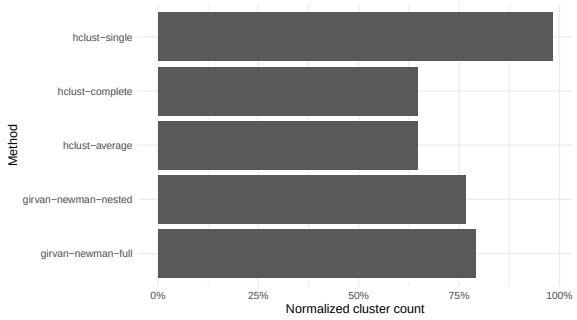
In the case of *single-linkage*, the distance of two clusters is equal to the minimum pairwise distance of their elements (highest pairwise nestedness value). Thus, in this case, the algorithm will try to merge clusters that have at least one highly nested vertex pair among each other.

As a less extreme solution, we can also use *average-linkage*, where the distance of two clusters is their average pairwise distance. This will make the algorithm aim to merge clusters with the highest average pairwise nestedness among them.

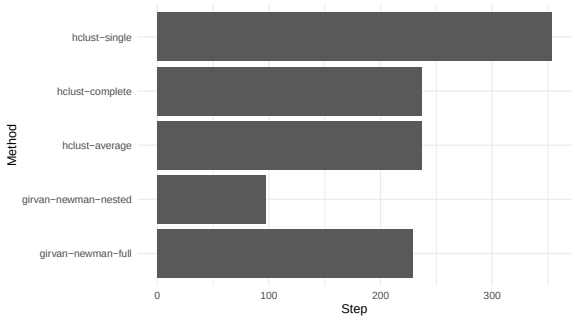
The different linkage methods we compared (except single-linkage) always found the same fully nested clustering first. This is because single-linkage clustering uses the pairwise minimum distance between two clusters, which can cause two non-nested clusters to merge at a distance of 0 if they had a single pair of fully nested vertices between them (e.g., because of overlapping nestedness).



(a) *First time to reach a fully nested clustering for each graph and method*



(b) *Average statistics across all non-bipartite graphs*



(c) *Average step counts for each algorithm's largest fully nested clustering. Here, comparisons are only meaningful inside algorithm families.*

Figure 4.13: *First time to reach a fully nested clustering on the non-bipartite networks, across various algorithms. One key highlight is that the "nested" top-down algorithm achieves a similar performance in much fewer steps.*

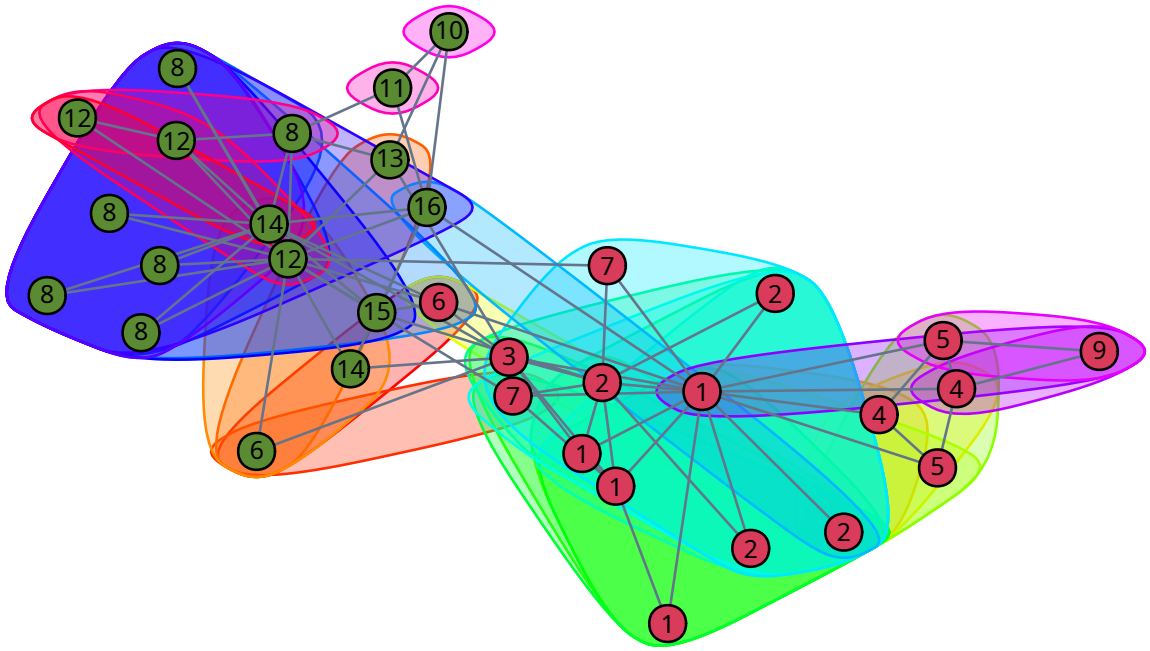


Figure 4.14: *First fully nested clustering on Zachary's karate club network using the bottom-up approach with average-linkage. Vertex labels encode the nested clusters the algorithm found, vertex colors encode the two groups the club split into, and the colored overlapping groups mark overlapping nested communities.*

4.4 Conclusions

In this chapter, we have presented several approaches for performing hierarchical clustering for nestedness on both bipartite and non-bipartite graphs. Across our methods, we have demonstrated a customizable framework for adapting bottom-up and top-down clustering methods for nestedness. We have also introduced a streamlined way to select optimal clustering levels from a clustering hierarchy. To this end, we have also introduced different metrics to measure the performance of each hierarchical level. We have found that the size-weighted mean nestedness ($\overline{\mu}_w$) is more suitable for evaluating clustering in progress, as it is more stable and close to monotonic (with at most a small deviation) on all graphs we tested.

We have evaluated our algorithms across synthetic bipartite and real bipartite and non-bipartite networks. In our experiments, the bottom-up algorithms using the average-linkage and complete-linkage distance methods found the largest fully nested clusters across all datasets.

Both algorithm families can be further enhanced: the bottom-up methods have a replaceable distance metric (for which we used $\mathrm{nest}(i, j)$ – Equation (1.1)) and linkage method, while the top-down algorithm's split condition can be swapped to change what the algorithm prioritizes. We have also made a reference implementation of our algorithms available as open-source code[1].

[1] The implementation (in R) is available at `https://github.com/Hanziness/r-nested-comms/tree/v0.3`

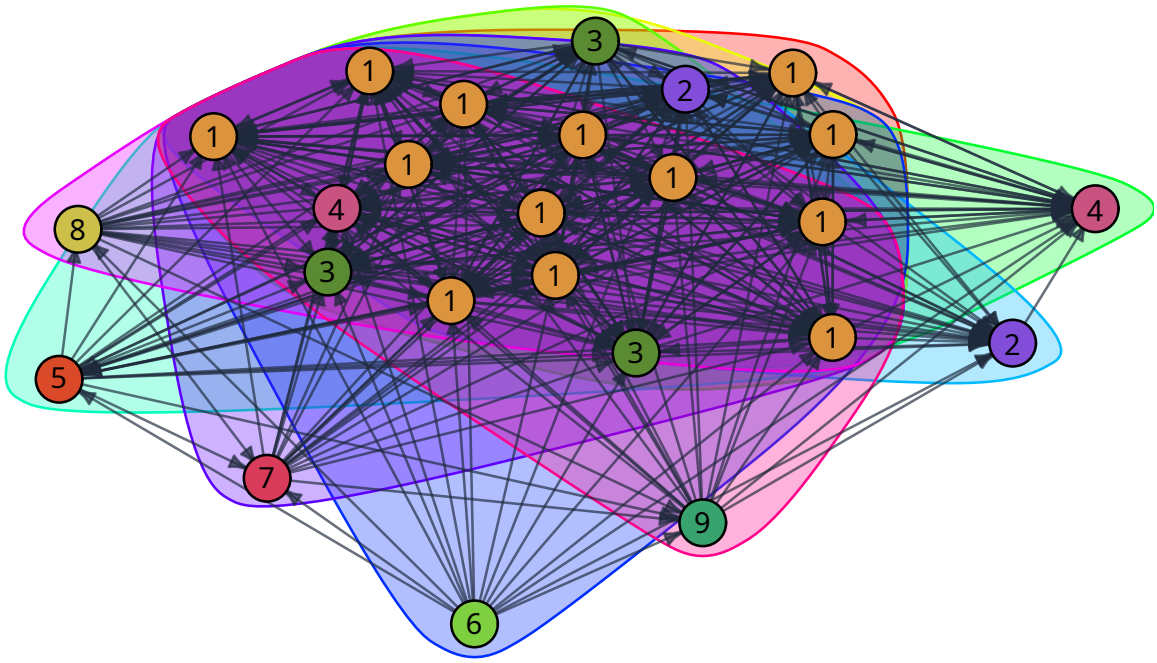
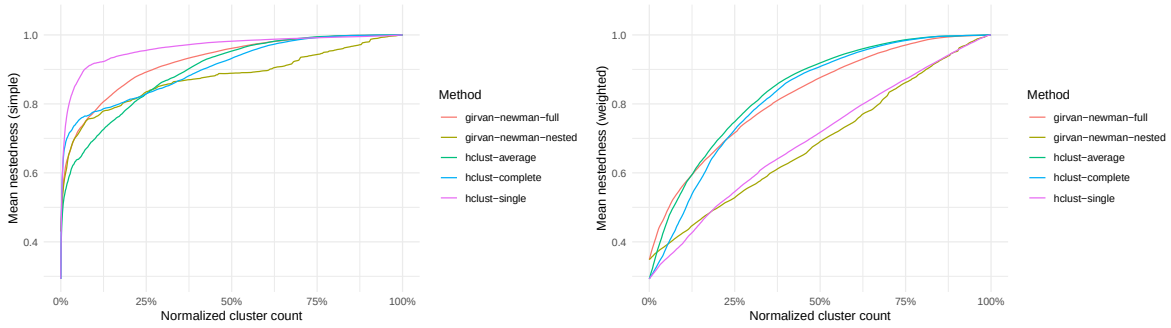


Figure 4.15: *First fully nested clustering on the primate association network [69] using the bottom-up approach with average-linkage. Vertex labels and colors encode the nested clusters the algorithm found, and the colored overlapping groups mark all the overlapping nested communities.*



(a) *Mean of simple means ($\overline{\mu}$)* (b) *Mean of weighted means ($\overline{\mu}_w$)*

Figure 4.16: *Average of mean nestedness values across all non-bipartite graphs in the function of normalized cluster counts, for all clustering algorithms.*

Chapter 5

Hierarchical clustering in portfolio optimization

In this chapter, I am going to demonstrate applications for hierarchical clustering in a different field. Portfolio selection is a widely researched topic that has drawn on the knowledge of several other fields, including network science [72]. It is also not just a complex problem in itself, but a game played by many people, usually with the goal of making high profits and avoiding losses. As such, we do not have a perfect solution, but the problem provides ample opportunity to create new methods that are better in one or more aspects than the others, for example, by creating less risky portfolios. There is a vast amount of historical data on past stock prices, which allows researchers to run large-scale simulations using their own methods. The methods developed for portfolio selection are interesting not only because they use a large set of tools to solve the same problem, but also because existing methods can be improved. Our primary idea here was also to improve upon the existing Markowitz model [73], one of the earliest portfolio selection models that uses a covariance matrix as its key element for optimization. It has since been shown that the model is not perfect [74], but we still wanted to demonstrate that older (and simpler) methods have room for improvement.

The motivation for working with the Markowitz model is that covariance matrices can be treated as adjacency matrices of graphs, where the nodes are the stocks and the weighted edges among them are the covariances. Once we work with graphs, we can apply the tools of network science. Community structures are meaningful in these networks as well, since investing in a set of highly correlated stocks (or, in other words, stocks in the same community) can lead to increased risk. Imagine that two stocks are highly correlated. If the price of one stock falls, then due to high correlations, the price of the other stock is likely to fall as well. The technique that reduces such risks by investing in stocks whose prices correlate the least is called diversification, but in order to apply it properly, we need to identify groups of stocks that are highly correlated. This is the idea that was applied in the algorithms presented by Raffinot [75].

Another approach is to transform (filter) the covariance matrix used by the Markowitz model to improve its estimations. One way to do this is to construct a minimal spanning tree, a concept closely related to single-linkage hierarchical clustering, and use the cluster distances in the tree to transform the covariance matrix. I followed this idea and employed hierarchical clustering to filter the covariance matrix used by the Markowitz model, and also compared the results to methods that rely on hierarchical clustering itself to construct the portfolio, such as the aforementioned model by Raffinot. The key ideas are the same as before (to find closely related items), but our goals here are the opposite: to avoid investing in items that are in the same cluster.

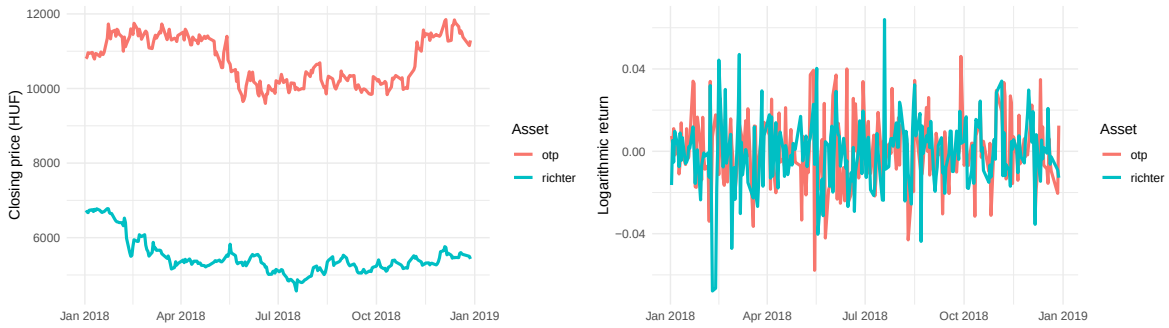
5.1 Introduction

Traditional portfolio selection approaches measure the risk of a portfolio by the standard deviation of its expected returns, which we would like to minimize. In practice, these methods perform well only when the covariance (or correlation) matrix constructed from the asset returns is properly cleaned. The statistical uncertainty (or noise) introduced by the covariance estimation is due to the fact that the number of observations (T) is not large enough relative to the number of assets (n). This is a common case of the curse of dimensionality, as T usually needs to be small due to the non-stationary nature of the correlation between returns [76]. A small T is also often preferred when we only want to consider the recent past of the market. There have existed solutions to this problem for some time, such as [72, 77], and a review has also been published in the topic [78]. These methods can be used to improve the performance of the original Markowitz portfolio model [79]. A good filtering method should preserve the structural part of the covariance matrix that is stable and contains actual information. Therefore, the filtering method that works best depends on the given stock market, and the characteristics of the assets under study.

According to a variety of observations, most complex systems, including the stock market, follow a hierarchical structure [80]. A plausible approach is to define this structure using a hierarchical clustering procedure. In this case, we can assume that the (hierarchical) dependencies between the returns can be described using a dendrogram [81]. In this chapter, we present possible applications of hierarchical clustering to the portfolio selection problem: on the one hand, we use it as a covariance matrix filtering procedure, and on the other hand, we use it directly to compose a portfolio, bypassing the need to solve the optimization problem. We perform our experiments on real data and compare the performance of the different methods using several metrics.

5.1.1 The portfolio selection problem

In the portfolio selection problem, an investor wants to allocate their capital among the stocks available in the market. The primary objective is to minimize the risk of the investment while achieving at least an expected level of return, or the highest possible return for a given level of risk. Typically, the solution is based on historical data for a given set of stocks (e.g., the Budapest Stock



(a) *Closing prices* (b) *Logarithmic returns*

Figure 5.1: *Daily closing prices (a) and logarithmic returns (b) of OTP and Richter on the Budapest Stock Exchange (BSE) in 2018.*

Exchange), which contains the prices of individual stocks over any period of time. Here, we work with daily closing prices, but it is not uncommon to work with smaller (e.g., weekly) and larger (e.g., hourly) resolutions. Logarithmic returns are often used instead of the raw daily closing prices. The logarithmic return of the i-th stock at time t is

$$x_{i,t} = \log \frac{P_i(t)}{P_i(t-1)} = \log P_i(t) - \log P_i(t-1), \tag{5.1}$$

where $P_i(t)$ denotes the closing price of the i-th stock on the t-th day. The main reason for using logarithmic returns is that these returns are time-additive (i.e., $x_{i,t} + x_{i,t+1} + \cdots + x_{i,t+j} = \log P_i(t+j) - \log P_i(t)$) and if the 1-period logarithmic returns are normally distributed (a common assumption that holds for our data), then the j-period returns will be as well. A comparison of daily closing prices and logarithmic returns is shown on Figure 5.1. We will therefore use these instead of the closing prices in our analysis. To obtain the average returns of the i-th stock in a time series of length T, we calculate

$$\overline{x}_i = \frac{1}{T} \sum_{t=1}^{T} x_i(t). \tag{5.2}$$

The covariance between stocks i and j can be defined using the following equation:

$$\sigma_{ij}^2 = \frac{1}{T-1} \sum_{t=1}^{T} (x_i(t) - \overline{x}_i)(x_j(t) - \overline{x}_j). \tag{5.3}$$

By ordering these elements into a matrix, we obtain the $\mathbf{\Sigma} = (\sigma_{ij}^2)_{i,j}$ covariance matrix, which contains the pairwise covariance of the time series of returns of all stocks, as well as the variance of the returns of each stock in the diagonal. The correlation matrix $\mathbf{C}$ is obtained from the covariance matrix through the normalization $C_{ij} := \sigma_{ij} / \sqrt{\sigma_{ii}\sigma_{jj}}$. It is important to note that the value of the

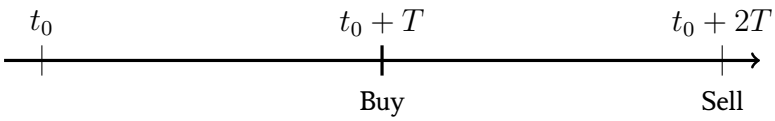


Figure 5.2: *Timeline of an investment in our experiments.*

correlation is not invariant to the logarithmic transformation of the returns, so normal returns are usually assumed for portfolio selection models [82]. However, in the case of real daily returns, the difference between normal and logarithmic returns is negligible, and we preferred to use logarithmic returns because of their previously mentioned advantages.

We assume the following situation for the rest of this chapter. An investor always decides on their portfolio on a given day, taking into account the prices of the previous T days. Risk analysis and diversification take place at this point, and so does the investment. The investor then waits another T days to evaluate their actual ("realized") returns and risks. The investment timeline of our simulations is visible in Figure 5.2.

5.2 Hierarchical clustering in the Markowitz model

A primary tool for risk reduction is diversification, or risk-sharing, which means investing in more than one security at a time. This method is intuitively motivated by the fact that if the returns of the securities we hold are not correlated, then when the price of one security falls, it is not inevitable that the price of the others will also fall, thus reducing the risk of an overall loss. The first portfolio optimization model to incorporate diversification was formulated by Harry Markowitz [73]. Since the definitions, and hence the measurement, of expected returns and risks are not clear, these two measures are determined either by the model's methodology (e.g., in the case of the *Markowitz model*) or by the investor (as in the case of the *Hierarchical Equal Risk Contribution* (HERC) model [83]).

The original Markowitz model defines the expected return of a portfolio as the weighted average of the mean returns of the assets, and the risk is defined as the variance of the portfolio's return. Markowitz showed that, given an upper bound on the risk or a lower bound the return, the optimal portfolio can be obtained by solving a convex quadratic programming problem. Here, we give a lower bound on the return, i.e., the minimum return acceptable to the investor (denoted by R) is the parameter. The quadratic programming problem to be solved is as follows:

$$\begin{aligned} \min_{\mathbf{p}} \quad & \mathbf{p}\boldsymbol{\Sigma}\mathbf{p}^{\intercal} \\ \text{such that} \quad & \sum_{i=1}^{n} p_i = 1 \\ & \sum_{i=1}^{n} p_i \overline{r}_i \geq R, \end{aligned} \tag{5.4}$$

where $\overline{r}_i$ is the estimated return of stock i ($i = 1, \ldots, n$) according to some statistical estimate, such as the average return ($\overline{r}_i = \overline{x}_i$) (see Equation (5.2)), and R is the minimum expected return. The technical condition $\sum_i p_i = 1$ ensures that exactly 100% of the capital is invested. The solution to this problem is a weight vector $\mathbf{p}$, which contains the proportions of capital to be invested in each stock. Note that negative weights are allowed, i.e., *short selling* is also possible. When solving, we further constrain the problem with $-1 \leq p_i \leq 1$. This helps to avoid less realistic situations where in the optimal solution we would have to invest a multiple of our capital in one stock (e.g., through loans).

5.2.1 Filtering covariance matrices

The Markowitz model has multiple limitations. For example, it can be shown that the resulting solution (the optimal portfolio) is highly sensitive to changes in the task's parameters. The quadratic optimization problem also requires the inversion of the covariance matrix $\boldsymbol{\Sigma}$ of the returns, which can lead to unstable solutions for a near-singular matrix. Furthermore, since we estimate covariance from a finite (T-long) time series, the non-stationary nature of the covariance between the returns means that the covariance matrix often carries statistical uncertainty (or simply put, noise), which can lead to poor risk estimation [84]. The latter is addressed by various filtering techniques such as random matrix theory (RMT) [78] or hierarchical clustering [72]. These methods result in a cleaned (or "filtered") covariance matrix $\boldsymbol{\Sigma}_f$, that can be used in the Markowitz model's objective function instead of the original $\boldsymbol{\Sigma}$.

5.2.2 Hierarchical clustering as a filtering technique

A covariance matrix $\boldsymbol{\Sigma}$ created from n stocks can be interpreted as an adjacency matrix of a complete graph of n vertices. The same is true for the correlation matrix $\mathbf{C} = (C_{ij})_{i,j}$, which contains the normalized covariance. We will use the latter for the filtering procedure. In the first step, we construct a distance matrix from the correlation matrix. We do this following [72, 81], with the help of the

$$D_{ij} = \sqrt{2(1 - C_{ij})} \tag{5.5}$$

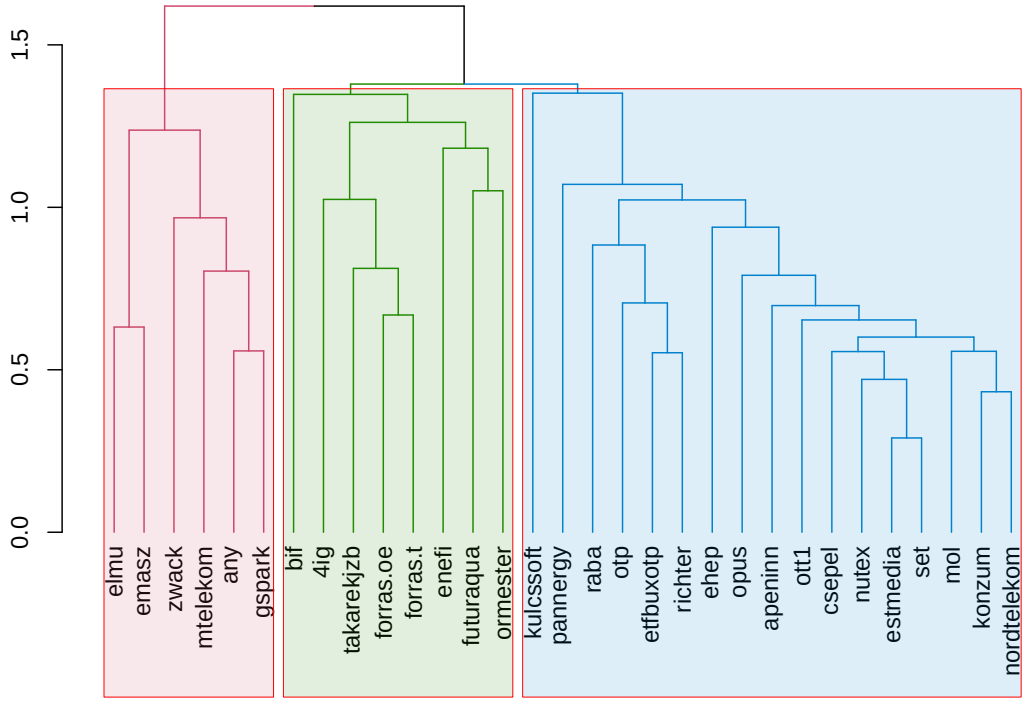


Figure 5.3: *Merge tree of a hierarchical clustering represented as a dendrogram, colored using* $k = 3$ *colors.*

ultrametric distance definition (in which the $D_{ij} \leq \max\{D_{ik}, D_{kj}\}(\forall i, j, k)$ condition is true), converting large correlations into small distances. We then perform hierarchical clustering on $\mathbf{D} = (D_{ij})_{i,j}$.

For clustering, we use an agglomerative method, which initially assigns each vertex to a separate cluster, then merges the two clusters with the smallest distance into a new cluster, repeating this process until there is only a single cluster containing all the stocks [85]. A crucial question is defining the distance between two clusters, on the bases of which the merging is performed. As we have seen in Section 1.1, popular distance metrics include *single-linkage*, which is closely related to minimal spanning trees, *complete-linkage* and *average-linkage*, which define the distance between a cluster $\mathcal{C}_1$ and $\mathcal{C}_2$ as follows:

$$
\begin{aligned}
d_{\mathrm{SL}}(\mathcal{C}_1, \mathcal{C}_2) &= \min_{i \in \mathcal{C}_1, j \in \mathcal{C}_2} D_{ij}, \\
d_{\mathrm{CL}}(\mathcal{C}_1, \mathcal{C}_2) &= \max_{i \in \mathcal{C}_1, j \in \mathcal{C}_2} D_{ij}, \\
d_{\mathrm{AL}}(\mathcal{C}_1, \mathcal{C}_2) &= \frac{1}{|\mathcal{C}_1| \cdot |\mathcal{C}_2|} \sum_{i \in \mathcal{C}_2} \sum_{j \in \mathcal{C}_2} D_{ij}.
\end{aligned}
$$

The algorithm produces a binary merge tree that stores which vertices or clusters were merged and at what distance. The leaves of the tree are the individual vertices (stocks) and the root is the cluster containing all vertices. This tree can be displayed as a dendrogram (see Figure 5.3), or it can be cut at a chosen hierarchy level to obtain exactly k clusters ($k = 3$ in the example shown in the figure).

We can take advantage of an important property of the clustering result: we have built a tree from our stocks using $n - 1$ merges. We can then replace our previous distance matrix $\mathbf{D}$ with the matrix $\mathbf{D}_{\mathrm{hclust}}$ containing the clustering distances with a total of $n - 1$ different values, and then use Equation (5.5) to transform it into a correlation matrix $\mathbf{C}_{\mathrm{hclust}}$, and finally convert it back into a $\mathbf{\Sigma}_{\mathrm{hclust}}$ covariance matrix. $\mathbf{\Sigma}_{\mathrm{hclust}}$ can then be used in the quadratic programming problem defined in Equation (5.4).

5.2.3 A null model and a greedy algorithm

The result of the hierarchical clustering can be used in an alternative way for the Markowitz model. The method we use consists of creating a correlation matrix $\mathbf{C}^0$ of size $n \times n$, where C^0_{ij} is the average correlation between stocks i and j according to some null model. There are several possible null models. One trivial possibility is to assume that all stocks are uncorrelated, which would result in $\mathbf{C}^0$ being the identity matrix of size $n \times n$. Here, we use a configuration matrix that generates $\mathbf{C}^0$, which can be created using the iterative procedure described in [86]. The assumption is that for each i stock in $\mathbf{C}^0$, the so-called correlation "strength" $C^0_i = \sum_j C_{ij}$ is constant, but each C_{ij} $(j = 1, \ldots, n)$ correlation is randomly generated.

The two matrices ($\mathbf{C}^0$ and $\mathbf{C}$) are used to form a difference matrix and then rescaled:

$$
\begin{aligned}
\mathbf{C}' &= \left| \mathbf{C} - \mathbf{C}^0 \right|, \\
\mathbf{D}_C &= -\mathbf{C}' + |\min \mathbf{C}'| + |\max \mathbf{C}'| \,.
\end{aligned}
\tag{5.6}
$$

The resulting matrix $\mathbf{D}_C$ can be interpreted as a weighted graph that can be linked to the correlation matrix. We perform rescaling because hierarchical clustering merges clusters with the minimum distance, which is precisely the largest difference between the correlation matrix and the null model.

The hierarchical clustering procedure on the matrix D_c is equivalent to optimization of the following, *modularity-like function* using a greedy algorithm [87]:

$$M = \sum_{i,j} \left| C_{ij} - C_{ij}^0 \right| \cdot \delta_{ij}. \tag{5.7}$$

δ_{ij} is a binary variable that indicates whether i and j are in the same cluster, i.e., $\delta_{ij} = 1$ if i and j are in the same cluster, otherwise $\delta_{ij} = 0$.

With this interpretation, we can not only define a clustering, but we can also describe the "goodness" of the clustering using a single number. We consider the clustering for which M is maximal, and then construct the covariance matrix for the Markowitz model by choosing only one stock from a cluster at random (which is how we chose to solve this task) or along some other consideration.

5.3 Hierarchical clustering for direct portfolio selection

If we want to overcome some of the limitations of the Markowitz model, we need to use different diversification techniques. Multiple recently published methods are based on hierarchical clustering, see [75, 83, 84], just like the previously described filtering techniques. Their main goals are to avoid inverting the covariance matrix and to directly exploit the hierarchical structure of the system under study when building the portfolio. However, this comes at a cost. The disadvantage of these methods is that they are less parameterizable than the Markowitz model: we cannot specify a minimum expected return (R) for these methods.

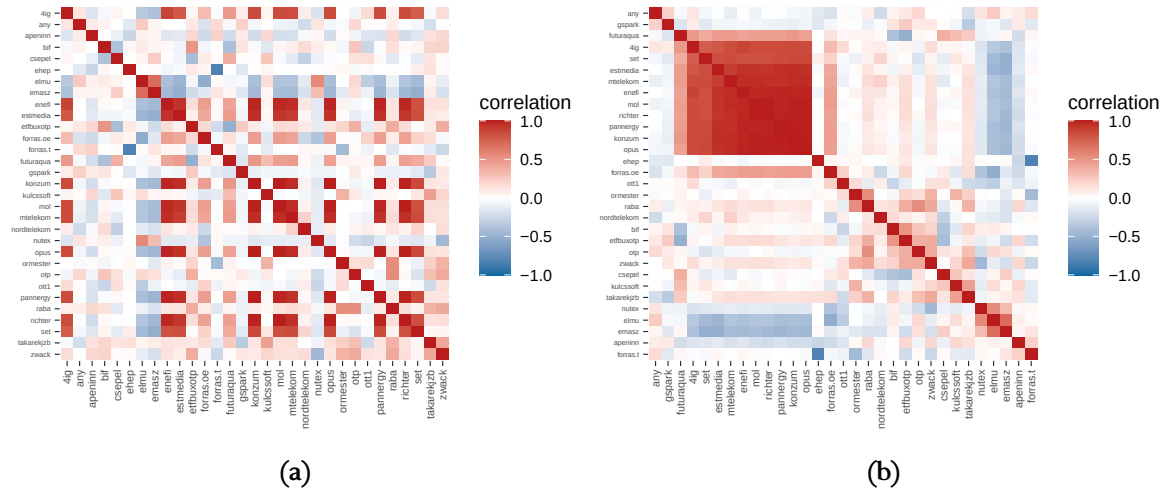
It is also important to point out that the direct application of graph clustering methods, regardless of the procedure, may lead to biased results for covariance or correlation matrices, since highly correlated pairs of vertices may not necessarily end up being clustered. This problem can be addressed in several ways [88]. We first present a possible approach following our previous work [89].

5.3.1 Hierarchical Risk Parity

The *Hierarchical Risk Parity* (HRP) algorithm [84], introduced in 2016, computes the optimal portfolio weights in three steps. First, it performs a hierarchical clustering, then a quasi-diagonalization on the covariance matrix. Finally, it recursively cuts the clustering dendrogram, assigning each stock a weight based on its contribution to the total risk and that of its parent clusters.

In the first step, we apply the procedure described in Section 5.2.2, but we do not perform the filtering of the covariance matrix, i.e., we only produce the merge tree from the distance matrix obtained from the correlation matrix.

In the second step, we rearrange the rows and columns of the $\mathbf{\Sigma}$ covariance matrix so that the largest values are along the diagonal. This can be achieved by using the order of the leaves of the



(a) (b)

Figure 5.4: *Correlation matrix before (a) and after (b) reordering.*

dendrogram to reorder the covariance matrix based on the result of the clustering applied in the first step. The exact algorithm is described in [84]. Figure 5.4 illustrates the result of this operation on a correlation matrix.

In the third step, the merge tree is iteratively split in two, starting at the root. At each split, the weight of the parent is distributed between the two new clusters according to their contribution to the variance of the parent. We repeat this until each stock falls into its own cluster. At the end of the procedure, we have the weight of each stock in the portfolio.

5.3.2 Hierarchical Equal Risk Contribution Portfolio

Hierarchical Equal Risk Contribution Portfolio (HERC) [83] is a modification of the HRP algorithm presented previously.

This algorithm also performs hierarchical clustering as its first step, but then determines the optimal number of clusters to cut the binary merge tree into. This number is based on a statistical index – in [83], the so-called *Gap index* is calculated for each possible k cluster count, and the algorithm chooses the value k where the index is at its maximum.

Once the optimal number of clusters has been determined, we first allocate the capital between clusters (using the variance distribution technique familiar from HRP), and then within the clusters. To do this, we can apply the recursive cutting used in HRP, by cutting and dividing the weights by some risk metric (e.g., variance) only until we reach k clusters. Once this is done, we divide the weight of each cluster equally among its stocks.

5.4 Comparison of methods

5.4.1 Data and methodology

Our experiments were conducted on real data using daily closing prices of 31 stocks from the Budapest Stock Exchange (BSE) from November 29, 2011 to October 18, 2019 and 454 stocks that were part of the Standard and Poor's (S&P) 500 index between September 18, 2008 and September 25, 2018. Both data sets consisted of daily closing prices, with a total of 1962 records for the BSE and 2523 records for the S&P 500 dataset.

To model the investment process, we set up a simulation environment that performs both the construction and evaluation of different portfolios at a given starting time of t_0. We then increase the value of t_0 by 10 and repeat the experiment. Following a sliding window approach, we repeat the evaluations until we reach the end of the dataset. At each start time t_0, we evaluate the investment model with an observation length of $T = 500$, observing prices for 500 days after t_0, and then make the purchase based on the closing prices of the past 500 days. After the purchase, we wait another 500 days, sell the portfolio, and evaluate its performance.

5.4.2 Evaluation criteria

For a resulting portfolio $\mathbf{p}$, we calculated indicators in three main categories: portfolio returns, risks, and size. In the case of returns and risks, we can talk about *estimated* and *realized* values. Estimated values are calculated based on past observations at the time of constructing the portfolio, while realized values are calculated when the portfolio is sold (T days after the investment). The relationship of these values can help us reveal how well each method estimates risk and return.

Returns

For a portfolio $\mathbf{p}$, we calculate two types of returns. The *estimated* return shows the return that could have been obtained at the moment of investment with the given portfolio weights, had the investor made the investment T days ago. This can be defined as the weighted average of past returns:

$$r_{\text{pre}} = \sum_{i=1}^{n} p_i \cdot \overline{x}_i, \tag{5.8}$$

where $\overline{x}_i$ is the return obtained over T days before the investment.

At the time of sale (T days after the investment), the ex-post (or *realized*) return of the portfolio can be determined, which tells us how the selling price of the stock compared to its purchase price:

$$r_{\text{post}} = \sum_{i=1}^{n} p_i \cdot \frac{P_i(t_0 + T) - P_i(t_0)}{P_i(t_0)}. \tag{5.9}$$

This value will be 0 if the selling price is equal to the purchase price, positive if the investment is profitable and negative if the investment made a loss.

Risks

Another important characteristic of portfolios is their risk and the accuracy of the methods used to estimate the risk. The estimated and realized risk of a portfolio can be determined using the following equations:

$$\sigma_{\text{pre}} = \mathbf{p}\boldsymbol{\Sigma}_{\text{pre}}\mathbf{p}^T, \tag{5.10}$$

$$\sigma_{\text{post}} = \mathbf{p}\boldsymbol{\Sigma}_{\text{post}}\mathbf{p}^T, \tag{5.11}$$

where $\boldsymbol{\Sigma}_{\text{pre}}$ is the covariance matrix computed over $[t_0 - T, t_0]$ and used in the portfolio selection, and $\boldsymbol{\Sigma}_{\text{post}}$ is the covariance matrix computed over $[t_0, t_0 + T]$.

From these two metrics, we can determine the relationship between the realized and the estimated risks for a portfolio. The *risk ratio*

$$\sigma_r = \left| \frac{\sigma_{\text{post}}}{\sigma_{\text{pre}}} \right| \tag{5.12}$$

measures the ratio of the two metrics, which is better the closer it is to 1. If the risk ratio is greater than 1, the risk was underestimated, if smaller, the risk was overestimated.

We can define the *change in risk* similarly:

$$\sigma_g = \left| \frac{\sigma_{\text{post}} - \sigma_{\text{pre}}}{\sigma_{\text{pre}}} \right|, \tag{5.13}$$

which is better, the closer it is to 0.

Portfolio size

In an investment, one may not only have to pay the price of the shares, but also transaction costs, for example. In this case, it may help to have a portfolio with fewer stocks, but this may also increase the risk by reducing the degree of diversification. The size of a portfolio $\mathbf{p}$ can be defined as the reciprocal of the sum of squared portfolio weights, but this can lead to distorted results in situations where negative weights are allowed. To compensate for this, the weights are first divided by the absolute norm of the portfolio vector (the sum of the absolute values of its elements):

$$\mathcal{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{n} \left(\frac{p_i}{||\mathbf{p}||_1} \right)^2}. \tag{5.14}$$

The portfolio size will be 1 if our capital is concentrated in a single stock in the portfolio, and n if we have invested equally in all stocks. We introduce a normalization of the portfolio size to make

it comparable across datasets of different size, and not dependent on n:

$$\hat{\mathcal{N}}_{\text{eff}} = \frac{\mathcal{N}_{\text{eff}} - 1}{n - 1}. \tag{5.15}$$

This value will then range between 0 and 1, where 0 represents a portfolio of 1 stock, and 1 represents a uniform weighting of n stocks.

5.4.3 Results

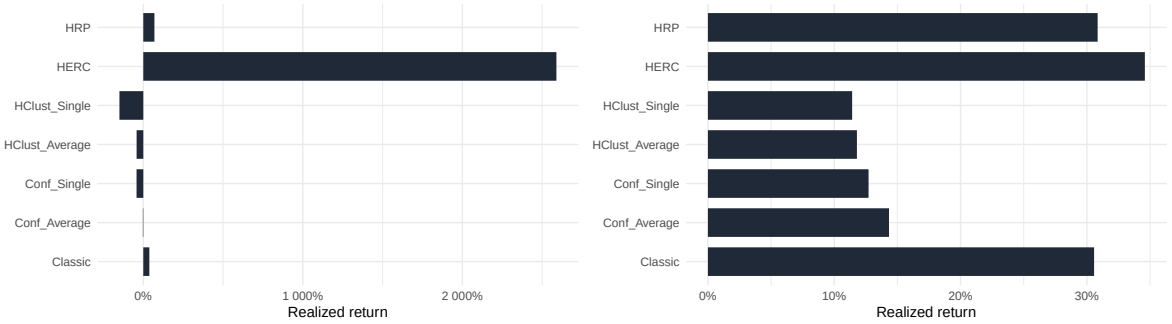
Below, we compare the performance of the methods tested, broken down by metrics. In the calculations, we consider the means and variances of the indicators (for all the portfolios assembled), with the expected return parameter of the Markowitz model (R) fixed at the average return of past observations. Tables 5.1 and 5.2 give an overview of the calculated metrics, highlighting methods that were stronger than the others in one area.

Returns

In terms of estimated returns, the hierarchical clustering-based portfolio selection methods performed worse on average (often with negative expected returns), but their realized returns were much better. We observed that the price of two stocks (OPUS and KONZUM) increased by several orders of magnitude between 2016 and 2018 – for example, the price of OPUS increased from 4.4 Ft to 708 Ft on the same day two years later. The HRP and HERC algorithms predicted this increase well and invested heavily in these stocks (with weights up to 30-40%), which resulted in a huge increase in the realized return ratio (up to 2000% for HERC and 365% for HRP).

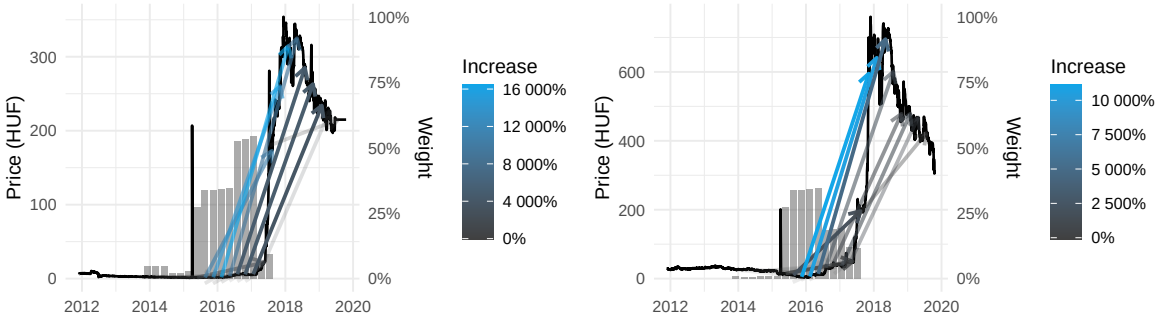
In comparison, the Markowitz model and its extensions performed worse: the original method produced an average realized return of 40.9%, the hierarchical clustering filtering methods were slightly better at 41%, and the configuration model-based method produced an average realized return of 1.1% in the case of average-linkage and -35% in the case of single-linkage.

The odd average realized return of 2000% can be explained by the fact that the prices of the two assets OPUS and KONZUM rose by unrealistically large amounts, and HERC invested heavily (in some cases with weights of over 50%) in these stocks. Figure 5.6 shows how the prices of these two stocks have changed over time, along with the $T = 500$ long investments and the weights assigned to each of the stocks by the HERC algorithm. This also explains why the risks were high for the HERC algorithm: these two stocks moved very close together, but they had combined weights of upwards 60-70%. While such a situation is highly unrealistic in the market (and could be the result of legislative changes, the disruption of the free market or unseen mergers), this example shows that the HERC method predicted the price increases very well in this particular case. We also note that the delay of $T = 500$ market days seemed to be a very fortunate length of time to hold these stocks for, as their prices increased tremendously during these periods.



(a) (b)

Figure 5.5: *Realized returns for all methods on the (a) BSE and (b) S&P 500 datasets.*



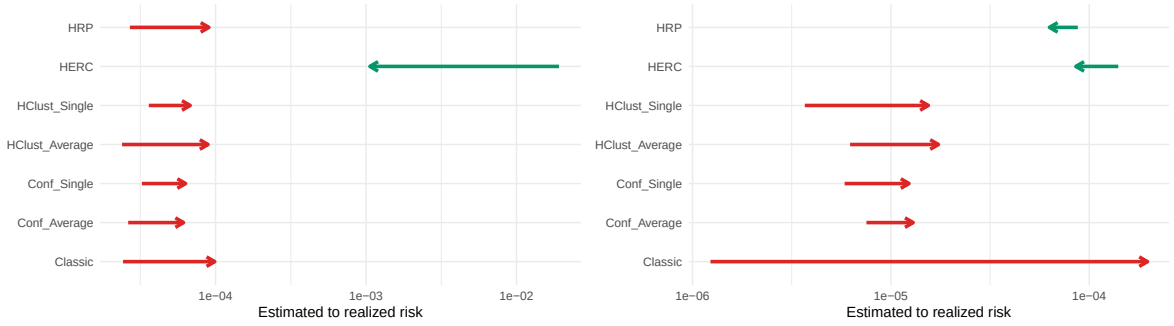
(a) *KONZUM* (b) *OPUS*

Figure 5.6: *Price changes of the assets KONZUM and OPUS. The black line marks the price of the stocks, while bars represent the portfolio weights assigned to these stocks at the time of buying. Arrows show the price increase from the time of buying to selling.*

Risks

On both datasets, the original Markowitz model and its extensions had low estimated risks, while the HERC model's predictions were two to three orders of magnitude higher – as seen on Figure 5.7a. The realized risks showed a different pattern: since most methods provided much better risk estimation, they usually produced higher estimated or lower realized risks. The exception was the HERC model, which produced much higher estimated and realized risks, but it also overestimated the risk instead of underestimating it. The original Markowitz model produced the lowest estimate on both datasets, but had a high magnitude of increase in realized risk, further demonstrating that the filtering methods improved overall risk estimation.

To get a clearer picture of the risk assessment of the methods, we can also look at the risk ratio (σ_r), plotted as the arrow length in Figure 5.7. For the BSE dataset, the original Markowitz model



(a) (b)

Figure 5.7: *Changes between estimated and realized risks on the (a) BSE and (b) S&P 500 datasets. The arrows point towards the realized risk and their lengths correspond to the magnitude of over- or underestimation – if red, the method underestimated the risk, if green, it overestimated it.*

underestimated the risk the most (3.98), while the average-linkage hierarchical clustering (3.47) and the HRP model (3.34) slightly improved on it. The configuration models were significantly better in this area (2.27 for average-linkage and 1.87 for single-linkage). The HRP and HERC algorithms behaved very differently, with the HRP underestimating the risk (along with the Markowitz model), and the HERC significantly overestimating it ($\sigma_r = 0.19$).

On the S&P 500 dataset (Figure 5.7b), the filtering methods significantly improved risk estimation, bringing the risk ratios down to between 1.79 (configuration model with average-linkage clustering) and 4.51 (hierarchical clustering with single-linkage clustering), while the original Markowitz model had a risk ratio of 167.35. Here, both the HRP and HERC models overestimated the risk, although much less than the HERC did on the BSE dataset: the algorithms had a risk ratio of 0.88 and 0.81, respectively, or overestimation factors of 1.14 and 1.23.

Portfolio size

On the BSE dataset (Figure 5.8a), the different methods produced portfolios of similar size, with the normalized index of several methods being close to 0.4. The exception was the HERC algorithm, which produced much more concentrated portfolios ($\hat{\mathcal{N}}_{\text{eff}} = 0.14$), with the smallest portfolio size of all methods. This is comparable to the finding that this method was able to achieve higher returns than the other algorithms by concentrating capital even more in stocks that were experiencing high price appreciation.

Portfolio sizes showed a completely different trend on the S&P 500 dataset (Figure 5.8b). The HERC algorithm, which previously produced concentrated portfolios, mostly allocated capital almost evenly across stocks when executed on the larger dataset. Likewise, the HRP algorithm also assembled more evenly allocated portfolios, but still much less so than HERC. The hierarchical

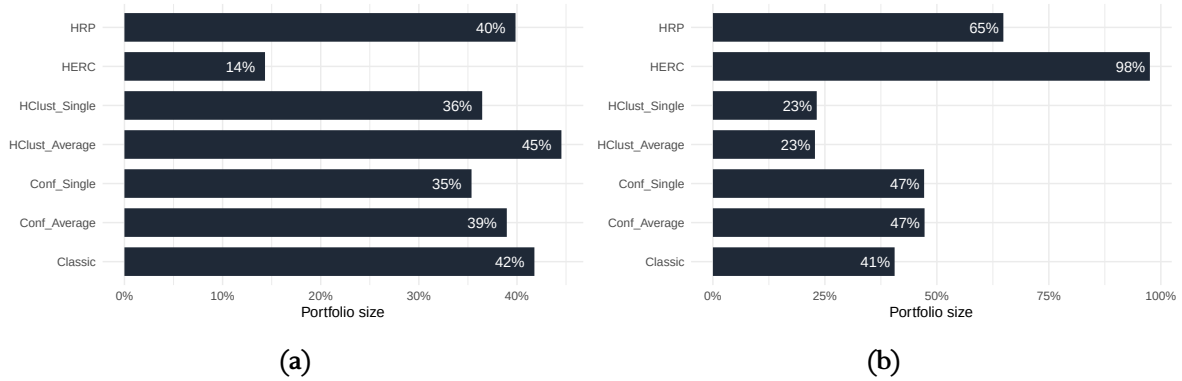


Figure 5.8: *Normalized portfolio sizes on the (a) BSE and (b) S&P 500 datasets.*

clustering filtering methods made the resulting portfolios more concentrated, while the configuration model-based approaches made them more balanced. The base Markowitz model produced a consistent 41% normalized portfolio size.

5.5 Conclusions

In this chapter, we compared portfolio selection methods based on hierarchical clustering, including methods that complement other portfolio selection models (through filtering) and new portfolio selection methods based entirely on hierarchical clustering. Our experiments on two real-world datasets show that the new portfolio selection methods were able to produce higher returns than the original Markowitz model, while also providing good, and sometimes much better, risk estimation. The HERC method also predicted two exceptional price spikes on the smaller Hungarian dataset, investing heavily in stocks whose prices rose steeply, thereby also increasing risk in the process. This model also tended to overestimate the risk rather than underestimate it, a behavior an investor might find preferable.

Aside from returns and risks, we also examined portfolio sizes and found that the filtering-based methods didn't show any particular tendency to increase or decrease portfolio sizes. The hierarchical clustering based filtering methods assembled more concentrated portfolios on the larger dataset, while the HERC algorithm produced highly concentrated portfolios on the small BSE dataset and very evenly distributed portfolios on the larger S&P 500 dataset.

In conclusion, based on our experiments, the filtering methods seem to be the preferred way to reduce risk estimation errors of the Markowitz model, while the hierarchical clustering based portfolio selection methods HRP and HERC have the ability to produce higher returns in exchange for higher risks. The reduction in risk estimation errors was much stronger when the days of past performance taken into account (T) was closer to the number of stocks.

Table 5.1: *Summary of the results of the analysis on the Budapest Stock Exchange data: averages are shown in cells, standard deviations in brackets. In the case of the Markowitz model, the target expected returns were the average returns.*

	Classic	Conf_Average	Conf_Single	HClust_Average	HClust_Single	HERC	HRP
r_{pre}	0.0004 (0.00)	0.0005 (0.00)	0.0006 (0.00)	0.0005 (0.00)	0.0007 (0.00)	-0.0016 (0.00)	0.0003 (0.00)
r_{post}	0.3802 (0.56)	-0.0277 (0.44)	-0.4226 (0.78)	-0.4191 (0.96)	-1.4944 (1.79)	**25.8978** (31.04)	0.6969 (0.52)
σ_{post}	0.0001 (0.00)	0.0001 (0.00)	0.0001 (0.00)	0.0001 (0.00)	0.0001 (0.00)	0.0011 (0.00)	0.0001 (0.00)
$\hat{\mathcal{N}}_{\text{eff}}$	0.4178 (0.04)	0.3896 (0.04)	0.3537 (0.04)	0.4453 (0.03)	0.3645 (0.04)	0.1432 (0.10)	0.3985 (0.07)
σ_{r}	3.9779 (3.94)	2.2662 (1.95)	1.8689 (1.34)	3.4700 (3.27)	**1.7393** (0.83)	**0.1937** (0.31)	3.3448 (3.30)
σ_{g}	3.0611 (3.88)	1.4853 (1.79)	1.0708 (1.18)	2.5071 (3.24)	**0.7766** (0.79)	0.8172 (0.28)	2.5196 (3.17)

Table 5.2: *Summary of the results of the analysis on the S&P 500 dataset.*

	Classic	Conf_Average	Conf_Single	HClust_Average	HClust_Single	HERC	HRP
r_{pre}	0.0006 (0.00)	0.0007 (0.00)	0.0007 (0.00)	0.0007 (0.00)	0.0007 (0.00)	0.0006 (0.00)	**0.0005** (0.00)
r_{post}	0.3057 (0.32)	0.1434 (0.16)	0.1272 (0.20)	0.1180 (0.18)	0.1142 (0.22)	**0.3459** (0.17)	0.3085 **(0.12)**
σ_{post}	0.0002 (0.00)	**0.0000** (0.00)	**0.0000** (0.00)	**0.0000** (0.00)	**0.0000** (0.00)	0.0001 (0.00)	0.0001 (0.00)
$\hat{\mathcal{N}}_{\text{eff}}$	0.4056 (0.11)	0.4725 (0.05)	0.4715 (0.06)	0.2278 (0.23)	0.2318 (0.24)	0.9755 (0.03)	0.6484 (0.10)
σ_{r}	167.3468 (52.93)	**1.7937** (0.48)	2.1923 (0.62)	2.9028 (1.15)	4.5093 (2.38)	0.8090 (0.49)	**0.8821** (0.50)
σ_{g}	166.3468 (52.93)	0.8024 (0.47)	1.1985 (0.61)	1.9028 (1.15)	3.5093 (2.38)	0.4783 (0.21)	**0.4712** (0.19)

Bibliography

[1] Wasserman, Stanley and Faust, Katherine. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Nov. 1994. ISBN: 9780511815478. DOI: 10.1017/cbo9780511815478.

[2] Jackson, Matthew O. *Social and Economic Networks*. Princeton University Press, Dec. 2008. ISBN: 9781400833993. DOI: 10.1515/9781400833993.

[3] Price, Derek J. de Solla. "Networks of Scientific Papers: The pattern of bibliographic references indicates the nature of the scientific research front." In: *Science* 149.3683 (July 1965), pp. 510–515. ISSN: 1095-9203. DOI: 10.1126/science.149.3683.510.

[4] Jordano, Pedro. "Patterns of Mutualistic Interactions in Pollination and Seed Dispersal: Connectance, Dependence Asymmetries, and Coevolution". In: *The American Naturalist* 129.5 (May 1987), pp. 657–677. ISSN: 1537-5323. DOI: 10.1086/284665.

[5] Stam, Cornelis J. "Modern network science of neurological disorders". In: *Nature Reviews Neuroscience* 15.10 (Sept. 2014), pp. 683–695. ISSN: 1471-0048. DOI: 10.1038/nrn3801.

[6] Yu, Yi and Xiao, Gaoxi. "Infection spreading, detection and control in community networks". In: *Journal of Complex Networks* 5.4 (Feb. 2017), pp. 625–640. ISSN: 2051-1329. DOI: 10.1093/comnet/cnw031.

[7] Sarvari, Hamed et al. "Constructing and Analyzing Criminal Networks". In: *2014 IEEE Security and Privacy Workshops*. IEEE, May 2014. DOI: 10.1109/spw.2014.22.

[8] Brandes, Ulrik et al. "What is network science?" In: *Network Science* 1.1 (Apr. 2013), pp. 1–15. ISSN: 2050-1250. DOI: 10.1017/nws.2013.2.

[9] Zachary, Wayne W. "An information flow model for conflict and fission in small groups". In: *Journal of Anthropological Research* 33.4 (1977), pp. 452–473. DOI: 10.1086/jar.33.4.3629752.

[10] Radner, Roy. "Hierarchy: The Economics of Managing". In: *Journal of Economic Literature* 30.3 (1992), pp. 1382–1415. ISSN: 00220515.

[11] Redhead, Daniel and Power, Eleanor A. "Social hierarchies and social networks in humans". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 377.1845 (Jan. 2022). ISSN: 1471-2970. DOI: 10.1098/rstb.2020.0440.

[12] Huxham, Mark, Raffaelli, Dave, and Pike, Alan. "Parasites and Food Web Patterns". In: *The Journal of Animal Ecology* 64.2 (Mar. 1995), pp. 168–176. ISSN: 0021-8790. DOI: 10.2307/5752.

[13] Murtagh, Fionn and Contreras, Pedro. "Algorithms for hierarchical clustering: an overview, II". In: *WIREs Data Mining and Knowledge Discovery* 7.6 (Sept. 2017). ISSN: 1942-4795. DOI: 10.1002/widm.1219.

[14] Bastolla, Ugo et al. "The architecture of mutualistic networks minimizes competition and increases biodiversity". In: *Nature* 458.7241 (2009), pp. 1018–1020. DOI: 10.1038/nature07950.

[15] Uzzi, Brian. "The sources and consequences of embeddedness for the economic performance of organizations: The network effect". In: *American Sociological Review* 61.4 (1996), pp. 674–698. DOI: 10.2307/2096399.

[16] Mariani, Manuel Sebastian et al. "Nestedness in complex networks: observation, emergence, and implications". In: *Physics Reports* 813 (2019), pp. 1–90. DOI: 10.1016/j.physrep.2019.04.001.

[17] Wright, David H. et al. "A comparative analysis of nested subset patterns of species composition". In: *Oecologia* 113 (1997), pp. 1–20. DOI: 10.1007/s004420050348.

[18] Patterson, Bruce D. and Atmar, Wirt. "Nested subsets and the structure of insular mammalian faunas and archipelagos". In: *Biological Journal of the Linnean Society* 28.1–2 (May 1986), pp. 65–82. ISSN: 0024-4066. DOI: 10.1111/j.1095-8312.1986.tb01749.x.

[19] Tacchella, Andrea et al. "A New Metrics for Countries' Fitness and Products' Complexity". In: *Scientific Reports* 2.1 (Oct. 2012). ISSN: 2045-2322. DOI: 10.1038/srep00723.

[20] Bascompte, Jordi et al. "The nested assembly of plant-animal mutualistic networks". In: *Proceedings of the National Academy of Sciences* 100.16 (July 2003), pp. 9383–9387. ISSN: 1091-6490. DOI: 10.1073/pnas.1633576100.

[21] König, Michael D., Tessone, Claudio J., and Zenou, Yves. "Nestedness in networks: A theoretical model and some applications: Nestedness in networks". In: *Theoretical Economics* 9.3 (Sept. 2014), pp. 695–752. ISSN: 1933-6837. DOI: 10.3982/te1348.

[22] Almeida-Neto, Mário and Ulrich, Werner. "A straightforward computational approach for measuring nestedness using quantitative matrices". In: *Environmental Modelling & Software* 26.2 (2011), pp. 173–178. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2010.08.003.

[23] Rodríguez-Gironés, Miguel Ángel and Santamaría, Luis. "How Foraging Behaviour and Resource Partitioning Can Drive the Evolution of Flowers and the Structure of Pollination Networks". In: *The Open Ecology Journal* 3.1 (Jan. 2010), pp. 1–11. ISSN: 1874-2130. DOI: 10.2174/1874213001003040001.

[24] Brualdi, Richard A. and Sanderson, James G. "Nested species subsets, gaps, and discrepancy". In: *Oecologia* 119.2 (May 1999), pp. 256–264. ISSN: 1432-1939. DOI: 10.1007/s004420050784.

[25] London, András, Martin, Ryan R, and Pluhár, András. "Graph clustering via generalized colorings". In: *Theoretical Computer Science* 918 (2022), pp. 94–104. DOI: 10.1016/j.tcs.2022.03.023.

[26] Junttila, Esa and Kaski, Petteri. "Segmented nestedness in binary data". In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM. 2011, pp. 235–246. DOI: 10.1137/1.9781611972818.21.

[27] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: https://www.R-project.org/.

[28] Newman, M. E. J. and Girvan, M. "Finding and evaluating community structure in networks". In: *Phys. Rev. E* 69.2 (2004), p. 026113. DOI: 10.1103/PhysRevE.69.026113.

[29] Atmar, W. and Patterson, B.D. "The measure of order and disorder in the distribution of species in fragmented habitat". In: *Oecologia* 96.3 (1993), pp. 373–382. DOI: 10.1007/BF00317508.

[30] Rodríguez-Gironés, Miguel Ángel and Santamaría, Luis. "A new algorithm to calculate the nestedness temperature of presence-absence matrices". In: *Journal of Biogeography* 33.5 (2006), pp. 924–935. DOI: 10.1111/j.1365-2699.2006.01444.x.

[31] Almeida-Neto, Mário et al. "A consistent metric for nestedness analysis in ecological systems: reconciling concept and measurement". In: *Oikos* 117.8 (2008), pp. 1227–1239. DOI: 10.1111/j.0030-1299.2008.16644.x.

[32] Bascompte Lab. *Web of Life: ecological networks database*. https://www.web-of-life.es. Jan. 2023.

[33] Hadfield, Jarrod D. et al. "A Tale of Two Phylogenies: Comparative Analyses of Ecological Interactions". In: *The American Naturalist* 183.2 (Feb. 2014), pp. 174–187. ISSN: 1537-5323. DOI: 10.1086/674445.

[34] Erdős, Pál and Rényi, Alfréd. "On the evolution of random graphs". In: *The Structure and Dynamics of Networks*. Princeton: Princeton University Press, 2006, pp. 38–82. ISBN: 9781400841356. DOI: 10.1515/9781400841356.38.

[35] Schaeffer, Satu Elisa. "Graph clustering". In: *Computer Science Review* 1.1 (2007), pp. 27–64. DOI: 10.1016/j.cosrev.2007.05.001.

[36] Xu, Dongkuan and Tian, Yingjie. "A comprehensive survey of clustering algorithms". In: *Annals of Data Science* 2.2 (2015), pp. 165–193. DOI: 10.1007/s40745-015-0040-1.

[37] McGlohon, Mary, Akoglu, Leman, and Faloutsos, Christos. "Statistical Properties of Social Networks". In: *Social Network Data Analytics*. Springer US, 2011, pp. 17–42. DOI: 10.1007/978-1-4419-8462-3_2.

[38] Bóta, András, Csizmadia, László, and Pluhár, András. "Community detection and its use in Real Graphs". In: *Proceedings of the Conference on Applied Theoretical Computer Science (MATCOS)*. 2010, pp. 95–99.

[39] Adcock, Aaron B, Sullivan, Blair D, and Mahoney, Michael W. "Tree-like structure in large social and information networks". In: *2013 IEEE 13th international conference on data mining*. IEEE. 2013, pp. 1–10. DOI: 10.1109/icdm.2013.77.

[40] Bascompte, Jordi. "Structure and dynamics of ecological networks". In: *Science* 329.5993 (2010), pp. 765–766. DOI: 10.1126/science.1194255.

[41] Gera, Imre, London, András, and Pluhár, András. "Greedy algorithm for edge-based nested community detection". In: *2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS)*. 2022, pp. 86–91. DOI: 10.1109/CITDS54976.2022.9914051.

[42] Solé-Ribalta, Albert et al. "Revealing in-block nestedness: Detection and benchmarking". In: *Physical Review E* 97.6 (2018), p. 062302. DOI: 10.1103/PhysRevE.97.062302.

[43] Kohler, Glauco Ubiratan. *Redes de interação Planta Beija-Flor em um Gradiente Altitudinal de Floresta Atlântica no Sul do Brasil.* 2011.

[44] Bartomeus, Ignasi, Vilà, Montserrat, and Santamaría, Luís. "Contrasting effects of invasive plants in plant–pollinator networks". In: *Oecologia* 155.4 (2008), pp. 761–770. DOI: 10.1007/s00442-007-0946-1.

[45] Blondel, Vincent D et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2008/10/p10008.

[46] Breiger, Ronald L and Pattison, Philippa E. "Cumulated social roles: The duality of persons and their algebras". In: *Social Networks* 8.3 (1986), pp. 215–256. DOI: 10.1016/0378-8733(86)90006-7.

[47] Ahn, Yong-Yeol, Bagrow, James P, and Lehmann, Sune. "Link communities reveal multiscale complexity in networks". In: *Nature* 466.7307 (2010), pp. 761–764. DOI: 10.1038/nature09182.

[48] McDaid, Aaron and Hurley, Neil. "Detecting highly overlapping communities with model-based overlapping seed expansion". In: *2010 International Conference on Advances in Social Networks Analysis and Mining*. IEEE. 2010, pp. 112–119. DOI: 10.1109/ASONAM.2010.77.

[49] Adamcsek, Balázs et al. "CFinder: locating cliques and overlapping modules in biological networks". In: *Bioinformatics* 22.8 (2006), pp. 1021–1023. DOI: 10.1093/bioinformatics/btl039.

[50] Lutov, Artem, Khayati, Mourad, and Cudré-Mauroux, Philippe. "Accuracy evaluation of overlapping and multi-resolution clustering algorithms on large datasets". In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2019, pp. 1–8. DOI: 10.1109/bigcomp.2019.8679398.

[51] Hubert, Lawrence and Arabie, Phipps. "Comparing partitions". In: *Journal of Classification* 2.1 (Dec. 1985), pp. 193–218. ISSN: 1432-1343. DOI: 10.1007/bf01908075.

[52] Fortunato, Santo. "Community detection in graphs". In: *Physics Reports* 486.3–5 (Feb. 2010), pp. 75–174. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2009.11.002.

[53] Rombach, M. Puck et al. "Core-Periphery Structure in Networks". In: *SIAM Journal on Applied Mathematics* 74.1 (Jan. 2014), pp. 167–190. ISSN: 1095-712X. DOI: 10.1137/120881683.

[54] Stone, Lewi, Simberloff, Daniel, and Artzy-Randrup, Yael. "Network motifs and their origins". In: *PLOS Computational Biology* 15.4 (Apr. 2019). Ed. by Ruth Nussinov, e1006749. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1006749.

[55] Csermely, Péter et al. "Structure and dynamics of core/periphery networks". In: *Journal of Complex Networks* 1.2 (Oct. 2013), pp. 93–123. ISSN: 2051-1329. DOI: 10.1093/comnet/cnt016.

[56] Gera, Imre and London, András. "Detecting and generating overlapping nested communities". In: *Applied Network Science* 8.1 (Aug. 2023). ISSN: 2364-8228. DOI: 10.1007/s41109-023-00575-2.

[57] Sales-Pardo, Marta et al. "Extracting the hierarchical organization of complex systems". In: *Proceedings of the National Academy of Sciences* 104.39 (Sept. 2007), pp. 15224–15229. ISSN: 1091-6490. DOI: 10.1073/pnas.0703740104.

[58] Lancichinetti, Andrea, Fortunato, Santo, and Kertész, János. "Detecting the overlapping and hierarchical community structure in complex networks". In: *New Journal of Physics* 11.3 (Mar. 2009), p. 033015. ISSN: 1367-2630. DOI: 10.1088/1367-2630/11/3/033015.

[59] Isogai, Takashi. "Clustering of Japanese stock returns by recursive modularity optimization for efficient portfolio diversification". In: *Journal of Complex Networks* 2.4 (July 2014), pp. 557–584. ISSN: 2051-1310. DOI: 10.1093/comnet/cnu023.

[60] Mannila, Heikki and Terzi, Evimaria. "Nestedness and segmented nestedness". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. Vol. 96. KDD07. ACM, Aug. 2007, pp. 480–489. DOI: 10.1145/1281192.1281245.

[61] Roux, Maurice. "A Comparative Study of Divisive and Agglomerative Hierarchical Clustering Algorithms". In: *Journal of Classification* 35.2 (July 2018), pp. 345–366. ISSN: 1432-1343. DOI: 10.1007/s00357-018-9259-9.

[62] Kahvecioğlu, Gökçe and Morton, David P. "Optimal hierarchical clustering on a graph". In: *Networks* 79.2 (2022), pp. 143–163. DOI: 10.1002/net.22043.

[63] Mantegna, Rosario N. "Hierarchical structure in financial markets". In: *The European Physical Journal B* 11.1 (1999), pp. 193–197. DOI: 10.1007/s100510050929.

[64] Girvan, M. and Newman, M. E. J. "Community structure in social and biological networks". In: *Proceedings of the National Academy of Sciences* 99.12 (June 2002), pp. 7821–7826. ISSN: 1091-6490. DOI: 10.1073/pnas.122653799.

[65] Tumminello, Michele et al. "Statistically Validated Networks in Bipartite Complex Systems". In: *PLOS ONE* 6.3 (Mar. 2011). Ed. by Eshel Ben-Jacob, e17994. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0017994.

[66] Bongiorno, Christian et al. "Core of communities in bipartite networks". In: *Physical Review E* 96.2 (Aug. 2017), p. 022321. ISSN: 2470-0053. DOI: 10.1103/physreve.96.022321.

[67] Micciché, Salvatore and Mantegna, Rosario N. "A primer on statistically validated networks". In: *Computational Social Science and Complex Systems*. Vol. 203. Proceedings of the International School of Physics "Enrico Fermi". IOS Press, 2019, pp. 91–103. DOI: 10.3254/190007.

[68] Sedgwick, Philip. "Multiple hypothesis testing and Bonferroni's correction". In: *BMJ* 349 (Oct. 2014), g6284. ISSN: 1756-1833. DOI: 10.1136/bmj.g6284.

[69] Leskovec, Jure and Krevl, Andrej. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[70] Newman, Mark. *Network data*. https://websites.umich.edu/~mejn/netdata. Jan. 2024.

[71] Rand, William M. "Objective Criteria for the Evaluation of Clustering Methods". In: *Journal of the American Statistical Association* 66.336 (Dec. 1971), pp. 846–850. ISSN: 1537-274X. DOI: 10.1080/01621459.1971.10482356.

[72] Tola, Vincenzo et al. "Cluster analysis for portfolio optimization". In: *Journal of Economic Dynamics and Control* 32.1 (2008), pp. 235–258. DOI: 10.1016/j.jedc.2007.01.034.

[73] Markowitz, Harry. "Portfolio selection". In: *The Journal of Finance* 7.1 (1952), pp. 77–91.

[74] Michaud, Richard O. "The Markowitz Optimization Enigma: Is 'Optimized' Optimal?" In: *Financial Analysts Journal* 45.1 (Jan. 1989), pp. 31–42. ISSN: 1938-3312. DOI: 10.2469/faj.v45.n1.31.

[75] Raffinot, Thomas. "Hierarchical clustering-based asset allocation". In: *The Journal of Portfolio Management* 44.2 (2017), pp. 89–99. DOI: 10.3905/jpm.2018.44.2.089.

[76] Bongiorno, Christian and Challet, Damien. "Non-parametric sign prediction of high-dimensional correlation matrix coefficients". In: *Europhysics Letters* 133.4 (Feb. 2021), p. 48001. ISSN: 1286-4854. DOI: 10.1209/0295-5075/133/48001.

[77] Laloux, Laurent et al. "Noise dressing of financial correlation matrices". In: *Physical Review Letters* 83.7 (1999), p. 1467. DOI: 10.1103/PhysRevLett.83.1467.

[78] Bun, Joël, Bouchaud, Jean-Philippe, and Potters, Marc. "Cleaning large correlation matrices: tools from random matrix theory". In: *Physics Reports* 666 (2017), pp. 1–109. DOI: 10.1016/j.physrep.2016.10.005.

[79] London, András, Gera, Imre, and Bánhelyi, Balázs. "Markowitz Portfolio Selection Using Various Estimators of Expected Returns and Filtering Techniques for Correlation Matrices". In: *Acta Polytechnica Hungarica* 15.1 (Jan. 2018), pp. 217–229. DOI: 10.12700/aph.15.1.2018.1.13.

[80] Simon, Herbert A. "The architecture of complexity". In: *Facets of Systems Science*. Springer, 1991, pp. 457–476. DOI: 10.1007/978-1-4899-0718-9_31.

[81] Tumminello, Michele, Lillo, Fabrizio, and Mantegna, Rosario N. "Hierarchically nested factor model from multivariate data". In: *Europhysics Letters* 78.3 (2007), p. 30006. DOI: 10.1209/0295-5075/78/30006.

[82] Embrechts, Paul, McNeil, Alexander, and Straumann, Daniel. "Correlation and dependence in risk management: properties and pitfalls". In: *Risk management: value at risk and beyond* 1 (2002), pp. 176–223. DOI: 10.1017/CBO9780511615337.008.

[83] Raffinot, Thomas. "The Hierarchical Equal Risk Contribution Portfolio". In: *SSRN Electronic Journal* (Aug. 2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.3237540.

[84] De Prado, Marcos Lopez. "Building diversified portfolios that outperform out of sample". In: *The Journal of Portfolio Management* 42.4 (2016), pp. 59–69. DOI: 10.3905/jpm.2016.42.4.059.

[85] Nielsen, Frank. "Hierarchical Clustering". In: Feb. 2016, pp. 195–211. ISBN: 978-3-319-21902-8. DOI: 10.1007/978-3-319-21903-5_8.

[86] Masuda, Naoki, Kojaku, Sadamori, and Sano, Yukie. "Configuration model for correlation matrices preserving the node strength". In: *Physical Review E* 98.1 (2018), p. 012312. DOI: 10.1103/PhysRevE.98.012312.

[87] Newman, Mark EJ. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23 (2006), pp. 8577–8582. DOI: 10.1073/pnas.0601602103.

[88] MacMahon, Mel and Garlaschelli, Diego. "Community detection for correlation matrices". In: *Physical Review E* 5 (2013), p. 021006. DOI: 10.1103/PhysRevX.5.021006.

[89] Gera, Imre and London, András. "Gráf alapú dimenzióredukciós heurisztikák részvénypiaci korrelációs mátrixokra". In: *Alkalmazott Matematikai Lapok* 37.2 (July 2020), pp. 211–224. DOI: 10.37070/AML.2020.37.2.06.

[90] Gera, Imre and London, András. "Hierarchikus klaszterezés és a portfólió-kiválasztás probléma". In: *SZIGMA Matematikai-közgazdasági folyóirat* 53.1 (2022), pp. 73–88.

[91] Gera, Imre and London, András. "Recovering Nested Structures in Networks: An Evaluation of Hierarchical Clustering Techniques". In: *Journal of Complex Networks* (2024). DOI: 10.1093/comnet/cnae039.

[92] Gera, Imre, Bánhelyi, Balázs, and London, András. "Testing the markowitz portfolio optimization method with filtered correlation matrices". In: *Proceedings of the 19th International Multiconference INFORMATION SOCIETY - IS 2016*. Institut Jožef Stefan, 2016, pp. 44–47.

[93] Gera, Imre and London, András. "Portfolio selection based on a configuration model and hierarchical clustering for asset graphs". In: *Proceedings of the MATCOS*. Vol. 19. 2019.

Summary

This PhD thesis concludes the research of more than four years in the fields of network science and data mining. My main goals were to widen the knowledge of community detection and clustering methods, focusing on the detection of communities of special graph structures. A key part of the thesis is to extend the set of available community detection algorithms for nestedness, a stiff but important graph structure that has mostly only been quantified and was only detected in bipartite graphs.

The dissertation consists of two major parts. In the first part, I introduced both a heuristic and an exact algorithm for finding overlapping nested structures in graphs, along with metrics that show how nested the entire graph is. Another key finding of this thesis is the adaptability of hierarchical clustering to various problems, which is presented in the second part. Clustering can provide an easier to understand picture of the community structure of a graph than overlapping community detection, and the hierarchical relationships of nodes can be used to gain a deeper understanding of the inner workings of a network. I applied these concepts to the problems of local nestedness detection and portfolio selection, two problems that can both be solved using clustering.

Nestedness as an overlapping community structure

In Part I, I showed that nestedness can be interpreted as a well-defined, overlapping local community structure. Nestedness in graphs means adhering to stiff structural constrains, requiring neighborhoods of vertices to be subsets of one another. Indices that quantify the level of nestedness have existed in the literature, as well as algorithms that detect disjoint fully nested clusters, though mostly for bipartite graphs only. The definition of nestedness can be extended to non-bipartite networks as well. I presented two approaches for finding overlapping fully nested subgraphs in general (non-bipartite) networks.

In Chapter 2, I introduced an adjustable heuristic algorithm for detecting nested structures in

networks using an edge-based approach. The algorithm uses the local nestedness metric $\mathrm{nest}(i, j)$ to iteratively assign edges to communities, while it allows adjusting the balance between performance and accuracy using a single parameter. I evaluated the algorithm on random and real-world bipartite graphs and showed that the algorithm detects a larger fraction of nested communities in its early steps, and that node and edge counts of the detected communities correlate strongly with the existing nestedness measure *discrepancy*.

Continuing on the work done in the previous chapter, in Chapter 3, I described an exact algorithm for finding all maximal nested subgraphs of a network. By constructing a nestedness graph and removing all redundancy from it, it allows not only to find the communities, but also to identify the position of each node (whether they are generalists or specialists) in the nested chains. I also introduced an algorithm to generate bipartite graphs with known nested structures, allowing to test similar algorithms introduced in the future. To be able to measure the amount of nestedness, I defined a metric that calculates the global nestedness index of the graph based on its nestedness graph. In my experiments, the algorithm showed that real-world bipartite networks often have a large amount of nested communities (in other words, low nestedness), partly due to the strictness of the definition, but some ecological networks displayed high nestedness values. Real-world non-bipartite networks, often used for benchmarking traditional community detection algorithms, seemed to show little nestedness, although in some cases, this property was also better to be avoided.

Both algorithms have the advantage of being able to work with general (non-bipartite) graphs, too. I have also made the reference implementation of each algorithm available as open source code.

Applications of hierarchical clustering

In Part II, I introduced specific applications of hierarchical clustering to problems that can be solved using clustering directly or indirectly. Hierarchical clustering not only creates multiple clustering levels with different resolutions, but also establishes a hierarchical relationship between the nodes that make up the clusters.

In Chapter 4, I adapted two hierarchical clustering approaches to detect disjoint fully nested subgraphs in networks. I defined multiple metrics to quantify both cluster- and graph-level nestedness in the networks studied. I evaluated the algorithms on multiple real-world network sets using different distance metrics, and found that using maximum (complete-linkage) and average (average-linkage) pairwise distances and the bottom-up clustering approach resulted in the largest fully nested clusters. This applied to both bipartite and non-bipartite networks.

In Chapter 5, I applied hierarchical clustering to the problem of portfolio selection in order to improve the performance of the traditional Markowitz model. I implemented a purely hierarchical clustering-based filtering, and a configuration model-based filtering algorithm for the covariance

matrix used by the Markowitz model, and compared them against the Hierarchical Risk Parity and Hierarchical Equal Risk Contribution algorithms. I compared the different algorithms on two real-world data sets of prices from the Budapest Stock Exchange and from the Standard and Poor's 500 index. My experiments showed that the implemented filtering techniques can significantly reduce the risk estimation errors and the portfolio size. The risk estimation showed the greatest improvements in situations where the sample size was close to the number of stocks. I also showed that the two hierarchical clustering-based portfolio selection algorithms generally produced higher realized returns, but also higher risks and less consistent portfolio sizes.

Contributions of the thesis

In the **first thesis group**, the contributions are related to detecting overlapping nested community structures. Detailed discussion can be found in Chapters 2 and 3.

I/1. I proposed an edge-based, adjustable heuristic algorithm for detecting overlapping nested communities.

I/2. I introduced an exact algorithm to derive a nestedness graph from networks. With this graph, all overlapping communities can be detected.

I/3. I defined metrics from the output of the overlapping nested community detection algorithms to quantify graph-level nestedness.

I/4. I showed that the new algorithms (unlike most previously found in the literature) do not require graphs to be bipartite.

I/5. I showed that common non-bipartite benchmark graphs show little, while some bipartite ecological graphs show large amounts of nestedness.

In the **second thesis group**, the contributions are related to the applications of hierarchical clustering. Detailed discussion can be found in Chapters 4 and 5.

II/1. I adapted two hierarchical clustering approaches to detect disjoint fully nested subgraphs in networks.

II/2. I created a unified approach to compare top-down and bottom-up algorithms.

II/3. I defined metrics based on the output of the nested hierarchical clustering algorithms to quantify graph-level nestedness.

II/4. I introduced hierarchical clustering-based filtering methods to improve the performance of the Markowitz portfolio selection model by reducing the estimation error (noise) in the covariance matrix.

II/5. I compared multiple hierarchical clustering-based solutions (including filtering algorithms and models) on the portfolio selection problem using real-world datasets, showing that the filtering algorithms are effective in reducing risk estimation errors, while the new methods are capable of achieving higher returns in exchange for higher risks.

Összefoglalás

Ezen disszertáció a hálózattudomány és adatbányászat területén végzett, több, mint négy év munkáját foglalja össze. Fő céljaim között szerepelt, hogy bővítsem a közösségkereső és klaszterező módszerekkel kapcsolatos ismeretek tárházát, különös tekintettel a speciális gráfszerkezetek által alkotott közösségekre. A munka egyik kulcspontja az egymásba ágyazottság (*nestedness*) felderítésére szolgáló közösségkereső algoritmusok létrehozása. Az egymásba ágyazottság egy szigorú követelményeket támasztó, de fontos gráfszerkezet, amelyet korábban nagyrészt csak számszerűsíteni, detektálni pedig csak páros gráfokban tudtunk.

A munka két fő részből áll. Az első részben bevezettem egy heurisztikus és egy egzakt algoritmust, amelyekkel átfedő egymásba ágyazott közösségeket lehet felderíteni gráfokban. Az algoritmusok mellett új metrikákat is definiáltam, amelyekkel az algoritmusok kimenete alapján számszerűsíthető a gráfok egymásba ágyazottsága. A disszertáció egy másik fontos pontja a hierarchikus klaszterezés különböző problémákra való adaptálhatósága, melyet a második részben mutattam be. A klaszterezés egy sokkal egyszerűbben értelmezhető képet tud adni egy gráf közösségszerkezetéről, mint az átfedő közösségkeresés, a csúcsok közötti viszonyok hierarchiája pedig mélyebb betekintést enged a hálózat felépítésébe. Ezeket a koncepciókat a lokális egymásba ágyazottság detektálására, valamint a portfóliókiválasztás problémájára alkalmaztam – mindkettő olyan probléma, amelyet a klaszterezés segítségével meg lehet oldani.

Egymásba ágyazottság, mint átfedő közösségszerkezet

Az I. részben megmutattam, hogy az egymásba ágyazottság egy jól definiált, átfedő közösségszerkezetként is értelmezhető. Az egymásba ágyazottság szigorú strukturális feltételeket támaszt a gráfokban, megkövetelve, hogy a különböző csúcsok szomszédságai részhalmazi viszonyban álljanak. A szakirodalomban számos olyan mutató létezik, amely az egymásba ágyazottság mértékét számszerűsíti, ahogyan olyan algoritmusok is léteznek, amelyek diszjunkt, teljesen egymásba ágyazott részgráfok csúcsait képesek felderíteni, ámbár ezek a módszerek leginkább csak páros

gráfokban működnek. Az egymásba ágyazottság definícióját nem páros gráfokra is kiterjeszthetjük. Két megközelítést is bemutattam átfedő egymásba ágyazott részgráfok felderítésére általános (nem páros) hálózatokban.

A 2. fejezetben bemutattam egy paraméterezhető heurisztikus algoritmust, amely átfedő egymásba ágyazott közösségeket keres egy él-alapú megközelítéssel. Az algoritmus a $\mathrm{nest}(i, j)$ lokális egymásba ágyazottsági mutatót felhasználva iteratívan rendeli hozzá az éleket közösségekhez, miközben egy paraméter állításával lehetővé teszi a pontosság és a teljesítmény egyensúlyának szabályzását. Az algoritmust véletlenszerűen generált és valós páros gráfokon is kiértékeltem, és megmutattam, hogy az algoritmus a korai lépéseiben megtalálja az egymásba ágyazott közösségek egy nagyobb részét, valamint hogy a közösségekben lévő csúcsok és élek száma erősen korrelál a meglévő *diszkrepancia* egymásba ágyazottsági mutatóval.

Folytatva az előző fejezetben elkezdett munkát, a 3. fejezetben egy egzakt algoritmust mutattam be, amellyel az összes (maximális) egymásba ágyazott részgráf felderíthető egy hálózatban. Az algoritmus egy egymásba ágyazottsági segédgráfot épít, és miután eltávolítja belőle a redundanciát, nem csak a közösségek olvashatók le a segédgráfról, hanem a csúcsok láncokban elfoglalt pozíciója alapján azok szerepe is (pontosabban, hogy egy csúcs inkább generalista vagy specialista). Emellett egy olyan algoritmust is bemutattam, amely képes előre ismert egymásba ágyazottsági szerkezettel rendelkező páros gráfokat generálni, lehetővé téve a hasonló, átfedő egymásba ágyazott részgráfokat kereső algoritmusok tesztelését. A gráfok globális egymásba ágyazottságának mérésére egy, az egymásba ágyazottsági gráfon alapuló metrikát is bevezettem. Kísérleteim során a közösségkereső algoritmus azt mutatta, hogy a valós páros gráfokban gyakran sok (kicsi) egymásba ágyazott közösség fordul elő, azaz alacsony az egymásba ágyazottság – ez részben a definíció szigorúságának is köszönhető, – habár néhány ökológiai hálózat nagy mértékű egymásba ágyazottságot mutatott. A valós, gyakran közösségkereső algoritmusok összehasonlítására is használt nem páros gráfokban kicsinek bizonyult az egymásba ágyazottság mértéke, azonban ez a jelenség bizonyos gráfok esetében elkerülendő is lehetett.

Mindkét algoritmusnak megvan az az előnye, hogy általános (nem páros) gráfokban is működik. Az algoritmusok referencia implementációját nyílt forrású kódként elérhetővé tettem.

A hierarchikus klaszterezés alkalmazásai

A II. részben a hierarchikus klaszterezés felhasználását mutattam be olyan problémákra, amelyeket klaszterezéssel direkt vagy indirekt módon megoldhatunk. A hierarchikus klaszterezés nem csak több szintű, különböző felbontású klaszterezést készít, hanem a klaszterezésben szereplő csúcsok hierarchikus viszonyát is kódolja.

A 4. fejezetben két hierarchikus klaszterezési megközelítést dolgoztam át diszjunkt, teljesen egymásba ágyazott részgráfok keresésére. Több metrikát is definiáltam az algoritmusok kimenetei alapján, amelyekkel klaszter és gráf szinten is mérhető az egymásba ágyazottság mértéke a

vizsgált hálózatokban. Az algoritmusokat több, valós hálózatokból álló adathalmazon, különböző távolságmetrikákkal is kiértékeltem, és megállapítottam, hogy a maximális és átlagos kötés távolságmetrikák, valamint az alulról felfelé (*bottom-up*) építkező algoritmus párosítása eredményezte a legnagyobb teljesen egymásba ágyazott klasztereket. Ez mind a páros, mind a nem páros gráfokban így volt.

Az 5. fejezetben a portfóliókiválasztás problémájában alkalmaztam hierarchikus klaszterezési eljárásokat a Markowitz portfóliómodell teljesítményének javítására. Megvalósítottam egy tisztán hierarchikus klaszterezésen alapuló és egy konfigurációs modell alapú kovarianciamátrix-szűrési eljárást a Markowitz-modellben, és összehasonlítottam őket a *Hierarchical Risk Parity* és *Hierarchical Equal Risk Contribution* algoritmusokkal. A különböző módszereket két valós – a Budapesti Értéktőzsde és a Standard and Poor's 500 részvényei alapján készült – adathalmazon is összehasonlítottam. A kísérleteim azt mutatták, hogy a szűrési eljárások nagy mértékben tudták csökkenteni az eredeti Markowitz-modell kockázatbecslési hibáit és a portfólióméretet. A kockázatbecslés elsősorban olyan szituációkban mutatott jelentős javulást, ahol a mintaszám közel volt a részvények számához. Azt is megmutattam, hogy a két új, hierarchikus klaszterezésen alapuló portfóliókiválasztási eljárás általánosságban nagyobb realizált hozamokat tudott produkálni, viszont e mellé magasabb kockázat és kevésbé konzisztens portfólióméretek is társultak.

A disszertáció tézisei

Az **első téziscsoportban** a hozzájárulásaim az I. részhez kapcsolódnak. Ezeket részletesen a 2. és 3. fejezetekben tárgyalom.

I/1. Bemutattam egy élalapú, testreszabható heurisztikus algoritmust átfedő, egymásba ágyazott közösségek keresésére.

I/2. Bemutattam egy teljes algoritmust, amellyel egy egymásba ágyazottsági segédgráf készíthető a hálózatokból. Ezzel a gráffal leolvasható az összes átfedő egymásba ágyazott közösség a bemeneti gráfban.

I/3. Olyan metrikákat definiáltam az algoritmusok kimenete alapján, amelyekkel gráf szinten mérhető az egymásba ágyazottság.

I/4. Megmutattam, hogy az új algoritmusok (a szakirodalomban korábban fellelhető algoritmusokkal szemben) nem feltételezik, hogy a vizsgált hálózatok párosak.

I/5. Megmutattam, hogy a gyakran közösségkeresési tesztgráfként használt nem páros hálózatok alacsony, miközben több páros ökológiai gráf nagy egymásba ágyazottsággal rendelkezik.

A **második téziscsoportban** a hozzájárulásaim a II. részhez kapcsolódnak. Részletes tárgyalásuk a 4. és 5. fejezetekben található.

II/1. Két hierarchikus klaszterező megközelítést dolgoztam át diszjunkt, teljesen egymásba ágyazott részgráfok keresésére.

II/2. Létrehoztam egy egységesített megközelítést a felülről lefelé (*top-down*) és alulról felfelé (*bottom-up*) építkező algoritmusok összehasonlításához.

II/3. Metrikákat definiáltam az egymásba ágyazottságot kereső hierarchikus klaszterező algoritmusok kimenete alapján, amelyekkel gráf szinten mérhető az egymásba ágyazottság.

II/4. Hierarchikus klaszterezésen alapuló szűrési eljárásokat mutattam be, amelyekkel a kovarianciamátrixban felmerülő mérési zaj csökkentésével javítható a Markowitz portfóliókiválasztási modell teljesítménye.

II/5. Különböző hierarchikus klaszterezésen alapuló eljárásokat (köztük szűrési eljárásokat és önálló modelleket) hasonlítottam össze a portfóliókiválasztás problémáján, valós adathalmazokon, és megállapítottam, hogy a szűrési eljárások javítanak a Markowitz-modell teljesítményén, míg az új modellek jobb átlagos hozamokat kínálnak, magasabb kockázatokért cserébe.

Publications

Journal publications

[56] Gera, Imre and London, András. "Detecting and generating overlapping nested communities". In: *Applied Network Science* 8.1 (Aug. 2023). ISSN: 2364-8228. DOI: 10.1007/s41109-023-00575-2

[79] London, András, Gera, Imre, and Bánhelyi, Balázs. "Markowitz Portfolio Selection Using Various Estimators of Expected Returns and Filtering Techniques for Correlation Matrices". In: *Acta Polytechnica Hungarica* 15.1 (Jan. 2018), pp. 217–229. DOI: 10.12700/aph.15.1.2018.1.13

[89] Gera, Imre és London, András. "Gráf alapú dimenzióredukciós heurisztikák részvénypiaci korrelációs mátrixokra". *Alkalmazott Matematikai Lapok* 37.2 (2020. júl.), 211–224. old. DOI: 10.37070/AML.2020.37.2.06

[90] Gera, Imre és London, András. "Hierarchikus klaszterezés és a portfólió-kiválasztás probléma". *SZIGMA Matematikai-közgazdasági folyóirat* 53.1 (2022), 73–88. old.

[91] Gera, Imre and London, András. "Recovering Nested Structures in Networks: An Evaluation of Hierarchical Clustering Techniques". In: *Journal of Complex Networks* (2024). DOI: 10.1093/comnet/cnae039, *(accepted for publication)*

Full papers in conference proceedings

[41] Gera, Imre, London, András, and Pluhár, András. "Greedy algorithm for edge-based nested community detection". In: *2022 IEEE 2nd Conference on Information Technology and Data*

Science (CITDS). 2022, pp. 86–91. DOI: 10.1109/CITDS54976.2022.9914051

Further related publications

[92] Gera, Imre, Bánhelyi, Balázs, and London, András. "Testing the markowitz portfolio optimization method with filtered correlation matrices". In: *Proceedings of the 19th International Multiconference INFORMATION SOCIETY - IS 2016*. Institut Jožef Stefan, 2016, pp. 44–47

[93] Gera, Imre and London, András. "Portfolio selection based on a configuration model and hierarchical clustering for asset graphs". In: *Proceedings of the MATCOS*. vol. 19. 2019

Acknowledgments

First of all, I would like to thank my co-supervisor, András London, for introducing me to the world of research in 2016, when I was still doing my bachelor's degree. Since then, we have worked together on many projects, and his motivation and optimism helped me keep going even in the most difficult times. I would like to thank András Pluhár for his immensely useful insights on how to approach and solve problems, and for his critiques that helped propel my research. Without their supervision, I would never have been able to present this thesis.

I would like to express my gratitude towards my colleagues at the Department of Computational Optimization and the "LANSO" research group. Our weekly meetings on various research topics, the opportunities to present our work in progress, made my work so much more fun. Thank you, Tamás, Bogi, András L., Eszter, Mihály, Viktor, Nóri, Ági, András C., Attila, Zita, Gyöngyvér, Benedek, Viki, Laci, and Ádám.

I would also like to thank my best friend and roommate, Armand for his insights and critical, curious, and thought-provoking questions and ideas. I feel that our near-constant discussions of research and life, and us teaching each other a variety of things we would never have learned on our own, have greatly broadened my perception of the world. Furthermore, I want to thank Szabi and Benedek for their company and help in our little friend group, and for all the fun we had together. I would also like to thank Márk for his ever lasting support. I wish you all the best on your adventures.

Finally, I cannot thank my family, my partner, and all my friends enough for their constant support and care. Thank you, Panni, for being with me on this adventure.

This work would not exist, and I would not be who I am today, without you all.

Imre Gera, 2024